SDN Query Abstraction for Switches as Distributed Databases

By
Kamran Ali Akhtar
NUST201260752MSEECS60012F

Supervisor
Dr. Saad Bin Qaisar
Department of Electrical Engineering

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Information Technology (MS IT)

In
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(August 2016)

Approval

It is certified that the contents and form of the thesis entitled "**SDN Query Abstraction for Switches as Distributed Databases** " submitted by **Kamran Ali Akhtar** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Saad Bin Qaisar**

Signature: _____
Date: _____

Committee Member 1: **Dr. Mian Hamayun**

Signature: _____
Date: _____

Committee Member 2: **Dr. Adnan Khalid Kiani**

Signature: _____
Date: _____

Committee Member 3: **Dr. Arsalan Ahmad**

Signature: _____
Date: _____

Abstract

The languages currently used for describing network policies are not so efficient and well known. The flow entries stored in network switches forwarding tables, can be considered as distributed databases. SQL, originally designed for relational databases, can be used for maintaining these distributed network switches forwarding tables. SQL inherently includes CRUD (Create, Retrieve, Update, and Delete) actions which are also used in network switch forwarding tables. In our work we have proposed that by writing well known SQL instructions using SQL framework on top of OpenFlow, we can leverage the DBMS CRUD actions for OpenFlow enabled network switches distributed databases changes. SQL being a well-known language has number of open source implementations available. We have built a controller that can understand SQL statements and translate them into OpenFlow instructions for applying CRUD actions on the switch flow tables in a network. By using this approach we can benefit from SQL features like ACID (Atomicity, Consistency, Isolation, and Durability) with little or no effort. For the implementation and testing of our proposed solution, we built a controller that accepts SQL instructions as input and deploys respective rules or query data to and from the switches distributed flow tables.

Dedication

I dedicate this thesis to my parents and my family.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Kamran Ali Akhtar**

Signature: _____

Acknowledgment

I am thankful to Allah for his countless blessings throughout my life and also during MS research. All the abilities in me and qualities in my work are due to Allah's grace. Parents are the most valuable individuals in this world. Their support and prayers led me where I am today. Their support is and will be a continuous source of inspiration for me. I am also thankful to my family members and friends for their support and prayers.

I owe sincere gratitude to my thesis supervisor, Dr. Saad Bin Qaisar. His valuable guidelines led me towards completion of research work. His trust in me gave a boost to morale. His guidance helped me to cope the challenges during the research phase. He remained a source of motivation for me during my MS. I am also thankful to my worthy Committee members, Dr. Adeel Baig, Dr. Zahid Anwar, Dr. Muhammad Shahbaz. Dr. Adnan Khalid Kiani and Dr. Arsalan Ahmad for their valuable guidance during the research and for their support of this work.

I am thankful to my friends for their motivation and guidance throughout the research phase. I am obliged to Zeeshan Ali, Shafique ur Rehman, Abdul Basit for all their support and guidance.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Abstractions in SDN

Abstraction layers in software development are a key to overcome complexity. Today's operating systems are most complex software that are built from many components. Basically the operating system is built with abstraction for two reasons. To hide the details of components of one layer from other so that a programmer working on one layer application, need not consider the complexity of other layers and can write a program within a defined scope. The layers have a specific interface with other layers for interaction. It also controls the interaction between different applications so that an application does not interfere with working of other application.

Abstractions are also used in OSI model to overcome the complexity of interaction of one application running on one device with application running on another device. Abstraction is also defined in TCP/IP networking model in which one layered application on one device is communicating with other application running on another device on corresponding layer. It has broken the complexity into many layers. Now each layer has some defined functionality and is responsible for specific tasks. With this approach now each layer can innovate individually and is not dependent on other layer. New techniques of data handling can be developed without propagating development errors to other layers. Each layer is providing a predefined interface and data in agreed form. Each layer performs specific functions on data received from its adjacent layer and forward to other layer.

With the advent of SDN now the control plan and data plan can run on two different physically separate devices. Basically SDN brought this abstraction. The southbound API for controlling data plan have been standardized by ONF as OpenFlow [1] and northbound API like Pyretic [2],

Frenetic [3], JOSN [4]etc. are revolutionizing to catch current needs of SDN. Each layer can now scale and innovate independently. This thing brought agility and scalability in developing modern networks with decreasing capital and operating costs. In this approach a logically centralized controller work with multiple heterogynous devices in data plan. SDN abstraction helps in creating horizontal platforms in which the number of devices can be added in data plan with increasing demands and more controllers can be added to process requests arising from data plan. Open standard of southbound API enables a programmer to write a single program that can control devices from different venders and different specification. Because abstractions break complexity and make things more manageable, in SDN it helps programmers to solve problems efficiently and increase their productivity. SDN gives solutions with efficient code and code reusability. Once a layer is established it handles its tasks independently without affecting other layers. Testers can test functionality of different layers separately and can find and mark bugs and propose solution for these bugs.

The abstractions also brought easiness in programming and especially in SDN, abstraction reduce programming effort and brought new programming paradigm. Abstraction of higher layers in SDN also solve coordination problems between different applications running simultaneously on the controller. Lower abstraction layers solve these problems transparently from upper layer and give a clean interface for programming. POX/NOX [5]are SDN controllers that give operating system like abstraction, for generating OpenFlow messages these requires to write low level OpenFlow structures and transmit these messages directly to the switches connected with the network. Writing application directly on POX controller require knowledge of lower level routines that store Openflow structures. These structures contain information generated in result of different events, packet information and if one has to send a message to the switch for performing any task he/she has to fill specific structure and send this structure to the switch. This approach is like when programmers are doing system programming for Windows like operating system. They are writing handlers for every event that can affect their application by filling different system structures and packing/unpacking information to do different tasks. In higher level programming languages like in visual basic, programmers only change required properties and select events that required modification and write code for only these events, they don't have to care about filling all structures to associate with a specific task.

In SDN new controllers like Floodlight [6], Opendaylight [7] are being built to break programming complexity and separate the concerns in programming different modules. With these advance controllers, programmers are given different abstractions and interface to write application for these

controllers. Now different controllers are built with different aims to solve different issues, some focus on scalability, some on application coordination, efficiency, controller to switch traffic control, security etc.

In computer networks the networking devices like switches, routers, NAT boxes etc. all contain a forwarding table that is populated with different routing algorithms, by listening to the packets flowing through these devices. These forwarding tables are of same kind with difference in information holding capacity. With advent of OpefFlow, now an OpenFlow enabled switch has a standardized format of forwarding table depending upon its version. The forwarding tables of OpenFlow enabled switches are called "flowtables". The SDN controller perform CRUD (Create, Retrieve, Update, and Delete) operations on the tables. These CRUD operations are same as those performed by DBMS on database tables.

1.2 Motivation

As all network devices have forwarding tables to perform packet forwarding and typical network control algorithms are running on these devices to populate these tables. For having a controlled behavior of the network these networked devices are configured by network administrators by using management consoles. These management consoles or management software are vender dependent. It is a very tedious task to configure a network of heterogeneous devices from different venders. For a large network that requires changing network management policy manual reconfiguration of devices do not work. In these conditions management scripts are used, but dynamic management policy cannot be deployed on network switch using these techniques. With SDN that separates control plan from data plan as shown in figure 1.1 now a logically centralized controller can program/configure heterogynous devices from different venders. SDN controller works with OpenFlow enabled switches. These switches now can be programmed with SDN controller to act like different devices like simple hub, layer 2 switch, router, NAT box, firewall, load balancer etc. The SDN controller basically performs CRUD operations on flowtables of network switches to give them behavior like simple hub, layer 2 switch, router, NAT box, firewall, load balancer etc.These CRUD operations are also performed in HTTP [8] like requests.

If we see from figure 1.2 in database domain the Database Management System (DBMS) and database are running on two physically separate machines. The database is maintaining different tables that contain actual data. The DBMS is performing CRUD operations on these tables using network connection. Programs that required operations on data contained in database

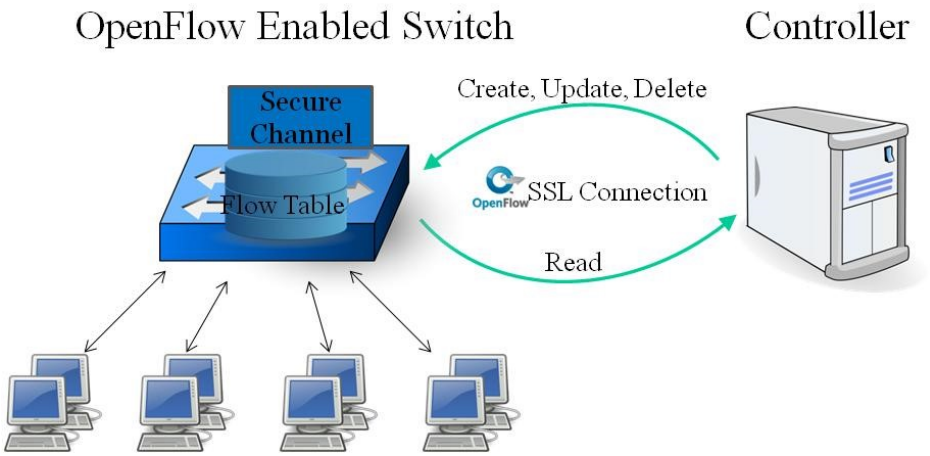

Figure 1.1: OpenFlow Switches and Controller.

tables use Sequential Query Language (SQL) to ask DBMS to perform these operation. The DBMS schedule these operations as transactions and execute these operations on database.

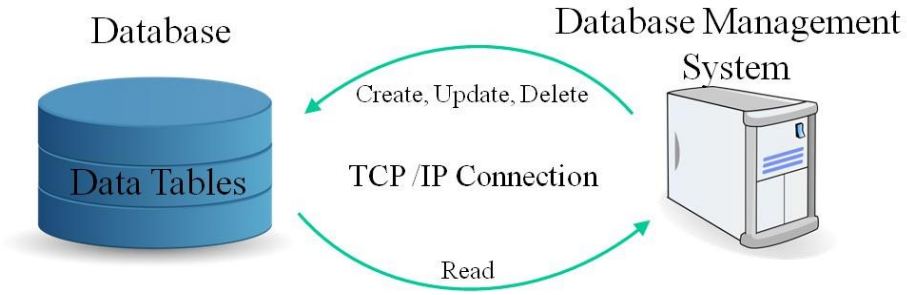

Figure 1.2: Database Management System (DBMS) Architecture.

Here the analogy can be seen; both the Databases and SDN use CRUD architecture. Both use Network connection to communicate with the data source, in case of DBMS the database uses servers to hold data and in case of SDN network switches are used to store flowtables. It is very interesting to see that database and SDNs are working in the same way and performing two very different tasks. The SDN is managing a network by deploying network management policy on underlying devices and databases are storing huge amount of data on data servers. But in SDN other controllers do not use SQL instead they are using "SQL like functions" or using and developing other higher level languages/API eg. JSON [4], REST API [9]. The examples of controllers that give SQL like functions in application programming are

Frenetic's sub-language for network query [3], [10] ,Flowlog [11] etc. Instead using SQL like syntax if we use SQL language that has evolved over time and is been standardized, we can efficiently deploy network management policy with SQL. The emergence of SQL and DBMS approaches with SDN will not only give the easiness to the programmers, it will also help to leverage database ACID properties and other transection processing properties.

Therefore, development of middleware API like SQL is highly required. For assessing ease of use, learning and preference of SQL over OpenFlow / Pyretic and interest from network application programmers and software developers we conducted a survey. In this survey we included over 73 programmers, all of them belongs at least MS study level and from research community. About half of the programmers had taken SDN course where they learn how to program for POX SDN controller. These programmers are researching in EE, CS and IT fields. All have studied data communication/networking courses. The survey Performa is attached in Annexure A and the results are discussed as under.

This survey was divided into two parts in first part some basic information about programmers was gathered. In second part data about their preference for using SQL, POX/OpenFlow and Pyretic was collected. For this, in first question in first part, the programmers were shown sample code snippets of MAC based firewall, written in SQL, POX/OpenFlow and Pyretic languages. The programmers compared three languages and chose their language of interest. The survey results are shown in the form of pie chart below.

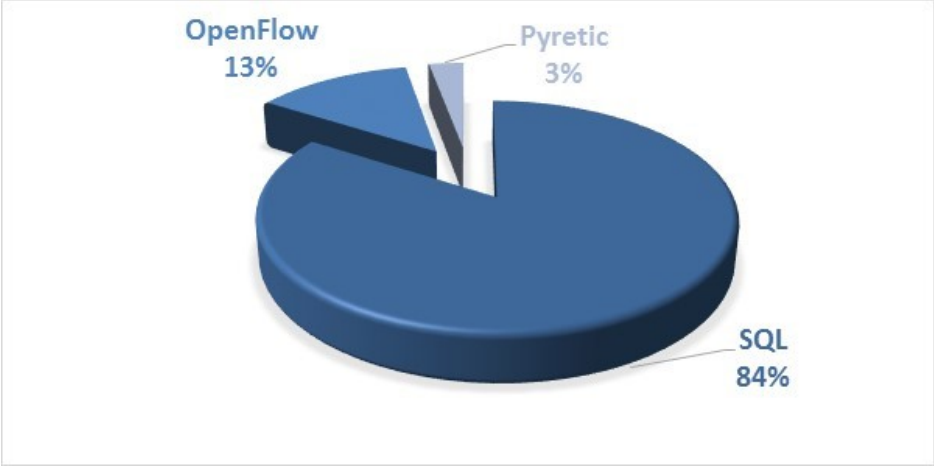

Figure 1.3: Preference for writing MAC based firewall.

It can be seen from pie chart that 84% preferred SQL because of its clean, easy and simple syntax.

Next they were posed question "With SDN, the forwarding tables of switches can be seen as database tables and with our test framework, SQL can be used for writing in these tables. If you want to write an SDN application then what language would you prefer among SQL, OpenFlow and Pyretic?". In the survey 82% programmers opted for SQL.

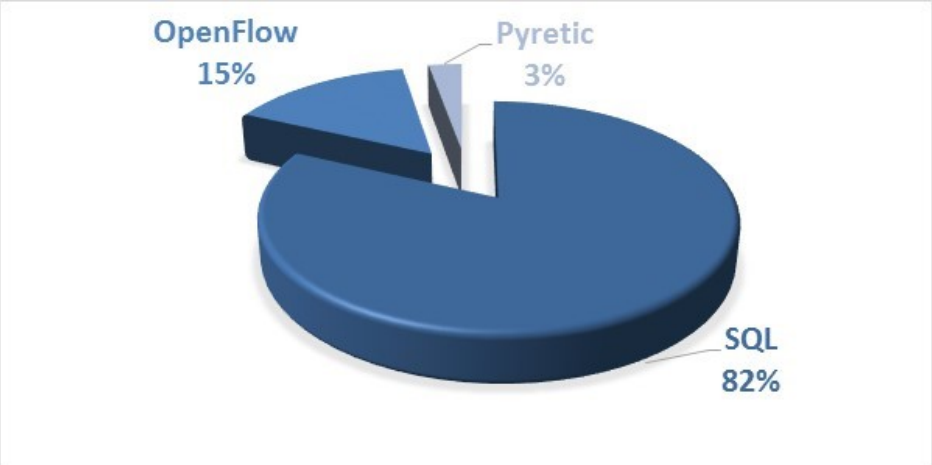

Figure 1.4: Preference for Using OpenFlow, Pyretic, SQL.

When we asked about easiness in learning from these languages we learned that it share with respect to learning is about 50%. For summarizing the results from this question we had taken average of marks given by the user for three controllers, the data is shown in Annexure B.

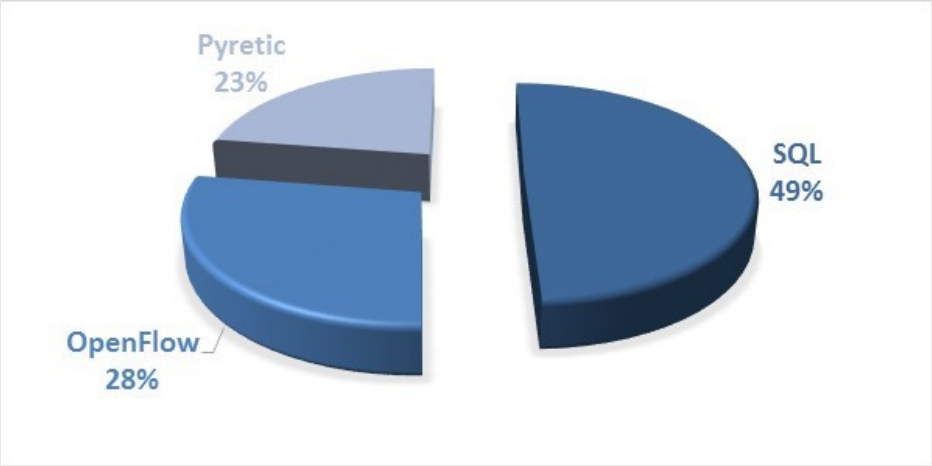

Figure 1.5: Give marks on scale of 100, how it is easy learning.

To the best of our knowledge there is no forum available that gives the

statistics about how many programmers have SQL knowledge and how many have POX/OpenFlow or Pyretic knowledge. To find this in our local domain, the following question was posed to the programmers at our SEECS campus; "In your thinking what % of all programmers have knowledge about SQL, POX/OpenFlow or Pyretic?" Data from their answers shown that about 66% programmers from all over programming community has SQL knowledge and do not need to lean it from root.

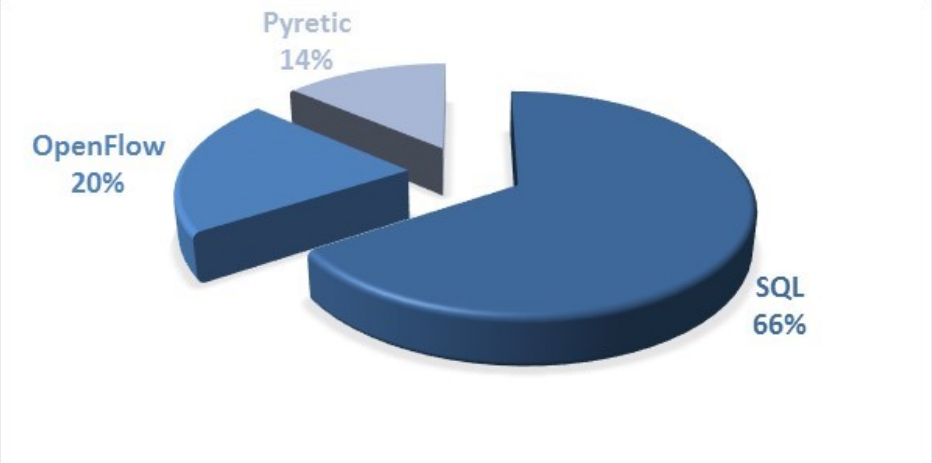

Figure 1.6: Programmers having knowledge of the 3 languages.

The above survey shows that SQL, as a northbound API, provides significant ease of use as compared with the traditional coding models in contemporary controllers. As more then 65% programmers have SQL knowledge, when they come to network programming they would find it very helpful. Considering the switches as database components reduces software design complexity. Our design reduces lines of code by approximately 20% (on average) as compared to the direct OpenFlow message representation in POX, so more programmers opted to program using SQL.

1.3 Problem Statement

SQL is a standardized and well-known language. SDN controllers already use functions that look like SQL instructions, for deploying network management policy. The learning curve of SQL is steep. With SQL emergence in SDN we can leverage a huge amount of verified code from database community. This can enable SDN network programmers to benefit from functionalities defined in databases. Especially as in database the transaction of one table

state to another, the Network management policy is also changing from one state to another but it requires reconfiguration or deployments on all network devices, means changes in all network devices flowtables.

In this study the suitability of SQL for efficient deployment of network management policy using database abstraction will be investigated. As database tables are analogous to network switches' flowtables, the suitability of SQL to change network switch flowtables will be investigated.

1.4 Proposed Solution

To verify the suitability of SQL for efficient deployment of network management policy, a SQSDN controller will be built. Different network management policy applications will be selected and these applications will be written for SQSDN controller. These SQL and non SQL application will be compared in terms of lines of code, and executed lines of code in different scenarios.

1.4.1 Expected Outcomes

By successfully implementing our proposed solution we expect that SQL can directly be used in SDN controllers. SQL is a standardized well known language as compared to other SDN languages. We want to propose a verified code base, from database domain, having various database functionalities for SDN domain. These functionalities can be enabled in SDN with little effort.

We also expect 10% to 25% reduction in lines of code and a significant reduction of executed lines of code, since a flow deployment instruction is repeatedly executed whenever a flow deployment is required or a change in flow-table is to be done.

1.5 Thesis Organization

This thesis is organized in mainly 5 chapters

Chapter 1 contains introduction to the SDN domain, motivations for building abstractions, motivation for building an SQL based abstraction and then propose a solution and presents our contribution in this domain.

Chapter 2 gives literature review of SDN and database fields. Defines the key words used in this thesis.

Chapter 3 elaborate the high level architecture of proposed idea, basic design features and the implementation details of proposed design, test applications, testing parameters and method of testing.

Chapter 4 has the test-bed details, experiment results and evaluation of the results.

Chapter 5 presents the conclusion and suggested future work directions.

Bibliography section mentions the main sources of information and the annexure section have supporting data used for building graphs in this thesis.

Chapter 2

Literature Review

2.1 Background

PLAN [12] is a language building active networks when only conventional network equipment exist. It follows strict functional rule for writing programs. For remotely execution of PLAN based programs it defined certain primitives. It attains its functionality by generating packets that carry programs or instructions and change these packet headers and call service routine written in other languages.

Before standardizing OpenFlow to solve the problem of network management that was caused by complexity in control and management plans 4D [13] approach was proposed. They have proposed the refactoring of control and management into three principle objectives; network level objectives, full network views and direct control. It divided its architecture into four planes i.e. data, discovery, dissemination and decision. It proposed to separate decision logic from the underlying protocols to control the communication between network elements. The decision plan maintains the higher level objectives and directs the configurations of network elements, how they process and forward packets. The monitoring/measurement data collected by these network elements like switches and routers help the decision plan to control network state. In 4D architecture the network decisions are taken in and logically centralized server. The network elements only run the discovery protocols and follow the instructions from decision plan. The discovery plan creates the logical identifiers for physical network elements and dissemination plan provide efficient communication between physical network elements and decision plan. The result of this technique is the auto configuration of network elements. It has issues in response time from decision elements as the network elements scale 4D become unstable. it is not suitable enterprise

level networks.

The modern networks relay heavily on centralized controller. All the packets that do not match any flow, are forwarded to control plan. In enterprise networks centralized controllers are heavily loaded with packets coming from switches. The enterprise administrators have to specify a fine-grain network policy to process these packets on data plan. The DIFANE [14] solution proposes a technique that partition flow rules on network switches, so that maximum network traffic can be processed on data pan and traffic between switch-controller is reduced. It deploy wildcard rules that process more traffic on data plan on ingress switches and partition the traffic in to data path and to authority switches. The flow rules deployed on authority switches selectively send packets to control for new flow deployments. Using this approach larger network can be controlled with fine-grained policies.

Onix [15] and Hyperflow [16]are presented for meeting the requirements scalability in large production network. Both are logically centralized by physically distributed controller. Onix gives a global view of network to application and provides an API for writing network applications that run on global prospective is suffered from issues like response time and unresponsive behavior. The Hyperflow in contrast make localize decisions and minimize the response time. Due to divided control plan it is resilient to component failures and network portioning. But it has only local network view for controlling network.

DevoFlow [17]breaks the coupling between control and global visibility, in a way that it maintains a useful amount of visibility without imposing unnecessary costs, it uses 10 53 times fewer flow table entries at an average switch and uses 10 42 times fewer control messages. DevoFlow experimented with both proposing control plan changes and data plan changes. Cloning of flow entries are proposed for decreasing flow setup messages, flow entries are decrease by using wildcards. Counters are approximated for decreasing switch to controller traffic. By using this approach the limitation of scalability in OpenFlow is addressed and more expandable solution is offered.

To resolve conflicts in sheared resource, whether a device has an authority to perform a task or not and other decision making is accomplished using hierarchical policies in conventional networks. Because the floatables of commodity switches deploy flow to forward packets and do not understand hierarchies this work proposed hierarchical flowtables [18] (HFT). With hierarchical policies in SDN, the flow roured are deployed on switches in hierarchic manner. The flow tables in different switches work in a hierarchy like it is a HFT and each level switch perform different operation. By using this technique the decisions are distributed over different levels of switches and the decision distribution is performed by translating the network into a tree

of flow tables.

Corybantic [19] proposed a design for modular composition of network management applications that are competing for same resource. It tries to optimize the objective functions to maximize the performance of the controller. For this in Corybantic different modules are build that take care of allocation of some objective function like bandwidth, power control and VM migration. It can be used for IaaS ((Infrastructure as-a Service) in cloud orchestration tools.

ONOS [20] is a distributed and logically centralized SDN operating system. The motivations behind building this controller architecture are scalability, performance and availability requirements from a larger network. It gives the operating system level abstraction of a global network view, fault tolerance with distributed but centralized controller. In ONOS architecture they run OF managers using Floodlight controller on physically separate machines and build a network view by collecting distributed key-value from different machines and from a graph database. It used a distributed registry that manages relationship between controllers and switch and react on attachment or detachment of switches. If a switch losses connection with one distributed registry instance, some other distributed registry instance will take its control. ONOS abstraction gives a global network view that empowers the application to deploy flow rules for full or partial path between source and destination. With this an application can use global view to find host location and its usage data to accomplish required tasks.

For solving different issues like efficiency, scalability, programmability, modularity, application composition and distributed control plan different controllers [21] are proposed. Each has its pros and cons. Here we review some of controllers and other architectures for their programming interface.

2.1.1 Software defined Networking

The increasing number of mobile devices that are fetching data from datacenters, distributed datacenters, cloud services and server virtualization are driving force for new innovative and agile network architecture. The conventional networks that are building from Ethernet-switches connected in tree like structure make sense for client-server applications. Today's dynamic computing, enterprise level datacenters and multi services carrier environments need new network architecture. The data traffic patterns in datacenters have been changed. Server is running as instance of virtual machines and migrating in different datacenters on the fly. The users are acquiring computational resources from enterprises with cloud orchestration tools. Enterprise businesses want their users to access their infrastructure and application any-

where from globe. Datacenters are handling massive amount of data.

These requirements are impossible to be met with conventional networking techniques. As the services and demands of networks are increasing the complexity of networks is increasing. The management of these networks becoming more difficult and cause inconsistent policies [3], [2]. The rapid growth in bandwidth required more network resource and conventional networks fails to accommodate this change. The vender dependence is blocking enterprises to use new services and capabilities. Businesses are seeking for open interfaces so that they can build agile networks.

Ethane [22]frame work also given network administrators to deploy a flow based fine-grained network management policy on network befor concept of SDN. The SDN is a new emerging network architecture, [23] that decoupled control plan from data plan. The control plan is now become directly programmable. With OpenFlow [1], [24] data plans open standard business are free from enterprise dependence. Anyone can write controller for meeting his specific requirements. Industry is building controller platforms where network applicant can be written. These are giving programming abstraction to the programmers.

The need if Information Centric Networking (ICN) is increasing. The network should change itself as changing information patters. In [25]study it is shown that SDN control and function can be used for adapting change as change in information patters. this approach is using existing IP networks and with the help of SDN performing ICN routing.

As OpenFlow has separated control plan and data plan it has enabled to build better management tools. The OFConfig protocol [24] uses the OF-Config 1.1 for network management. In this technique they translated parameters from OFConfig to OVSDB for monitoring and controlling OpenvSwitch (OVS).

In this paper [26] a scenario of VM migration is discussed. During migration process there may be packet losses or service disruption or bandwidth chocking and cause service agreements problems. Authors proposed that with SDN a controller will take decisions where to install or remove packet forwarding rules in a network solve this problem. It ensures that the required bandwidth is available on the switches where the VM is being migrated and the migration process may not induce any packet loops or higher latency. Working an abstract network view, the controller architecture has solved this resource allocation problem.

2.1.2 Database

In this early database techniques development paper Jeffrey [27] proposed that the databases should be updated with database integrity constraints. In this paper a methodology is proposed for updated in databases. Its reason to change the old theory is based upon the argument that, if a small change in theory is required then the update should be minimal and if several changes are required then the old theory should be replaced with new theory. It states that facts in database are of differed forms: derived and facts based on actual information. The update should see how the logical database is formed.

Active-database System [28] the integration of production rules like triggers, protection, alerts and version control in to databases are proposed. With these, a database can change dynamically and become an active application. In some SDN applications, triggers are used for communication with databases.

The schema like SQLGraph [29] translate its instructions in to SQL relational model to take its benefits like locking, security, concurrency, query optimization and ACID properties. It uses Gremlin language that works with graph database and support CRUD operation, run over a query processing framework to translate its queries into SQL queries. It translate Gremlin query into single SQL query to eliminate client server protocol overhead and efficiently deploy its query using relational database engine. RDF is another language to query graphs uses SPARQL. To optimize the SPARQL it is proposed in many works [30], [31] to translate it into SQL language.

2.2 Related Work

For implementing a high level policy on in a typical network, it contains heterogeneous set of devices that include switches, firewalls, NAT, middle boxes [32]and load balancers. These devices need to program separately. For managing these devices distributed scripting is required that is error prone. With advent of OpenFlow [1]that is potential enabler of SDN the dumb switches can be programmed to achieve functionalities like router, security device or load balance. In this approach a logically centralized controller that is an x86 based computer implement the high level policy. It computes routes between nodes and deploying appropriate flow entries in the underlying switch table. These flow entries have two parts, match and action. According to OpenFlow standard [33] switch collect the actions in the action set after matching the packets against various fields and perform actions like forward,

flood or drop the packet according to these actions in the action list. The switch also collects the statistics by updating counters associated with that flow for network traffic analysis and management.

Now defining policy directly in NOX [5] and Beacon [34] is difficult. These controllers install flows on per event basis, in these programmer based on these controllers, has to care about installing and how flows affect other flows. Flow management language (FMS) was designed specifically for defining network policies. [35] it simplified the task of coding for defining a policy. It works fine for writing new policies and has issues in writing subsequent policies that required additional code outside FML.

Changing networks management policy is common to produce instability in networks. A computer network can become unusable if network management policy is not correctly installed on it. Even in process of deployment of new network management policy networks become unresponsive. Different policies and techniques have been proposed to solve this problem. In these works [36], [37] an update abstraction proposed that works on different run-times system.

The network application designers are moving towards modular programs that are executing complex network function the dependencies become unavoidable. The concurrent operations of these applications that may be programmed to run independently may cause incorrect network behavior. The Maestro [38]controller is giving an abstraction so that network application can deploy their required flows on network revives without affection rules from other languages. It used directed acyclic graphs (DAGs) to solve the concurrency problem

NetCore [39] is a high level declarative language that has compositional expressive and formal semantics. The language is used for expressing packet forwarding policies. NetCore has a compiler to compile network policies and a new run-time system. Its run-time is responsible for issuing commands to get traffic statistics and installation of rules on the network switches.

For improving network management with SDN Procera [40]framework proposed. It basically addressed three problems in network management. The frequent changes with respect to the user behavior that cause change in network conditions and state changes, use of high level language for having batter visibility of network conditions to control and diagnose a problem network policy and how a network can be configure with concurrent policies. It uses high-level functional-language to respond network events. With its functional behavior it can reconfigure network devices to data usage, traffic flow, time and authentication status.

In this work [41] the authors tried to solve the network configuration problem that occurs when more than one policy are implemented or changed

concurrently. They used abstract topology and proposed an abstract packet processing model with pyretic language that is using sequential composition operator to process packets. This scheme gives the illusion that a packet is processed from one policy and then from other and the operation of two policies do not overwriting the changes by other. The order of the two policies implementation would be predetermined. They used virtual fields for identifying the level of policy being implemented by Pyretic [2] language.

Yuki [42] has proposed an architecture that uses logical and physical database to store network state. This architect is proposed to solve requirement of asynchronous updates in network. In this approach they stored has two type to table to store information, one is called physical information table, is storing properties related with physical network, like: propagation delay and link capacity. Other is called intermediate code tables are storing properties related with network working like; flow path of a type of packets and path IDs (VLAN tags). In this different application are running as configuration engines. The configuration engines are running independently from each other and updating the logical databases. The network administrator active different configuration engines to achieve his desired behavior. The configuration engines can be added running controller. The implementation of this architecture can run on different controllers and a separate conversion driver is required each controller.

Peng proposed a Stateman [43] service as an abstraction, so that they can run transparently from each other. The service manages network state and its changes and other network application operate independently from each other. Network management application can read and propose any change in network state, according to their requirement. The Stateman collect all the changes from different applications and merge these changes to form a target state and update the network to the target state. This service can be used as an abstraction for coordination between different network management applications.

Chapter 3

Design and Implementation

As in cloud computing where data centers running at different geographical locations provide the web services and data services. Users from different parts of world can acquire resources like web servers, leased dedicated servers and custom networks. It requires the use of Cloud Orchestration Tools (COTs) like OpenStack, CoudStack, Eucalyptus, and Abiquo etc. These tools help users to get their requirements fulfilled. Some COTs have graphical user interfaces to get user requirements and give full control of the required resources. These COTs use higher level programming languages and APIs to tell underlying controllers to reconfigure there devices to get required functionality. Now these devices can be physical devices or software or virtual devices like Open vSwitch, virtual routing devices etc., which are in network virtualization. Frenetic, Pyretic like languages and REST, JSON like APIs are used to write programs for SDN controllers or to communicate requirements to the SDN controller. At this layer these Frenetic, Pyretic like languages are used for Network Function Virtualization.

These languages and APIs are evolving and more efficient methods are being invented to have better agility and dynamic behavior. As the Frenetic [3], Pyretic [2] and Kinetic [44] like languages use a Runtime System that is written on top of NOX/POX controllers and give a programming abstraction for higher layers. Other controllers like Floodlight [6] use REST, JSON APIs to give interface for external application to communicate with controller application to convey their requirements. For innovation and ease of programming at this layer SQL database abstraction is proposed. COTs can directly generate SQL instruction or can use a middle level abstraction like Pyretic to use SQL for deployment of network policy or configure network to meet the requirements.

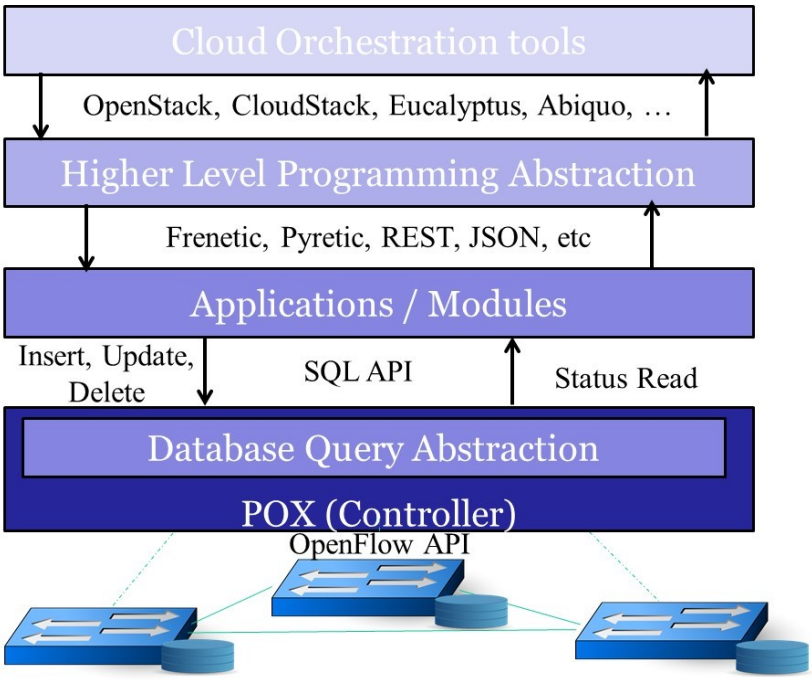

Figure 3.1: High Level Architecture of SQSDN Controller.

3.1 Architecture

For deployment of network management policy we built a SQSDN controller. Its basic architecture is shown bellow. There are two main modules of Controller. One module contains a DBMS implementation. We had included SQLite3 code base for this purpose and the other is coordination module that handles events, data from application and give API for sending OpenFlow messages and receiving SQL statements. On top of these modules user have to write application that generate SQL statements to do CRUD operation on flowtables.

We have used the functionality of POX controller as a SDN operating system as it gives OS level abstraction. Our implementation's modules are adding a programming abstraction over operating system abstraction. On top of programming level abstraction programmers write their application.

As SQL is giving database abstraction for distributed tables over network switches, we had included a full implementation of DBMS in our architecture.

From DBMS implementation we used its code base to get our required functionalities. For getting more functionalities from DBMS in SDN, now one have to only connect modules with SDN, as code is already included in our implementation. The DBMS implementation is shown in figure as below.

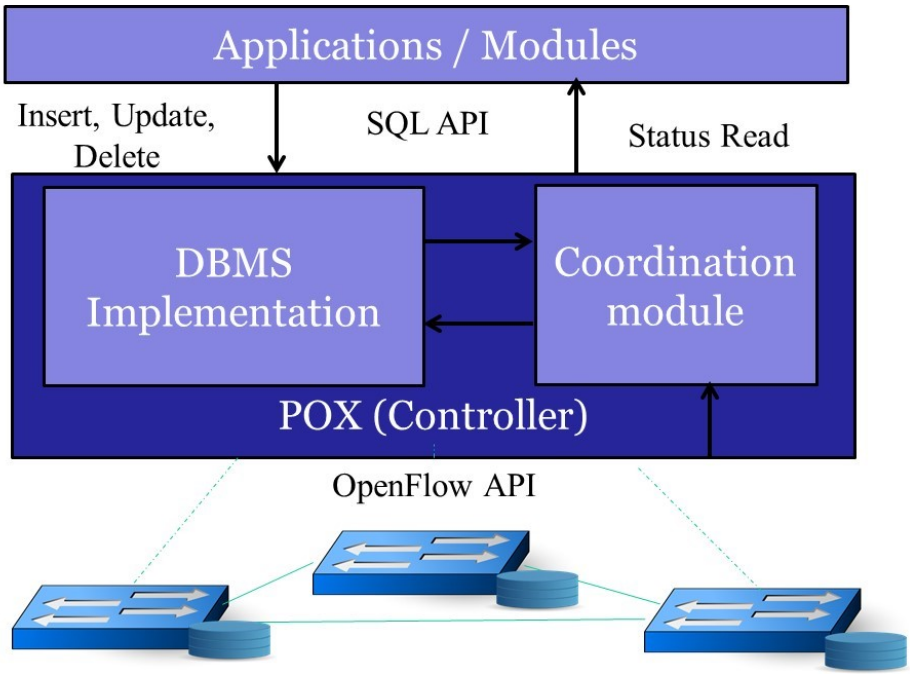

Figure 3.2: Architecture of SQSDN Controller.

As from figure, it also has built in form of modules and abstractions. Each abstraction is giving specific API's for interaction with other abstraction.

The SQL instructions are mainly of two types Data definition language (DDL) and Data Modification language (DML) [45]. DDL is concern with database schema i.e. number and type of tables, field and data types of these fields. The DML is concerned with SDN. As by this type of instructions, SQL perform CRUD operation. The instructions used for these operations are insert, select, update and delete/drop. These SQL instructions are received on DBMS interface that passes these instructions to SQL's command processor.

The command processor send these instructions to tokenize for recognizing their type and then these tokens are send to parser. The parser module check the code for Symantec and syntax errors and recognize its function. The parser calls appropriate code from generator module to generate its code. As the SQLite is an independent DBMS that can run on any operation system the code generated from its implementation run on a virtual machine that generate operating system understandable code. This code is having ACID properties and use pager to achieve these properties. The B-tree module handles the data for saving on operating system file level abstraction.

In our implementation we have used functionalities up to parser module and leave the inclusion of other properties for future work. We have written

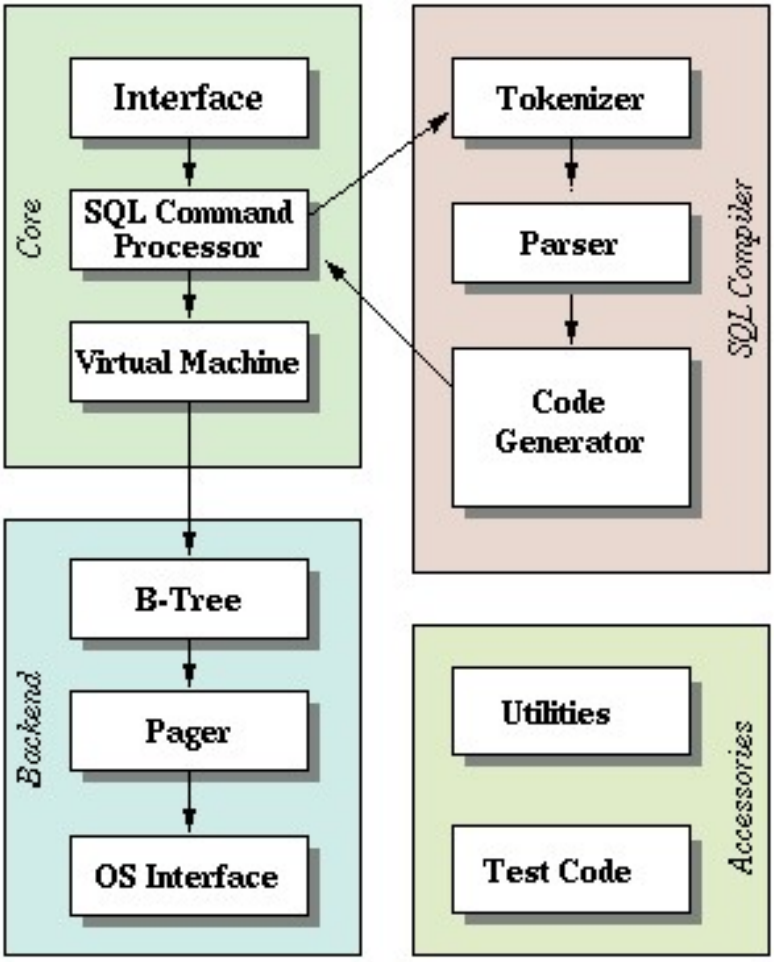

Figure 3.3: Internals of DBMS.

our own code generator module that does not send control back to command processor module. It generates the code for POX operating system that send's packets to modify flowtables on network switches.

3.2 Design

The detail design of SQSDN controller will elaborate the working of this implementation. As shown in figure below; the programmers can write application by considering network switches as distributed database tables, they query these tables with database abstraction i.e SQL. This SQL API performs CRUD operations on these database tables using SQSDN controller. Here in POX, SQL uses the OpenFlow that is running as lower level API like other higher level languages C, FORTRAN use assembly as lower level language or object code. The SQL statements from SDN Applications are received at coordination module that sends these instructions to core module. This is where DBMS implementation lies. It performs DBMS operation tokenization, parsing and code generation in this module and finally send OpenFlow message using POX abstraction to network switches.

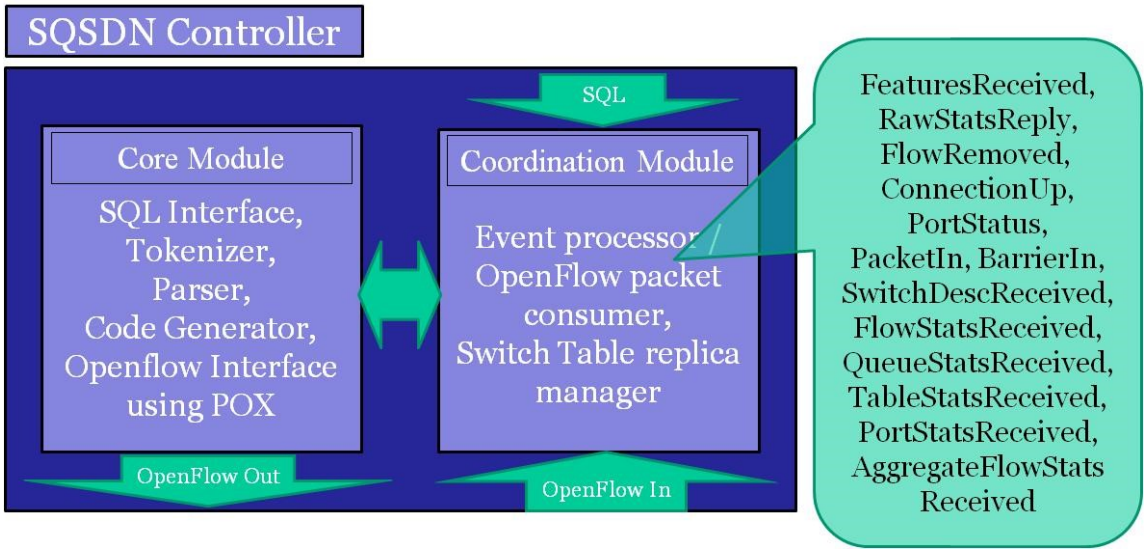

Figure 3.4: Detailed Design of SQSDN.

In response to these flow deployments switches can generate messages, these messages are consumed by event processor in coordination module. The coordination module is also managing a replica of switches for getting statistics from these switches.

3.2.1 Single Query Execution Model

The single query execution model as shown in figure below will help in explaining how the SQL query is executed in SQSDN controller. When a SQL query is raised it is received at coordination module which sends it to core database module that parse that query and understand it's meaning, it also generate friendly error messages to tell programmer where there is the problem in this query. According to its meaning it identifies its CRUD operation and call code generation module that generate flow mod messages for OpenFlow. Then these messages are sent to the switch using POX operation level abstraction.

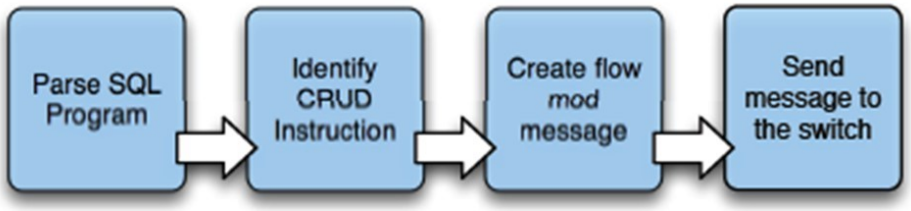

Figure 3.5: Stages for processing the SQL.

3.2.2 Testing Parameters

As we are building an abstraction to include a high level language (SQL) as an API in SDN. SQL is a language of relational databases and is considered as higher level language then C, C++, Visual Basic. It reduces the line of code to implement CRUD operation on network switches.

As the flow deployment instructions are repeatedly called in different programming scenarios according to requirements with varying parameters, we will also test the number of times the flow deployment instructions are used.

We will compare the line of code (LOC) in writing SQL SDN applications with application written in POX. We also compare the flow deployment instructions that are executed in performing different tasks using both techniques to get an idea about the executed line of code (ELOC).

3.2.3 Test Scenarios

We have selected two cases from simple application and two from advance level application to compare LOC and ELOC. We borrowed Hub, Layer2 forwarding (LSW) application from POX distribution and taken firewall and

Intrusion detection and mitigation system from developers from universities and written our own respective application to compare with these application

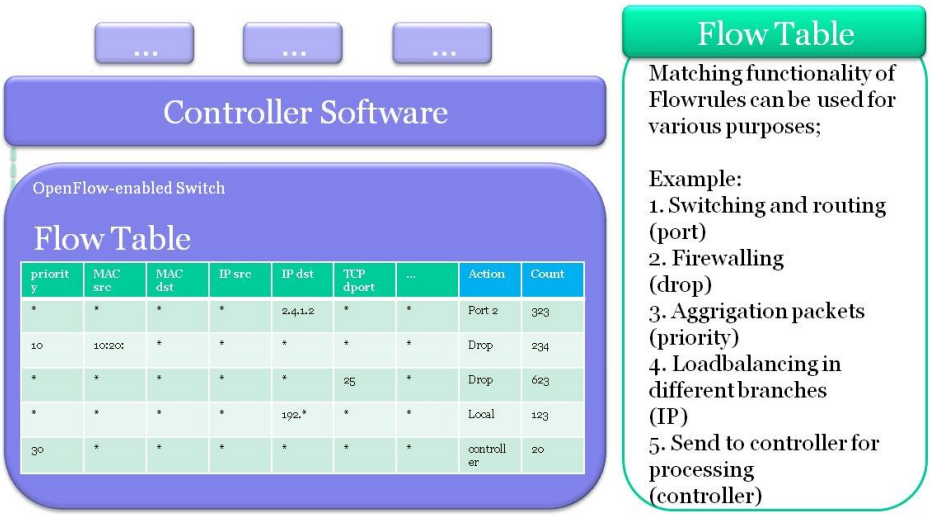

Figure 3.6: Flexibly in Setting Flow-Based Control.

Our SQSDN application works by taking flow table as shown in above figure. It can be seen different rules have different meaning and by deploying appropriate flow rule we can convert a OpenFlow enabled switch to any device like switch, router, firewall, load balancer and to forward packets to any other port for deep packet inspection.

Chapter 4

Experimental Results and Evaluation

For implementation and results generation a test bed deployment was used. For this purpose a computer system with Core i3 1.7GHz processor and 64bit operating system was utilized. Hyper V technology was enabled on it, so that the virtual environment can run as truly as 64 bit environment. We have used Oracle VM virtual box for creating virtual environment and managing virtual machines resources. We installed Ubuntu 14.04.1 LTS LINUX operating system on the test bed virtual machine with wireshark [46]as network monitoring tool and IPERF [47] for traffic generation tool. For emulating network behavior we have installed mininet [48]version 2.1.0 and for getting SDN OpenFlow enabled switching functionality, we have ovs-vswitch [49](Open vSwitch) 2.3.90 that enables us to use OpenFlow [33]versions 0x1:0x1 on network switches.

For development purpose we have used POX latest version available on github. We have developed a framework naming SQSDN on top of POX network controller operating system.

4.1 Line of Code comparison

Line of code comparison of POX and SQSDN application hub, firewall learning switch and intrusion detection and mitigation system can be seen in table and depicted below. In the table it can be seen that writing Hub code in POX it used 22 lines and SQSDN used 9 which is more than 50% less as compared to POX. In case of firewall, POX application consumed 36 lines and SQSDN based application consumed 29 lines that are 20% less. For LSW, POX has 96 lines of code as compared with SQSDN that has 85 lines

of code with 12% decrease. In case of IDMS, POX based application used 125 lines of code as compared 87 lines with SQSDN which is 30% decrease.

Table 4.1: Line of code comparison.

Controllers \ Application	Hub	Firewall	LSW	IDMS
POX	22	36	96	125
SQSDN	9	29	85	87

These statistics are also shown in graph, here it can be seen that on average the SQSDN based application reduced 28% of line of code for building network application.

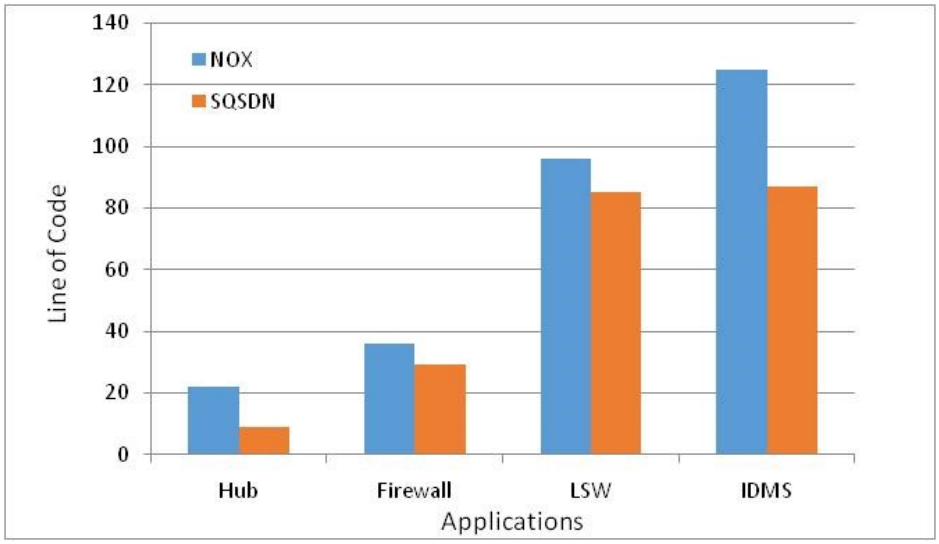

Figure 4.1: Line of Code Comparison of Different Applications.

4.2 Executed Line of Code

The network applications mostly do CRUD operations recursively and are making flow modification in one way or the other. It is presented in the figure below that if the selected four example applications make 100 flow deployments, how many LOCs will be executed in flow deployment instructions? We have gathered statistics for POX and SQSDN implementations and plotted in this figure.

From table, it can be seen that POX Hub will execute 300 lines of code as compared with SQSDN that do the same task by executing 200 lines. LSW

POX application executed 700 lines of code as compared with SQSDN based LSW that executed 500 lines of code in deployment of 100 flows on a switch. Firewall POX application executed 700 LOC and SQSDN executed 200 LOC. The IDMS POX application executed 700 LOC as compared with SQSDN that executed 700 LOC to deploy 100 flows in switch flowtable. Here it should be noted that as more matching instruction are used for packet forwarding the difference in ELOC's would be more prominent. In case on Hub it just forwards messages, LSW matches 5 fields and for Firewall and IDMS only one field is matched.

Table 4.2: Executed Line of code comparison .

Controllers \ Application	Hub	Firewall	LSW	IDMS
POX	300	700	700	700
SQSDN	200	500	200	200

From graph it can be seen that as complex application are written in SQSDN, it will reduce more LOC as compared with POX based application and in case of Executable Lines of Codes (ELOC) the difference will be more significant.

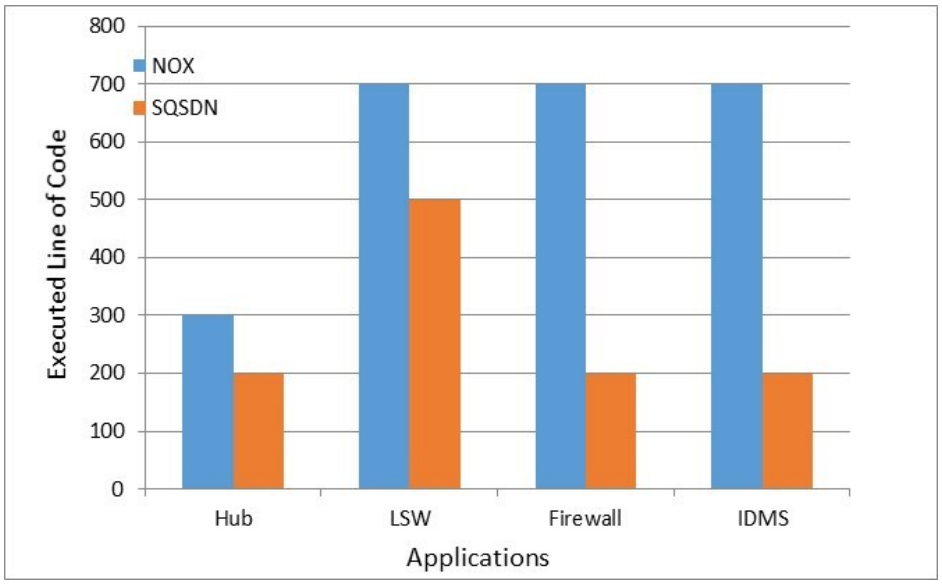

Figure 4.2: ELOC Comparasion for 100 Flow deployments.

In the results shown in figure 4-3 we have used firewall application scenario for both implementations (SQSDN and NOX). We gradually changed the number of blocking rules from 1 to 25 and recorded the ELOCs. For executing

this scenario we had emulated a network of 50 nodes using mininet and tested the network for "All Pair's Connectivity". We changed the blocking rules by writing 1 rule in blocking policy file and repeating the experiment by varying this value up to 25.

From Figure 4-3 it can be seen that with the use of SQSDN there is increase in executed lines of code with increase in blocking rules but this increase is less in SQSDN as compared with NOX implementation. The factor of decreasing ELOC's will be very beneficial in larger networks, therefore large networks can significantly benefit by deploying SQSDN instead of NOX.

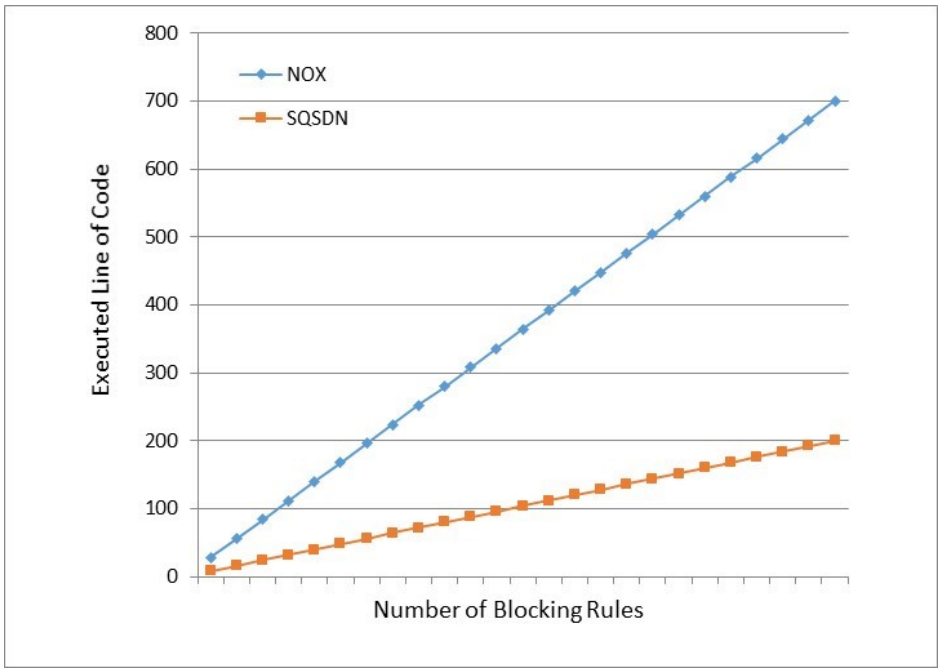

Figure 4.3: ELOC Firewall Flow deployments by varying blocking rules.

In figure 4-4 the results from LSW applications are displayed. In this scenario we increased the number of hosts from 2 to 50 and ran all pair connectivity test. In this test we gathered statistics by measuring number of deployed flows, in a process where all hosts checks the connectivity with other hosts available in the network. For this ping command is used that is using ICMP messages for testing connectivity.

In this figure 4-4 the results show that as the number of hosts increases these is exponential increase in flows deployed and hence the number of ELOCs are also increasing exponentially for POX application. From this it can be concluded that the SQSDN implementation is performing much better than POX implementation.

For university campus network where thousands of nodes are communication with each other, this exponential increase in ELOCs become very prominent and shows significant margin of decreasing ELOCs for SQSDN implementation.

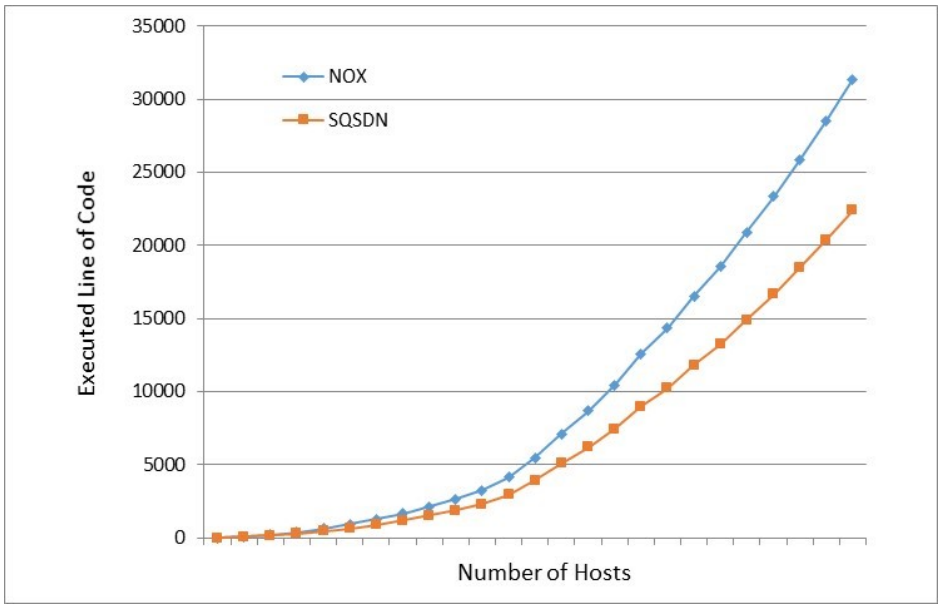

Figure 4.4: ELOC LSW Flow deployments by varying number of hosts.

4.3 Qualitative Analysis

As we built an abstraction layer of SQL over POX operating system. It caused steeper learning curve and reduced the coding complexity. Most of the programmers have knowledge of SQL as they learn it in databases course. As the SQL is not limited to network applications, learning SQL will also help programmers in developing other application.

Chapter 5

Conclusion and Future Work

5.1 Research Contributions

Our research shows that SQSDN framework that enables the use of SQL, for writing network applications it gives significant advantage in terms of reduction in lines of code and help in developing other better northbound APIs. It can be used directly for network function virtualization [50]. It provides ease for programmers to write run-time [39] environments for higher level abstractions.

SQSDN controller also enabled programmers to use database features in networks domain. As this controller parses the SQL query and identify errors in writing code. It pinpoints the code errors to help programmer to correct the coding errors. With this, programmers do not need to worry about data types and include different types of files to get OpenFlow structures for filling and sending messages to OpenFlow switches. The clean interface of SQL facilitates programmers to write a single instruction to define matches in the flow and action that is performed if the packets coming on a switch match with the flow.

It gives the way to emerge a very mature and standardized language from database domain to computer networks domain. It has provided a verified code base that is evolving over years from database domain to SDN. If we see on internet many open source projects are available from where database implementations can be adopted that have varying features. Some DBMS's support distributed database, some are build for handling very large data volumes and some do very secure transaction. For proof of concept we used a DBMS for our implementation that is platform independent and create a file based database on local disk. It has all the functionalities that a standard database has. For having any special functionality like triggers, database

distributed over a grid of computers, any other DBMS can be used.

We have enabled some functionality of databases that can be used in SDN and all the code base of DBMS is available with our implementation. Very little modification in verified code base can enable other functionalities like ACID (Atomicity, Consistency, Isolation, Durability) properties that are used for transaction handling DBMS's.

SQDSN provides simplicity and ease in performing CRUD (create, read, update, delete) operations on network switch flowtables. SQL also gives a new programming paradigm in which it gives significant ease in writing proactive network applications.

5.2 Problems

In developing our SQSDN framework we faced problems like DBMS implementation was in C language and we are building our controller in python language in which POX is built. As SDN is a relatively new, there is less support available from community to resolve any issues. The tools that we have used are evolving and often get into wrong configuration if we install non compatible versions.

5.3 Conclusion

In this study we investigated the suitability of SQL for efficient deployment of network management policy, using database abstraction. The results gathered from examples that were using SQSDN controller as well as the survey from programmers show that the SQL is well suited for efficient deployment of network management policy. The SQSDN controller has reduced the lines of code for developing network application for about 10% to 25%. As a result it has also reduced significant amount of executed line of code, because a flow deployment instruction is repeatedly executed whenever a flow deployment is requested or a change in flowtable is required.

Now SQL parsing, syntax and semantics checking facility of DBMS is being used in SQSDN controller. The DBMS is generating friendly messages in case of errors in SQL flow deployment instruction.

The use of SQL's standers and clean interface brought easiness and caused a steeper learning curve for programmers. It is very helpful for relatively new programmers, as flow deployment with SQSDN is providing user friendly errors to help them quickly pinpoint and fix these errors. It also helped them in a way that they do not need to worry about data types of IP's,

MAC addresses and ports. They just have to write a string query and pass it to the DBMS and flow would be deployed on the switch.

5.4 Future Work

Our framework make it possible to leverage ACID (Atomicity, Consistency, Isolation, Durability) properties from DB domain that guarantee DB transactions with reliably and robustness. For future work the ACID properties of DBMSs would be enabled for network switches. It will introduce a new paradigm where a DBMS will control a network transition from one state to another. It will ensure network transactions' reliability and durability. As distributed database DBMSs are also being used in databases field, these databases are used for managing data from database grids or clusters of servers. The example of these DBMS is Oracle database grid [51]and Oracle Clusterware [52]. In future the DBMSs as discussed above would be explored for building distributed controller, where a controller can run on multiple, physically separate locations and will manage its data (switches flowtables) transparently from user.

Appendix A

Data used in constructing graphs

In table A-1 and A-2 the executed lines of code are displayed in deployment of flow rules using firewall. The value of blocking rules is incremented from 1 to 25 for taking statistics of ELOCs.

Table A.1: Executed Line of Code for Flow deployments using firewall

POX ELOC's	28	56	84	112	140	168	196	224	252	280	308	336	364
SQSDN ELOC's	8	16	24	32	40	48	56	64	72	80	88	96	104
# of Blocking Rules	1	2	3	4	5	6	7	8	9	10	11	12	13

Table A.2: Executed Line of Code for Flow deployments using firewall

POX ELOC's	392	420	448	476	504	532	560	588	616	644	672	700
SQSDN ELOC's	112	120	128	136	144	152	160	168	176	184	192	200
# of Blocking Rules	14	15	16	17	18	19	20	21	22	23	24	25

The tables A-4, A-5, and A-6 show the values of flow deployed and the executed number of lines of code in an all pairs connectivity check scenario.

Table A.3: Executed Line of Code for Flow deployments using LSW

POX ELOC's	14	98	210	392	630	924	1274	1680	2142	2660
SQSDN ELOC's	10	70	150	280	450	660	910	1200	1530	1900
# of Hosts	2	4	6	8	10	12	14	16	18	20

Table A.4: Executed Line of Code for Flow deployments using LSW

POX ELOC's	3262	4144	5488	7126	8708	10458	12586	14336
SQSDN ELOC's	2330	2960	3920	5090	6220	7470	8990	10240
# of Hosts	22	24	26	28	30	32	34	36

Table A.5: Executed Line of Code for Flow deployments using LSW

POX ELOC's	16548	18536	20888	23338	25886	28532	31346
SQSDN ELOC's	11820	13240	14920	16670	18490	20380	22390
# of Hosts	38	40	42	44	46	48	50

Appendix B

Survey Form

Survey for testing SQL suitability for deployment of network management policy

1. Have you written any application using POX controller? ☐

2. Have you used SQL for querying relational data bases? ☐

3. Consider following scenario to build MAC based firewall;

- Openflow code:

```
msg = of.ofp_flow_mod()
msg.priority = 20
msg.actions.append(of.ofp_action_output(port=of.OFPP_NONE))
match = of.ofp_match()
match.dl_src = EthAddr('00:00:00:00:00:05')
match.dl_dst = EthAddr('00:00:00:00:00:01')
msg.match = match
event.connection.send(msg)
```

- Pyretic language:

```
not_allowed = none
not_allowed = not_allowed + match(srcmac=MAC('00:00:00:00:00:05'), dstmac=MAC('00:00:00:00:00:01')) +
match(srcmac=MAC('00:00:00:00:00:01'), dstmac=MAC('00:00:00:00:00:05'))
allowed = ~not_allowedreturn allowed >> act_like_switch()
```

- SQL:

```
query = "INSERT INTO OF_table (MAC_src, MAC_dst, Action) VALUES ('00:00:00:00:00:05' , '00:00:00:00:00:01',
'drop')"
sql_flow(event,query)
```

What method you would Preference for writing MAC based firewall SQL, Openflow or Pyretic?

_____

4. With SDN now the forwarding tables of switches can be seen as database tables and with our test framework SQL can be used for writing in these tables. If you want to write an SDN application then what method you prefer to use?

Openflow ☐ Pyretic ☐ SQL ☐

5. Give marks on scale of 100, how it is easy learning;

SQL for Switch table _____, Openflow structures, _____, Pyretic _____

8. What you think % of all programmers have knowledge;

SQL_____, Openflow structures, _____, Pyretic, _____

Name& Signature_____ Highest degree title: _____

Specialization:_____ Institute: : _____Cell # :_____

Figure B.1:

Bibliography

[1] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[2] Joshua Reich, Christopher Monsanto, Nate Foster, Jennifer Rexford, and David Walker. Modular sdn programming with pyretic. *Technical Reprot of USENIX*, 2013.

[3] Nate Foster, Rob Harrison, Michael J Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: A network programming language. *ACM SIGPLAN Notices*, 46(9):279–291, 2011.

[4] Douglas Crockford. *The application/json media type for javascript object notation (json).* Internet Engineering Task Force (IETF), 2006.

[5] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[6] Ryan Wallner and Robert Cannistra. An sdn approach: quality of service using big switchs floodlight open-source controller. *Proceedings of the Asia-Pacific Advanced Network*, 35:14–19, 2013.

[7] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014.

[8] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. *Compilers, Principles, Techniques.* Addison wesley, 1986.

[9] Li Li and Wu Chou. Design and describe rest api without violating rest: A petri net based approach. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 508–515. IEEE, 2011.

[10] Nate Foster, Arjun Guha, Mark Reitblatt, Alec Story, Michael J Freedman, Naga Praveen Katta, Christopher Monsanto, Joshua Reich, Jennifer Rexford, Cole Schlesinger, et al. Languages for software-defined networks. *IEEE Communications Magazine*, 51(2):128–134, 2013.

[11] Tim Nelson, Andrew D Ferguson, Michael JG Scheer, and Shriram Krishnamurthi. Tierless programming and reasoning for software-defined networks. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 519–531, 2014.

[12] Michael Hicks, Pankaj Kakkar, Jonathan T Moore, Carl A Gunter, and Scott Nettles. Plan: A programming language for active networks. *Submitted, November*, 1997.

[13] 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. *ACM SIGCOMM Computer Communication Review*, 35(5):41–54, 2005.

[14] Minlan Yu, Jennifer Rexford, Michael J Freedman, and Jia Wang. Scalable flow-based networking with difane. *ACM SIGCOMM Computer Communication Review*, 40(4):351–362, 2010.

[15] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.

[16] 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*, pages 3–3, 2010.

[17] Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: scaling flow management for high-performance networks. *ACM SIGCOMM Computer Communication Review*, 41(4):254–265, 2011.

[18] 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*, pages 37–42. ACM, 2012.

[19] Jeffrey C Mogul, Alvin AuYoung, Sujata Banerjee, Lucian Popa, Jeongkeun Lee, Jayaram Mudigonda, Puneet Sharma, and Yoshio Turner. Corybantic: towards the modular composition of sdn control programs. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, page 1. ACM, 2013.

[20] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6. ACM, 2014.

[21] Rahamatullah Khondoker, Adel Zaalouk, Ronald Marx, and Kpatcha Bayarou. Feature-based comparison and selection of software defined networking (sdn) controllers. In *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*, pages 1–7. IEEE, 2014.

[22] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.

[23] ONF Market Education Committee et al. Software-defined networking: The new norm for networks. *ONF White Paper*, 2012.

[24] RajaRevanth Narisetty, Levent Dane, Anatoliy Malishevskiy, Deniz Gurkan, Stuart Bailey, Sandhya Narayan, and Shivaram Mysore. Openflow configuration protocol: implementation for the of management plane. In *Research and Educational Experiment Workshop (GREE), 2013 Second GENI*, pages 66–67. IEEE, 2013.

[25] Markus Vahlenkamp, Fabian Schneider, Dirk Kutscher, and Jan Seedorf. Enabling information centric networking in ip networks using sdn. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–6. IEEE, 2013.

[26] 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, pages 67–72. ACM, 2012.

[27] Ronald Fagin, Jeffrey D Ullman, and Moshe Y Vardi. On the semantics of updates in databases. In *Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 352–365. ACM, 1983.

[28] Umeshwar Dayal, Eric Hanson, and Jennifer Widom. Active database systems. 1994.

[29] Wen Sun, Achille Fokoue, Kavitha Srinivas, Anastasios Kementsietsidis, Gang Hu, and Guotong Xie. Sqlgraph: an efficient relational-based property graph store. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1887–1901. ACM, 2015.

[30] Artem Chebotko, Shiyong Lu, and Farshad Fotouhi. Semantics preserving sparql-to-sql translation. *Data & Knowledge Engineering*, 68(10):973–1000, 2009.

[31] Stephen Harris and Nigel Shadbolt. Sparql query processing with conventional relational database systems. In *International Conference on Web Information Systems Engineering*, pages 235–244. Springer, 2005.

[32] Mehdi Bezahaf, Abdul Alim, and Laurent Mathy. Flowos: a flow-based platform for middleboxes. In *Proceedings of the 2013 workshop on Hot topics in middleboxes and network function virtualization*, pages 19–24. ACM, 2013.

[33] Open Networking Foundation (ONF). *OpenFlow Switch Specification, http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf*, 2011.

[34] *Beacon: A java-based OpenFlow control platform. , http://www.beaconcontroller.net.*

[35] Timothy L Hinrichs, Natasha S Gude, Martin Casado, John C Mitchell, and Scott Shenker. Practical declarative network management. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 1–10. ACM, 2009.

[36] 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*, page 7. ACM, 2011.

[37] 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*, pages 323–334. ACM, 2012.

[38] Zheng Cai et al. The preliminary design and implementation of the maestro network control platform. 2008.

[39] Christopher Monsanto, Nate Foster, Rob Harrison, and David Walker. A compiler and run-time system for network programming languages. In *ACM SIGPLAN Notices*, volume 47, pages 217–230. ACM, 2012.

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

[41] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing software defined networks. *NSDI, Apr*, 2013.

[42] Yuki Kawai, Yasuhiro Sato, Shingo Ata, Dijiang Huang, Deep Medhi, and Ikuo Oka. A database oriented management for asynchronous and consistent reconfiguration in software-defined networks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–5. IEEE, 2014.

[43] Peng Sun, Ratul Mahajan, Jennifer Rexford, Lihua Yuan, Ming Zhang, and Ahsan Arefin. A network-state management service. *ACM SIGCOMM Computer Communication Review*, 44(4):563–574, 2015.

[44] Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhammad Shahbaz, Nick Feamster, and Russ Clark. Kinetic: Verifiable dynamic network control. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 59–72, 2015.

[45] Michael Otey and Paul Conte. *SQL Server 2000 Developer's Guide*. McGraw-Hill Professional, 2000.

[46] *Wireshark protocol Analyzer (was Ethereal), http://www.wireshark.org.*

[47] *iperf, TCP and UDP bandwidth performance measurement tool, http://code.google.com/p/iperf.*

[48] Rogério Leão Santos de Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. Using mininet for emulation and prototyping software-defined networks. In *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, pages 1–6. IEEE, 2014.

[49] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of open vswitch. In *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, pages 117–130, 2015.

[50] ETSI Industry Spec. Group,network function virtualisation,.

[51] Meikel Poess and Raghunath Othayoth Nambiar. Large scale data warehouses on grid: Oracle database 10 g and hp proliant servers. In *Proceedings of the 31st international conference on Very large data bases*, pages 1055–1066. VLDB Endowment, 2005.

[52] Kandhasamy Gopalakrishnan. *Oracle Database 10g Real Application Clusters Handbook*. McGraw-Hill, 2007.