

# Actualizing Fast Lanes using Software Defined Networking and Smart Contract



By  
**Muhammad Muneem Shabir**  
00000172214

Supervisor  
**Dr. Syed Taha Ali**  
**Department of Electrical Engineering**

A thesis submitted in partial fulfillment of the requirements for the degree  
of Masters of Science in Electrical Engineering (MS EE)

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

(September 2019)

# Approval

It is certified that the contents and form of the thesis entitled “**Actualizing Fast Lanes using Software Defined Networking and SmartContract**” submitted by **Muhammad Muneem Shabir** have been found satisfactory for the requirement of the degree.

Advisor: **Dr.Syed Taha Ali**

Signature: \_\_\_\_\_

Date: \_\_\_\_\_

Committee Member 1: **Dr. Wajahat Hussain**

Signature: \_\_\_\_\_

Date: \_\_\_\_\_

Committee Member 2: **Mr. Muhammad Imran Abeel**

Signature: \_\_\_\_\_

Date: \_\_\_\_\_

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

Signature: \_\_\_\_\_

Date: \_\_\_\_\_

# Abstract

Fast Lanes in service provider's network are provisioned on each node throughout the route. Number of resources are required to construct a Fast Lane over a large network and a Fast Lane keeps on retaining the network resources (bandwidth, Data rate etc.) permanently. Client pays for services to develop a Fast Lane as well as retain the networking resources, which together add a significant amount to clients expenditures. Using third party financial platforms to pay ISPs, is a hindrance in fast service delivery. It lacks transparency, adds an overhead to the process and it is prone to cyber attacks. We propose a solution which uses SDN to centrally control the network and instantly construct an end-to-end Fast Lane using Ethereum based SmartContract. First, we have developed SDN controller App which lay a Fast Lane over the network for end-to-end communication, then we have developed a SmartContract on Ethereum to eliminate presence of 3rd party financial services. Using Ethereum we have built single pane-of-glass facility for consumer to acquire pay-as-you-go Fast Lanes access. In the end Intel SGX has been used for reliable communication of control traffic between Ethereum and SDN module. Research in this project presents the realization of open source platforms in SDN and Ethereum to provide on-demand service of Fast Lanes with SmartContracts.

# 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: **Muhammad Muneem Shabir**

Signature: \_\_\_\_\_

# Dedication

I dedicate this study to my parents, their wholeheartedly support and encouragement kept me motivated throughout the degree.

# Acknowledgment

All praise and thanks to Almighty Allah who blessed me with opportunities, strength and resourcefulness to pursue and complete the studies.

I would like to pay gratitude to my beloved parents. Without their constant support and guidance I would never be able to achieve what I have now. Their sincere prayers, wishes and love have made it possible for me to come this far.

My special thanks goes to Dr.Syed Taha Ali for being an amazing teacher and then for his guidance and incredible support throughout the journey of this research.

Here's to the friends. To the ones I met at NUST, your friendship empowered me and helped me get going through thick and thin. Thank you for your tremendous support and love.

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Overview . . . . .	2
1.2	Motivation . . . . .	3
1.3	Problem statement . . . . .	4
1.4	Proposed approach . . . . .	4
1.5	Contributions . . . . .	6
<b>2</b>	<b>State of the Art</b>	<b>7</b>
2.1	Conventional data communication networking . . . . .	7
2.2	Traditional electronic payment mechanism . . . . .	10
2.2.1	Credit payment system . . . . .	10
2.2.2	Debit payment system . . . . .	10
2.2.3	Pros and cons . . . . .	11
2.3	Software defined networks(SDN) . . . . .	11
2.3.1	History of SDN . . . . .	12
2.3.2	SDN Architecture . . . . .	13
2.3.3	SDN benefits and use-cases . . . . .	14
2.4	Software defined networking- Control plane . . . . .	15

<i>TABLE OF CONTENTS</i>	vii
2.4.1 Overview . . . . .	16
2.4.2 Ryu Controller . . . . .	16
2.5 Software defined networking -Data plane . . . . .	17
2.5.1 OpenFlow overview . . . . .	18
2.5.2 OpenFlow Architecture . . . . .	18
2.6 SmartContracts . . . . .	20
2.6.1 Brief History of BlockChain . . . . .	20
2.6.2 Ethereum . . . . .	21
2.7 Software Guard Extensions . . . . .	24
2.7.1 On-the-fly decryption . . . . .	25
2.7.2 Thread Model approach . . . . .	25
2.7.3 SGX support . . . . .	25
2.7.4 SGX operation . . . . .	25
<b>3 Proposed Architecture</b>	<b>27</b>
3.1 Overview . . . . .	27
3.2 Tools and Components . . . . .	29
3.2.1 Ryu SDN Controller . . . . .	30
3.2.2 RESTful API . . . . .	30
3.2.3 Topology API . . . . .	30
3.2.4 Web3 Library . . . . .	30
3.2.5 OpenSGX . . . . .	31
3.2.6 Remix Ethereum IDE . . . . .	31
<b>4 Design and Implementations</b>	<b>32</b>



4.1	Fast Lanes actualization by SmartContracts application specifications . . . . .	32
4.1.1	Application components . . . . .	32
4.1.2	Application workflow . . . . .	33
4.2	Ryu SDN Controller Fast Lanes application . . . . .	34
4.2.1	Network Discovery . . . . .	35
4.2.2	Network Monitoring . . . . .	35
4.2.3	Delay calculation . . . . .	35
4.2.4	Main module . . . . .	36
4.3	SmartContract . . . . .	36
4.3.1	Ethereum Virtual Machine . . . . .	36
4.3.2	Remix IDE . . . . .	36
4.3.3	Web3 Library . . . . .	37
4.3.4	Ropsten blockchain . . . . .	37
4.4	Software guard extensions . . . . .	37
4.5	API server . . . . .	38
<b>5</b>	<b>Results and Observations</b>	<b>39</b>
5.1	SDN network topology . . . . .	39
5.2	SmartContract structure . . . . .	41
5.3	Software Guard extensions . . . . .	42
5.4	Testing scenario . . . . .	42
5.5	results . . . . .	43
<b>6</b>	<b>Conclusion</b>	<b>48</b>

<i>TABLE OF CONTENTS</i>	ix
<b>7 Future work</b>	<b>51</b>
7.1 OpenSGX compatibility . . . . .	52
7.2 Topology discovery . . . . .	52
7.3 Smart contract . . . . .	52

# List of Figures

1.1	SDN based Network for Fast Lanes with Ethereum based Smart-Contract . . . . .	5
2.1	Conventional IP network . . . . .	8
2.2	Traditional networking building blocks . . . . .	9
2.3	SDN architecture . . . . .	14
2.4	Ryu controller architecture . . . . .	17
2.5	OpenFlow connection between switch and controller architecture	18
2.6	SGX architecture . . . . .	24
2.7	SGX operation flow . . . . .	26
3.1	Proposed architecture of SDN Fast Lanes with Ethereum based SmartContract . . . . .	28
4.1	Component description of SDN Fast Lanes actualization by SmartContract . . . . .	33
4.2	Component description of SDN controller application for Fast Lanes . . . . .	34
5.1	Flow table entries in OpenFlow switch . . . . .	40

5.2 Network topology discovered by Ryu controller . . . . . 40

5.3 SmartContract implementation on Remix IDE . . . . . 41

5.4 SmartContract implementation on Remix IDE . . . . . 41

5.5 Flow table entries in OpenFlow switch . . . . . 43

5.6 SDN Fast Lanes controller application and sub-modules initialization . . . . . 43

5.7 switches and Ryu controller connection setup . . . . . 44

5.8 SmartContract input functions on user-interface . . . . . 44

5.9 SmartContract successful transaction dialog . . . . . 45

5.10 Generating key-pair for SGX Enclave . . . . . 45

5.11 Encrypting data using the key-pair and storing it in Enclave . 45

5.12 SGX terminal output for Enclave data (Fast Lanes provisioning request) is accessed . . . . . 46

5.13 Provisioned Fast Lanes statistics on Ryu controller . . . . . 47

# Chapter 1

## Introduction

This research work contains the design and implementation of system which allows the user to acquire a Fast Lane over an IP network using Software Defined Networking and Pay for it by Ethereum based block-chain network. With the help of SmartContract, financial arbitration is settled beforehand and user can leverage pay-as-you-go method. Several techniques have been introduced to improve quality of service on Fast Lanes when acquiring Fast Lanes and eliminated the role of middleman from provisioning the network service and payment mechanism. Overview of thesis is described next, followed by motivation and problem statement. Approach to address the problem statement have been described with low-down explanation of each module and component below.

## 1.1 Overview

With increasing footprint of smart devices and emergence of Internet-of-things, usage of internet has been increased exponentially. Researchers are constantly evolving devices and units to make them smart and connected to internet to have a remote access and control. Smart devices have given boost to internet usage, but the conventional way of handling internet is major hindrance of evolving IT solutions to a smart system. Software defined networking over the past few years is a subject area of exploration to address this issue. Handful of developments have been made so far and rest are yet to come. Personalized internet control is leveraged in this thesis using an application written in Python language off RESTful API of Ryu controller. Controller Application in this project is a program which allows user to sign-up for a specific amount of time, lay a Fast Lane for a flow of his choosing and pay for it through fully transparent distributed block chain mechanism known as Ethereum. A SmartContract is written on Ethereum block-chain. It is an electronic agreement between end-user and network service provider, and does not require any middlemans involvement. It protects both parties interests and execute the agreement with 100 percent transparency and integrity. SmartContract is written in Solidity language and hosted on Ethereum virtual machine and that is why it entertains the benefits of immutability and no party can coarse or breach the agreement. SmartContract in our project provides users with interface to select a flow of their choosing and pay for it. SmartContract on the bases of pre-defined agreement carry out the financial transactions and instruct SDN to spin-off the Fast Lane and let end user

enjoy the seamless, top quality of service end-to-end flow. SGX is positioned between SDN and EVM for both modules to exchange control traffic through it. SGX provides the role of trusted platform module between Ethereum and SDN to curb any manipulations in control traffic from adversary.

## 1.2 Motivation

Emergence of digitization has made internet the backbone of the modern world and ISP role is no more confined just to provide the network access. ISPs are required to provide personalized services to the clients which include but not limited to QoS, tunnels, L3 VPNs, L2 VPNs, content delivery, streaming, storage, gaming services and so on. Some on-premise services i.e. video conferencing, gaming, streaming and storage area network traffic between clients' different sites required smooth and better end-to-end network paths which need to be provisioned right away in case of mission critical services. With the dawn of SDN, network personalization has started taking place. It provides service provider the ability to control and manage its network centrally and write any functions required to perform, using SDN controller API. SDN controller can interact to 3rd party applications and programs using programmable interface and personalized services can be achieved with better QoS and response time by developing programs on API. The monetizing platforms should no longer be owned by 3rd parties because that gives them unnecessary network statistics which in the era of big data is a significant thing to give way. Ethereum blockchain addresses this solution by offering the ability to execute contracts on a distributed computing

platform.

### 1.3 Problem statement

Client requests service providers to provide them with a Fast Lane for better end-to-end connectivity. Service providers on-demand provision a Fast Lane which is static in nature. Team of technical resources lay this end-to-end Fast Lane by configuring each node on hops and it stays provisioned whether any traffic is transferred or not, and customer pays for acquiring this constant facility. This Fast Lane is monetized using a billing and charging system at service provider. Charging system is often a system provided by a bank who controls the finances on ISPs behalf. A whole solution is deployed to get notified for the transaction from banking portal and use that to decide whether to provide client with service or not. Involvement of a bank put clients and ISPs information at risk and any contenders in market can leverage off that. Banking systems when working online are prone to cyber attacks and any manipulation can cause catastrophic events for a client who is paying for a mission critical application. Moreover, charging for a service through a 3rd party involves an overhead in the daily business.

### 1.4 Proposed approach

To spin off a self-provisioned Fast Lanes through SmartContracts deployed on distributed ledger network, we have built an application on top of SDN controller by using its API. SDN due to its flexible nature and centralized



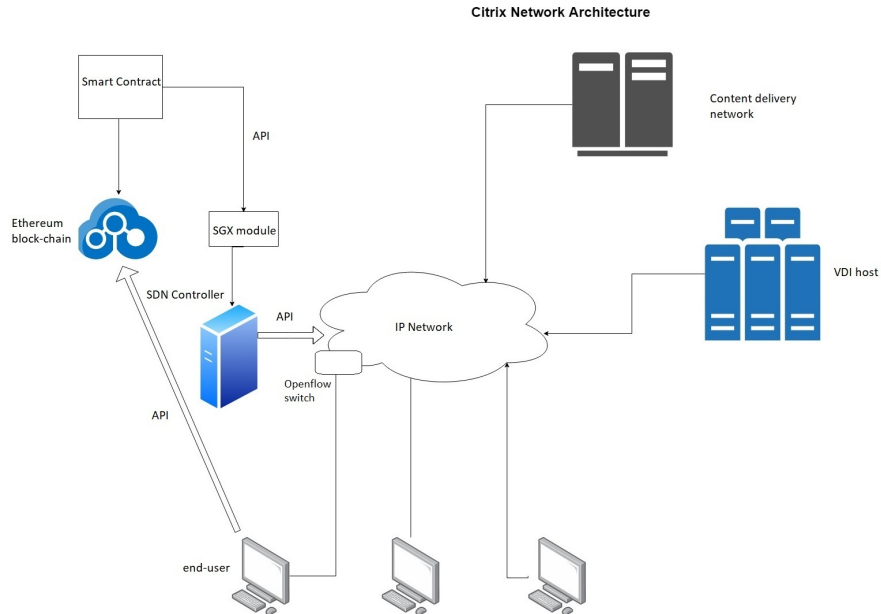


Figure 1.1: SDN based Network for Fast Lanes with Ethereum based Smart-Contract

control allows the developer to build applications to control and manage networks centrally. We integrated SDN module with Ethereum virtual machine and developed a SmartContract on it. SmartContract while on one hand allows the user to select a Fast Lane and proceed for payment. It on the other hand passes on the instructions to SDN controller for provisioning the Fast Lanes. To protect the control traffic from manipulation, an SGX module has been introduced with guarantees for protection of control traffic between modules in discussion. This presented architecture is further elaborated in chapter 3. High level design diagram of proposed architecture is illustrated in figure 1.1.

## 1.5 Contributions

Our primary contribution in this project comprises developing a prototype system which provision on-demand Fast Lanes on SDN network using Smart-Contract as pay-as-you-go-method. The whole project mainly consist of four functional areas.

- IP network topology discovery.
- Bandwidth and Delay based optimized path selection for end-to-end link over an SDN based IP network.
- SmartContract in Solidity language deployed on Ethereum virtual machine to execute financial transactions, carry out the contract between parties to provision Fast Lanes and provide user-interface to invoke and revoke a service.
- Software guard extension as a trusted platform module for reliable control traffic exchange between different modules.

# Chapter 2

## State of the Art

This section of thesis elaborates all components of Ethereum virtual machine and its function to execute SmartContracts. We intend to present the application of Ethereum virtual machine as a financial system and contract arbitrator between concerned parties alongside the network. We also explain the building blocks of software defined networking.

### **2.1 Conventional data communication networking**

Traditional data communication is carried out using the network of interconnected nodes which are connected in different type of typologies. These interconnected nodes are categorized in a hierarchy and operations. These devices use the framework of OSI model and typically work on layer 2 or layer 3 [1]. Devices designed on layer 2 are called switches. Switch is responsible to make decision based on incoming MAC address and forward it to the right

port based on destination MAC address in the frame.

Router on the other hand handles the packet on the bases of source and destination IP addresses and forward the packet to respective sub network. Hundreds of switches forming a LAN network can aggregate on a single port of router for a gateway. In conventional packet-switch networking, routers and switches are mainly dedicated hardware devices. Their role recently has been expanded from dedicated hardware appliances to virtualized software modules. This allows a single server in datacenter to house tens of virtual routers and switches on top of shared computing resources [2]. Devices in traditional networking are always proprietary and their fabrics are built on top of FPGA and ASICs [3].

Conventional networking has 3 building blocks mentioned below

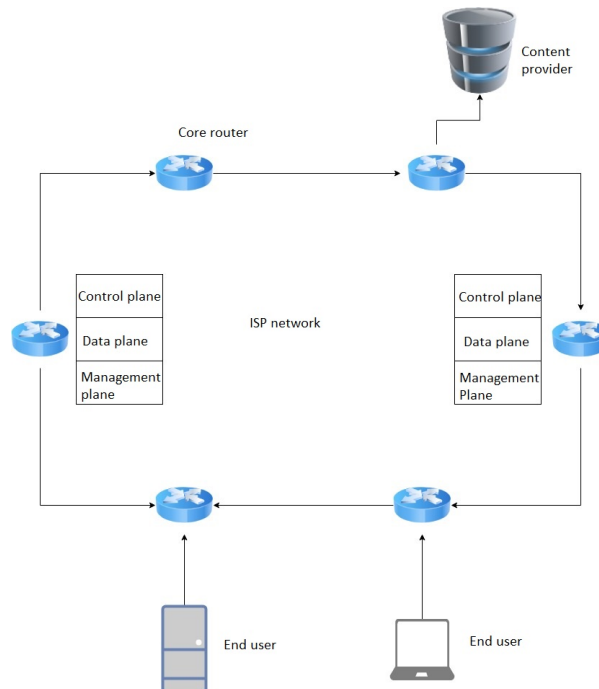


Figure 2.1: Conventional IP network

- **Data plane:** In routers and switches data plane helps to forward traffic to the next hop on the bases of forwarding table built by control plane. Data plane is mostly a CPU based process.
- **Management plane:** Management plane is built to provide interaction with control plane for Command-line, network monitoring and management tools.
- **Control plane:** It uses system information and routing protocols i.e. RIP, OSPF to fill out the forwarding tables. Control plane is hard coded on the fabric.

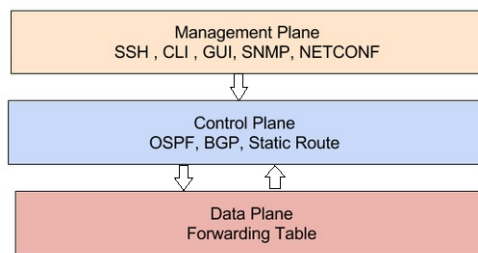


Figure 2.2: Traditional networking building blocks

These planes are implemented on a same appliance whether it is a switch, router or a firewall. It limits the scalability, flexibility and becomes the hindrance in the realization of advance technology paradigms i.e. IoT . In coming topics, we are going to discuss software defined networking that curb these limitations and opens new research opportunities in networking.

## **2.2 Traditional electronic payment mechanism**

Electronic payment method is a mean to pay for a service acquired without paying in cash or cheque. Payment is provided online using credits or account transactions for the services which are either acquired online or in tangible form. One of the most common online payment method is credit card or debit card provided by some bank. Costumer in possession of these cards can pay for a service he wants to acquire through internet. Service providers faces user-interface provided by the bank to the costumer for payment. This portal is provided by a 3rd party bank which controls and manages costumers credit or debit history and indicates the service provider to provide a service once the amount is paid.

### **2.2.1 Credit payment system**

In this system costumer in possession of credits which are equivalent to real time currency can acquire this service through sending those credits off to the bank and bank on behalf of costumer sends the real currency to the service providers account. Bank let service provider knows the transaction and service provider gives the service to the concerned costumer.

### **2.2.2 Debit payment system**

This system requires the costumer to debit the payment in service providers account by using the middle-ware banks portal. Service provider cross verifies the payment with bank and allows the costumer to acquire service for paid amount of time.

### 2.2.3 Pros and cons

Electronic payment system is built to facilitate the customer to pay for online services without paying personal visit to the bank. Customer can transfer the funds anywhere and at any moment when he requires the service. Where it provides ease and comfort with payment, it has many cons as well. A customer and service provider must rely on third party system and trust it with their financial history. Sometimes that information can lead to disclosures which competitors can leverage on. 3rd party portals are sourced from banks and If their system is down it becomes hindrance between service providers and customer. While 3rd party payment facilities deprive you of your anonymity it is also prone to financial attacks. E-payment attacks are growing up by 30 percent, therefore, technical companies are focusing on finding attack-proof solutions which are sophisticated to integrate, and can rule-out the need of 3rd party's involvement. We will go in details on proposed technology in the topics to follow.

## 2.3 Software defined networks(SDN)

SDN provides abstraction to decouple the control plane and data plane from each other which in traditional way are implemented on a same device. Decoupling the control plane and data plane helps to control the SDN network through single-pane-of-glass by a centralized controller [4]. This centralized control allows the network operators and developers to run and implement policies centrally while actions take place on forwarding nodes that are spread throughout the networks [5]. SDN controller provides a programmable in-

terface called northbound API which allows the user to develop applications to manage data plane [6]. Southbound API helps controller to interact with forwarding nodes. Various protocols are available to run on southbound API. i.e. NETCONF, SNMP and OpenFlow [7]. These vendor agnostic protocols provide the seamless connectivity between multi-vendor network forwarding nodes and keeps the simplicity on forwarding plane. OpenFlow by far has been adopted widely in SDN networks and stand out the most [8].

Open networking foundation presented the concept of OpenFlow protocol whose main purpose was to provide a seamless connectivity between controller and forwarding nodes while having no regards to the vendor specification. Due to its simplicity and subtle packet formation OpenFlow has been adapted widely and currently considered as a standard protocol on Southbound API of SDN [8].

### 2.3.1 History of SDN

McKeown et al in 2008 presented the idea of software defined networking in a paper OpenFlow: Enabling Innovation in Campus Networks [9]. The idea of programmable data plane to make intelligent forwarding decisions is dated back to 1997 [10]. After that, concept to decouple the data and control plane has been emphasized time to time by various projects such as Tempest, PCE, NCP, ForCES, RCP with the focus for better network wide control in Ethernet, MPLS and ATM framework networks. Solutions i.e. Ethane [11], SANE [12], NOX [13] presented the sophisticated decoupling of data plane and forwarding plane with minimum effort on data plane nodes



gave rise to advancement in this area of research and became driving force for the orchestration of OpenFlow [9]. OpenFlow opens research area for new service running on top of network by moving the control of network centrally and introducing the concept of segmentation without effecting network performance.

### 2.3.2 SDN Architecture

Software defined networking is a paradigm in which network programmability is made possible by abstracting [14] different functionality layers. The abstraction layer at the bottom of hierarchy is called infrastructure or data plane layer. This layer comprises forwarding nodes which are communicated by controller for forwarding decisions via OpenFlow [9] protocol through Southbound API. Another layer of abstraction is the control plane layer. This layer is the heart of network operating system. All the decisions for forwarding planes are taken in this abstraction layer by applications running on the northbound interface of centralized controller. As compared to southbound interface, northbound interface provide flexibility and variety of functions on top to program and manipulate network functions. Controllers as such Open Daylight [15], Ryu [16] and NOX [13] support Java or Python programs to run on their northbound API. Figure 2.3 is describing the SDN hierarchical architecture.

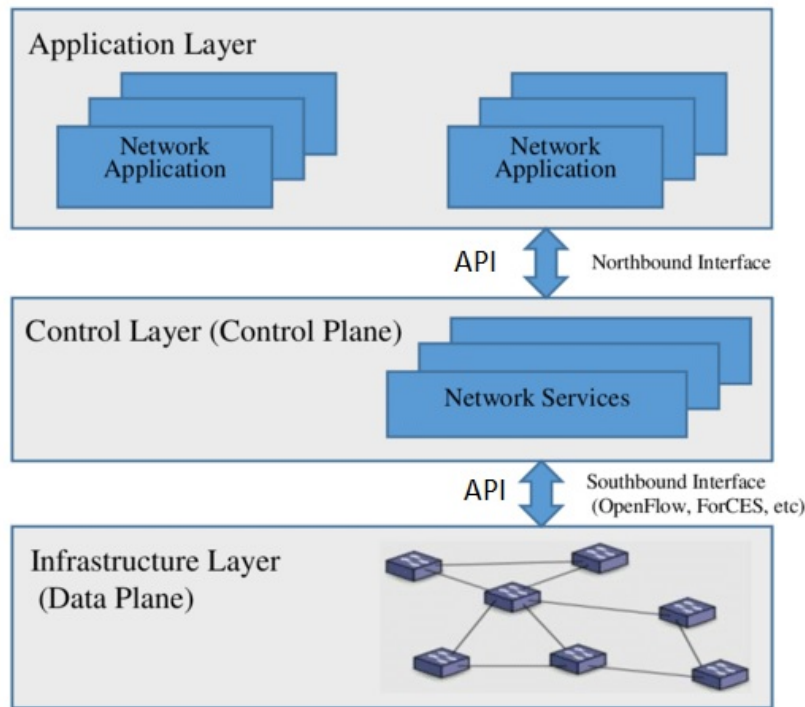


Figure 2.3: SDN architecture

### 2.3.3 SDN benefits and use-cases

There are plenty areas of networking where SDN can be implemented to improve network control, performance and scalability. The most prominent fields where SDN has been under the research are mentioned below.

- Data Center networks:** Granular control and features of SDN i.e. network virtualization and segmentation make SDN useful for Data center networking. The integration of NFV [16] and SDN provides better control over the rack and stack of network and brings down the CAPEX and OPEX to incredibly low level as compared to traditional vendor proprietary networking solutions.

- **network management system (NMS):** Due to SDNs centralized controller, it provides user better control over the network through single pane of glass. i.e. FlowVisor [17].
- **Traffic engineering:** SDN provides network slicing and segmentation which is the essence of overlay networks and hence make extremely easy to implement traffic engineering protocols on top of it. I.e. mBGP, MPLS-TP, MPLS-TE, VPLS, L3-VPN, L2-VPNs.
- **Mobility networks:** Light virtual access points [18] open research areas in wireless and mobility networks.
- **Security networks:** Piggybacking security functions [17] on SDN enables the cloud-based security networking. Any firewall or security application can be developed at the centralized controller and implemented networkwide.

## 2.4 Software defined networking- Control plane

Controller or network NOS (network operating system) [19] is a centrally located network director and manager which interacts with all the nodes spread in the network for flow decisions. Controller takes the decision and provides the data plane nodes with flow entries. Data plane fabrics take the forwarding decisions on the bases of flow entries provided. OpenFlow [9] is used for back and forth communication between controller and data plane nodes.

### 2.4.1 Overview

After the emergence of OpenFlow in 2008 several controllers have been designed for SDN networking. There are some trade-offs with respect to performances and flexibility among the controllers that have been built so far. But their I/O and APIs architecture is similar. All of them use REST API on their northbound interface for management and functionality and OpenFlow on their southbound interface to interact with data plane nodes. The most prominent and advancing open source controllers are Open Day Light [15], POX [20] and NOX [13]. Market leading vendors have also manufactured computing devices dedicated for SDN controllers and bundled up their license on them. Cisco APIC [21], VMware NSX [22] and HPE VAN are most prominent ones in enterprise market.

### 2.4.2 Ryu Controller

Our SDN network in this project is based on Ryu controller [20]. Ryu is a Python written controller, licensed under Apache2.0, and is available for anyone to use.

It has standard OpenVswitch support on its northbound end. Ryu is developed and supported by NTT labs Japan. Ryu has vast variety of libraries and modules available and is by far the most advancing SDN controller. Architecture of Ryu controller is illustrated in Figure 2.4.

Ryu project has have several built-in APIs to perform various network functions. Some common Ryu APIs are below.

- Firewall API: Using this API, developer can write functions to make

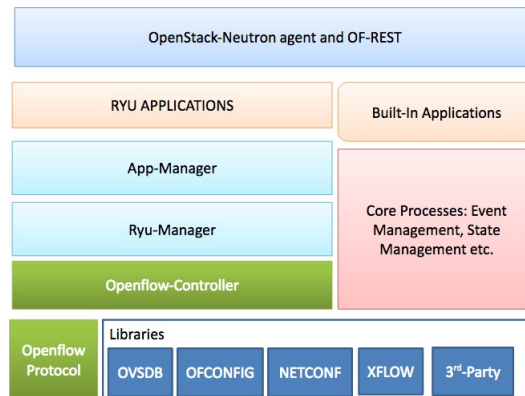


Figure 2.4: Ryu controller architecture

data plane switches perform firewall functions. The security functions can be as complex as Next-generation-firewalls.

- QoS API: This API is used to prioritize services and flows.
- Flow-Table API: Along with flow manipulations this API is useful to pull statistics for flows, queues and buffer values.
- Topology API: This API helps to draw network-wide topology in GUI using Apache2.0 web server. It orchestrates the topology using LLDP packets.

## 2.5 Software defined networking -Data plane

Data plane is a layer of forwarding devices spread across the network. These devices can be routers or switches with their primary job to switch packets from one to the other using the information provided in the flow table by Ryu Controller.

### 2.5.1 OpenFlow overview

It is used for communication between centralized controller and data plane nodes. OpenFlow started rolling out in 2008 [9]. It allows the flow entries in data plane nodes to be modified centrally in the real time.

Data plane switches which support the OpenFlow protocol are called OpenFlow switches. OpenFlow switches establish a SSL channel with SDN controller and use instructions sent by controller to update their flow tables.

### 2.5.2 OpenFlow Architecture

The primary motivation behind OpenFlow is to provide seamless communication in vendor-agnostic environments. It is most used protocol in SDN and considered a standard on southbound API as appose to NETCONF, SNMP, MPLS etc.

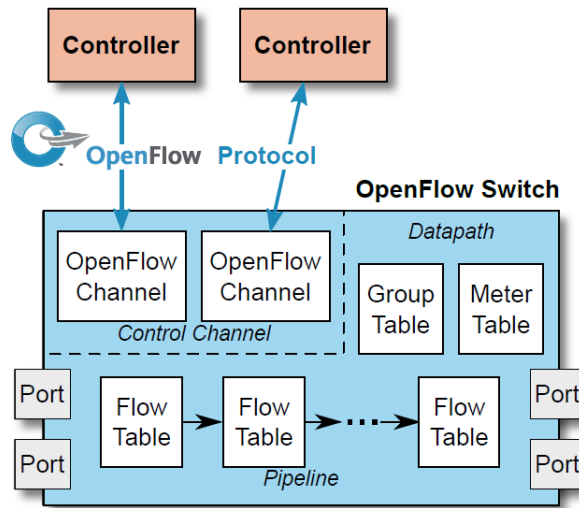


Figure 2.5: OpenFlow connection between switch and controller architecture

Switch which supports OpenFlow protocol is also called OpenFlow switch.

The main component of OpenFlow switch is the flow table entry database. This routing table lookalike database is used for all forwarding decisions. OpenFlow establishes a secure channel between the switch and controller before the real control packets start flowing back and forth. OpenFlow provides a separate table which is called group flow. This table is useful for decisions that can be applied to prefixes or users in group for Policy based routing.

Flow table entries have following blocks.

- **Header:** Header field contains all the addresses from layer 2 to layer 5. It contains source and destination IP addresses, Mac addresses, TCP/UDP port numbers etc.
- **Actions:** This field is a stack of actions which can be applied to the particular packets. i.e. drop the packet, forward the packet to some port, or send it to the controller as shown in table 2.1.

Action	Role
Push-Tag/Pop-Tag	push and pop MPLS and VLAN tags
Drop	Drop the packet if no action is specified
Group	Process the packet through the specified group.
Set-Queue	set Queue-ID for packet. It is used for QoS
Output	Forward a packet to specific output port

Table 2.1: Open flow action entries

- **Counter:** Statistics are collected using the counter. Counters are triggered when events are generated.
- **Match entries:** It is a predefined set of information that can be used for matching with incoming packets and process them for actions. Match can be based off IP addresses, Mac address, VLANs, Port

no. etc. If there is no match entry available for the incoming packet, the packet goes to the controller as *packetIn*. Controller evaluates the *PacketIn* and updates the switch with flow entry for future. Predefined match fields are available in OpenFlow specification documents [8].

## 2.6 SmartContracts

SmartContract is a self-executing contract between concerned parties with terms of agreement written into the lines of code [23]. It is not constrained to a hardware or server, rather it is stored and implemented on a distributed blockchain network. SmartContract is implemented on a blockchain [24] platform called Ethereum [25] which provides the ability to be executed and accessed across the internet. In later sections we will discuss SmartContracts and building blocks in details.

### 2.6.1 Brief History of Blockchain

Blockchain are considered brainchild of Satoshi Nakamoto -his identity is unknown to this day. Satoshi in 2008 presented the idea of ledger based financial system. The idea of distributed ledger relates to the formation of bit-torrent [26]. It is a peer to peer network where parties can share files without having any involvement of a centralized point or node. Satoshi presented an electronic currency called bitcoin which is based on cryptographic proof instead of trust [26] and hence proved quite promising in online financial settlements. By 2017 the Bitcoin Blockchains have grown by 100GB in size. In 2013 Vitalik Butern in his white paper *A Next-Generation Smart-*



*Contract and Decentralized Application Platform* shaped up the blockchains to a whole new concept. He presented the concept of blockchain based virtual machines with the intent to build decentralized applications. Vitalik Butern named it Ethereum [27]. From then on Ethereum has become the most sophisticated platform for decentralized applications and SmartContracts. A financial algorithm is also defined which is called Ether, Ether works significantly different, compared to the previously built coins. Ether is a value to pay for any execution on the ledger. Ether is intended to be taken as a fuel for computing power rather than a coin [27].

### **2.6.2 Ethereum**

Ethereum is a public Blockchain based distributed computing platform and OS (operating system). It is built to host SmartContracts. It is based on an improved version of Satoshi Nakamotos blockchains as financial platform. A token named Ether is used whose blockchain is implemented by Ethereum platform. Ethereum platform is also used as Ethereum virtual machine (EVM) which run and executes scripts using public nodes spread across the internet. Ethereum virtual machines' instructions set and scripting language is made to be universally computational as appose to the bitcoin platform.

#### **Ether**

Ether is a base token for Ethereum operations which provides the publicly available distributed ledger for computations and transactions. All computations in EVM are enforced by transactions which converts the computational

effort in gas and instruct the EVM to run certain script when the specific amount for the gas is paid [27]. Ether, the token with symbol ETH is used to pay for gas.

### **Addresses scheme**

Every Ethereum node, whether computing or client has a unique Ethereum address associated with it. Identifier for hexadecimal number scheme 0x is used in concatenation with 20 rightmost bytes taken off Keccak-256 hash of ECDSA public key. Each address contains 40Hexadecimal digits in total. i.e. 0xb793F5fA9ba35494bE833614fffBA74279579267. Contract and transactional accounts both follow the same address format and are indistinguishable. As appose to bitcoin which uses base58check to validate the address, any keccak-256 hash can do for Ethereum address if properly put in the format.

### **Ethereum virtual machine**

Using the publicly available distributed ledger on peer to peer network, Ethereum provides a virtual machine on top of it. By pooling all the resources available on Ethereum network off Ethereum nodes, Virtual machine encompasses the 256-bit register stack to run codes written in SmartContracts as intended [27]. The EVM is developed in various programming languages i.e. C, C++, C, Python, Ruby, Rust, etc.

### **Ethereum as a SmartContract**

SmartContract [27] is a protocol which digitally verifies, facilitates and enforces the negotiations and implementation of an agreement between two parties without a middleman. SmartContracts are written with a script in an interactive tool. Solidity is used to detail a SmartContract. SmartContract written in solidity is converted to byte code using solidity compilers. EVM byte code in compliance with the Ethereum virtual machine runs the code on the blockchain. Smart Contracts on Ethereum are categorized in two levels.

1. High level abstraction is provided for scripts which can be written in various programming languages i.e. Serpent, Low-level-lisp, Mutant, solidity etc. Of all above languages, solidity is being used in production the most due to its library and structure like C.
2. On the low level, script written in interactive tool is converted to EVM byte code.

SmartContracts can either be private (hashed) or public, which guarantees the transparency between two parties withholding an agreement and abiding by its instructions through computation on EVM [27]. This transparent attribute of SmartContracts give rise to the possibility of creating many distributed applications for gaming, auctions or financial deals with complete transparency and integrity [28].

## Ethereum on permissioned ledger

As appose to the public blockchain ledger a concept of permissioned ledger is introduced. Permissioned ledger adds an access control layer on top of blockchain to grant or deny the features to certain individuals or accounts [29]. JP. Morgan Chase has presented a permissioned ledger based Ethereum coin JPM coin in a bid to provide integrity between consensual parties without revealing their identities in financial on-goings [29]. A project by ConsenSys is rolled out by the name Pantheon which provides privacy using access control [30].

## 2.7 Software Guard Extensions

SGX (Software guard extensions) is a bundle of instruction codes hosted by modern processors. They allow to build a private memory region known as Enclaves. Content in Enclave [31] is cryptographically protected and can not be read or modified by any CPU even if it has a higher-level access privilege.

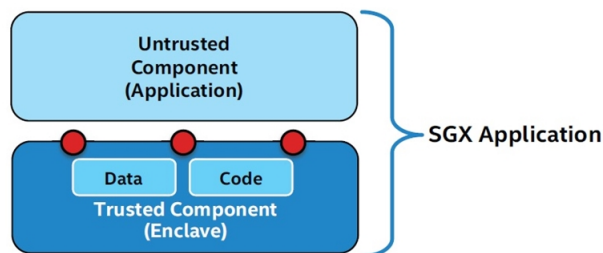


Figure 2.6: SGX architecture

### **2.7.1 On-the-fly decryption**

Data that is kept in the Enclave is cryptographically encrypted using the pair of RSA keys between memory block and CPU. The data is decrypted using the CPU's Enclave key and computed. It cannot be accessed by any other computing body from outside of Enclave. Process thus makes it sure that the code which should run in the Enclave can neither be spied on nor used by other code in any way.

### **2.7.2 Thread Model approach**

In this model all cyber variabilities can be identified, categorized and listed from a potential attacker's point of view. The data or the code running in Enclave uses thread model to become attack-proof. In this design Enclave is marked as trusted and all resources or entities outside of Enclave are marked as potentially hostile, including operating system, Kernel, Hypervisors, I/O devices or computing units etc.

### **2.7.3 SGX support**

SGX support in the CPU is identified by its CPUID Structured Extended feature Leaf<sup>7</sup>, with EBX bit 02. But SGX is only available to the applications if it is enabled from the BIOS.

### **2.7.4 SGX operation**

Every code leveraging the advantages of SGX is written in two sections, trusted and untrusted. When application runs, it instantly creates an Enclave

in memory space which becomes a trusted memory space. Trusted memory space (Enclave) provides a function (trust-function) which upon called from external space, computes the code in secure environment and returns the value for external use. Processing data can be seen by the Enclave, but it is not accessible from the outside. Trusted data remains in the Enclave memory unchanged when trust-function returns.

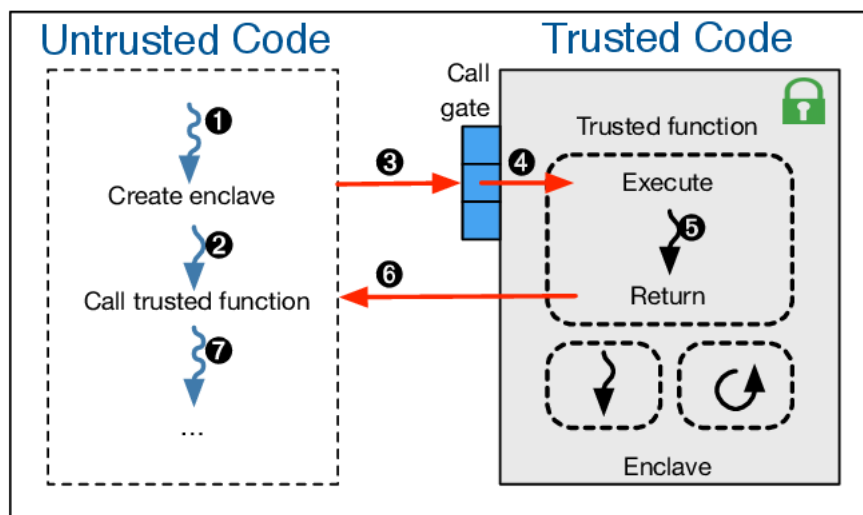


Figure 2.7: SGX operation flow

# Chapter 3

## Proposed Architecture

In this chapter proposed architecture and its working flow has been discussed in detail.

### 3.1 Overview

This project aims to automate service provider network commissioning. From high-level-design's perspective the project proposes the automated design approach in network which provision the end-to-end Fast Lanes for IP flows using the Software defined networking. Monetization for Fast Lanes as a service is carried out using Ethereum based SmartContract. ABI (Application binary interface) based GUI (graphical user interface) is provided to the costumer to acquire Fast Lane on pay-as-you-go bases. This advanced monetization of service provider networks curbs the role of financial management party as a middleman and increases the revenue. It also reduces the management and provisioning time drastically low. The SGX positioning between

SDN and Ethereum network provides trusted environment for control traffic.

Our project consists of four main components mentioned below

- Ryu SDN Controller Application
- Ethereum based SmartContract
- API server
- Software guard extensions

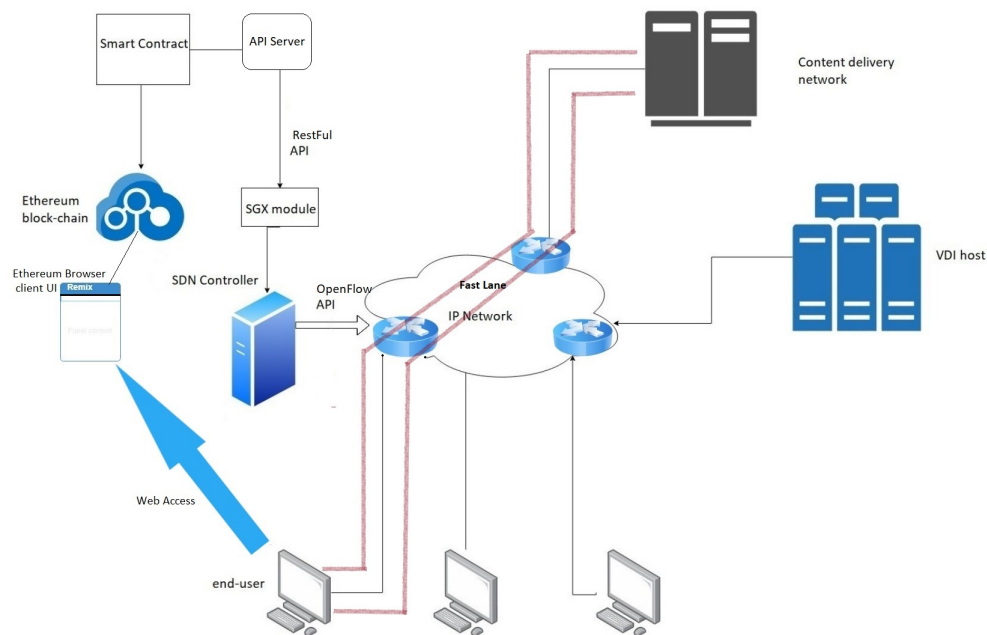


Figure 3.1: Proposed architecture of SDN Fast Lanes with Ethereum based SmartContract



First component, controller app is responsible for provisioning Fast Lanes on SDN nodes and statistics collection of entire networks. Second component is a payment platform which is built on Ethereum blockchain.

It is a robust financial payment which not only monetize the network service but also send and receive control messages with Ryu controller to take required action on the network. API server acknowledges the user request and payment fulfillment and instructs controller through APIs to provision the requested Fast Lane. At the last, SGX (software guard extension) plays the vital role in providing trusted environment for control traffic between Ethereum platform and API server. Architecture of proposed design is shown in Figure 3.1.

## 3.2 Tools and Components

- Ryu SDN Controller
- RESTful API
- Topology API
- Web3.py library
- OpenSGX
- Remix Ethereum IDE

### **3.2.1 Ryu SDN Controller**

Ryu SDN controller is a Python based SDN controller which uses the component-based framework with huge variety of APIs and libraries to make network agile and easy to manage. Ryu is supported and advanced by NTT. It has a wide-spread community of developers, active support and periodic upgrades with documentation and hence becomes the best choice off all available SDN controllers.

### **3.2.2 RESTful API**

The RESTful API often goes with the name REST API is built on the north-bound interface on Ryu controller to interact with upper layer application with the intent to retrieve network statistics, network information and functions manipulations.

### **3.2.3 Topology API**

This API communicates through northbound interface with application layer interfaces. It provides topology information formulated with the help of LLDP packets dispersion. This API allows the graphical illustration of networkwide topology.

### **3.2.4 Web3 Library**

This library is built in Python module to interact with Ethereum platform. It has the API quite like JavaScript API and it is used to translate likewise scripts in Python format.

### **3.2.5 OpenSGX**

OpenSGX [32] is an SGX emulator built on the binary translations of Qemu [33]. It provides similar functionality as Intel's proprietary hardware SGX. Sgplib library is used to perform Intel-like operations inside and outside the Enclaves. OS emulator [32] and Enclave program loader implements the SGX abstractions and stacks regions appropriately [32].

### **3.2.6 Remix Ethereum IDE**

Remix is a set of components and tools to interact with Ethereum whether to build a SmartContract or debug a dApp [27]. It can be used through its web access with UI or can be downloaded from GitHub for local setup.

# Chapter 4

## Design and Implementations

### 4.1 Fast Lanes actualization by SmartContracts application specifications

In this research project we have developed a Ryu controller SDN application to provision end-to-end Fast Lanes. We have also designed and deployed SmartContracts to work as a payment gateway and further initiate instructions securely to the Ryu application to actualize the Fast Lanes. Applications modules and components are elaborated in the sections below.

#### 4.1.1 Application components

the Fast Lane actualization application consists of four main modules listed below.

- Ryu SDN Controller Fast Lanes Application
- Ethereum based SmartContract

- Software guard extensions
- API server

These modules are illustrated in the figure 4.1 below.

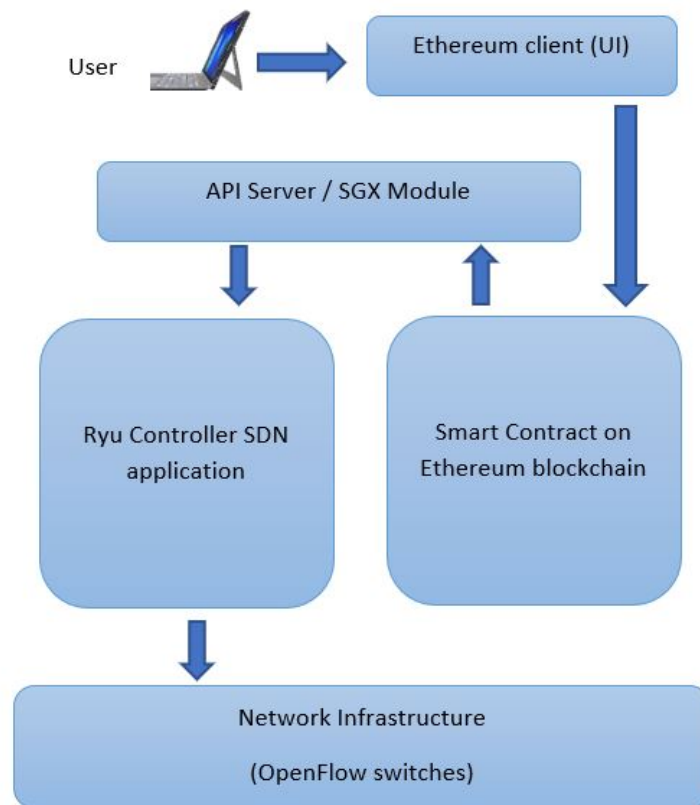


Figure 4.1: Component description of SDN Fast Lanes actualization by SmartContract

### 4.1.2 Application workflow

Customer opens up Ethereum IDE, selects the Fast Lanes for a flow and pays through his Ethereum account. The instructions from Ethereum after

confirmation of payment and valid IP addresses goes to SDN controller to spin off a Fast Lane on network. Mechanism follows following steps.

1. Customer selects and pays a Fast Lane on Ethereum IDE
2. Ethereum IDE sends the instructions to SDN controller
3. Instructions are intercepted by SGX
4. SGX protects the instructions in Enclave and sends the secure version to SDN controller
5. SDN controller provisions a Fast Lane

## 4.2 Ryu SDN Controller Fast Lanes application

This Python written application in SDN controller is the main component which provision the Fast Lanes and implement network functionalities.

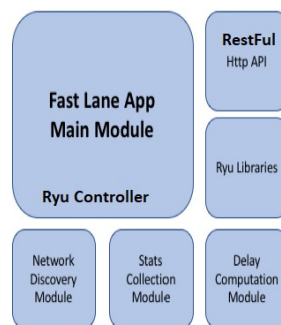


Figure 4.2: Component description of SDN controller application for Fast Lanes

It is divided in four sub modules for functional simplicity. Applications sub modules are illustrated in figure4.2. Each module provides a different function and pulls information off the network which helps to orchestrate the overall functionality of Fast Lanes application.

In sections to follow we will explain out each submodule in detail.

### **4.2.1 Network Discovery**

This module uses the Topology API to retrieve all the network information. i.e. number of switches, routers, links, ports, VLANs and sort the information out to develop a topology which further by using GUI creates a network wide topology illustration. The topology API uses LLDP packets and retrieve information from all the devices connected in the network.

### **4.2.2 Network Monitoring**

This module gather statistics from all devices connected in the network. Statistics include run time counter readings for bytes sent and received, hits on a port, aggregations of devices, active and disable flows, traffic load on each port and link etc. These statistics help the main module to define decision regions and actions on top of all.

### **4.2.3 Delay calculation**

This module calculate delays between network devices for all possible routes. The information statistics are stored in Python structures which are later used by main module to take decisions for Fast Lanes provisioning.

#### **4.2.4 Main module**

This module is the application on SDN controller which configures the Fast Lanes. In the initial phase, it takes the information of topology and network statistics using network discovery and network monitoring tool to arrange the network wide topology. Then it runs the delay calculation mechanism on top of topology to find out delay values through all possible routes then using K-shortest path algorithm it chooses the k-best paths for a flow based on pre-defined metrics i.e. hop count, delay or bandwidth.

### **4.3 SmartContract**

In this project SmartContract is deployed on public ledger to perform both as a payment gateway and user interface to invoke and revoke a Fast Lane. SmartContract is built on following modules

#### **4.3.1 Ethereum Virtual Machine**

We have used an Ethereum Virtual machine to implement SmartContract. It is a 256-bit stack which uses computing power from Ethereum nodes on distributed networks and provides a virtual machine-like environment to run scripts and functions.

#### **4.3.2 Remix IDE**

This interactive tool is used to access SmartContract's GUI, where customer is facilitated with options to select flow and cost is deducted and transferred



when customer interacts after providing his Ethereum address. If customer does not contain sufficient balance, the code does not execute Fast Lanes requests.

### 4.3.3 Web3 Library

This Python library uses JAVA Script API to interact with Ethereum. We leverage this library to pull data from SmartContract and save in a local storage where it is being updated whenever change occurs in SmartContract's output.

### 4.3.4 Ropsten blockchain

This blockchain for Ethereum platform is deployed alongside the primary production blockchain and is publicly available with the sole purpose to provide testing facility. We have developed our SmartContract on Ropsten network for testing purpose. Our SmartContract can similarly be deployed on production network if integrated for monetizing purpose.

## 4.4 Software guard extensions

OpenSGX is built on top of Qemu for trusted platform module functionality. SGX Enclave external interface takes the control data from SmartContract, save it in trusted memory zone where the data is immutable and only readable when invoked by a request initiated from inside the Enclave. This functionality has been implemented in C language with Sgplib inside the Enclave.

The data in protected memory zone in a read-only fashion is used by SDN controller to provision the Fast Lanes.

## 4.5 API server

API server is built to interact with multiple APIs of SDN controller to perform defined functionalities to implement Fast Lanes. API server from one side reads data (Control traffic) off SGX Enclave and on the other side call SDN APIs and feed this data for Fast Lanes provisioning. API server keeps an event based check on SmartContract data in SGX. Whenever the data is updated API server makes sure that the updated data is transferred to the SDN controller.

# Chapter 5

## Results and Observations

In this chapter we are going to observe and obtain desired results from our test bed implementation and discuss the significance of observations

### 5.1 SDN network topology

We have implemented network topology using Mininet network emulator. Nonetheless our controller application is topology independent, we have implanted topology illustrated in figure 5.2 to use as a reference point. Five switches are connected using links powered with Ethernet protocol on layer 2. All five switches are named as dpid:1, dpid:2, dpid:3, dpid4, dpid,5. Dpid1 is a stub switch which connects to the ring of dpid:2, dpid:3, dpid:4 and dpid:5. There is another link between dpid:4 and dpid:2 to induce complexity. Using Mininet we implemented these switches as OpenVswitch which are running on OpenFlow version 1.3 and backwards compatible with previous versions. Topology is hard-coded using Mininet's Python API mytopo. When SDN

```

mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=11.874s, table=0, n_packets=0, n_bytes=0, idle_age=11, priority=210,ip,nw_src=10.0.0.3,nw_dst=10.0.0.4 actions=NORMAL
 cookie=0x0, duration=11.91s, table=0, n_packets=0, n_bytes=0, idle_age=11, priority=210,ip,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x0, duration=11.833s, table=0, n_packets=24, n_bytes=4790, idle_age=1, priority=100 actions=drop
mininet>

```

Figure 5.1: Flow table entries in OpenFlow switch

controller connects with Mininet on loopback interface 127.0.0.1:6633, it retrieves the entire topology in Python structure using topology discovery module. Decision rules taken by SDN controller are enumerated in switches' flow table shown in figure 5.1 above.

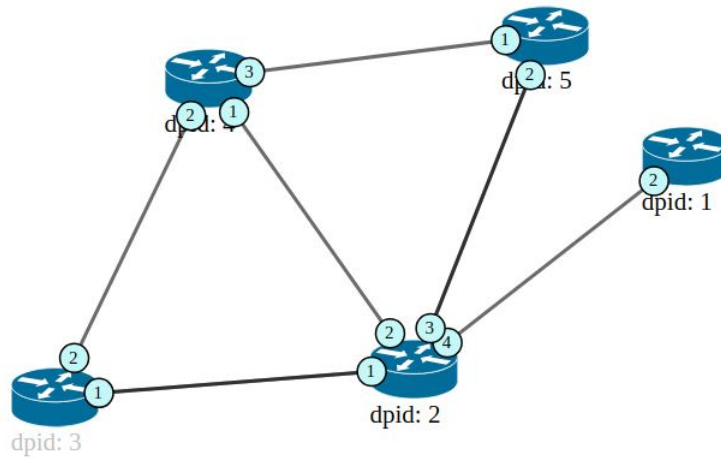


Figure 5.2: Network topology discovered by Ryu controller

## 5.2 SmartContract structure

We are using Remix IDE to compile and execute SmartContract. Remix is a publicly available ethereum ID which also provide user-interface to pass and retrieve function values of the SmartContract code.

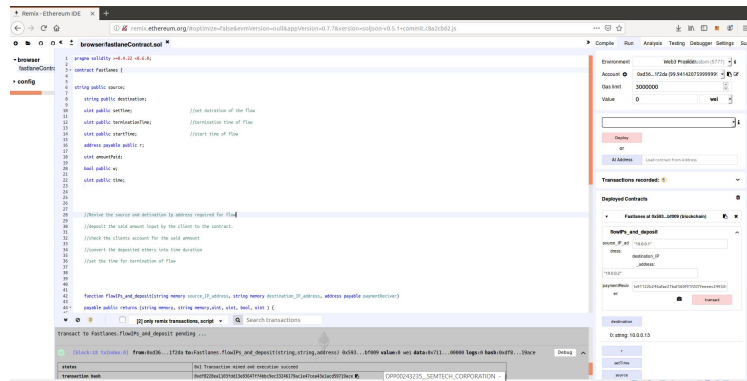


Figure 5.3: SmartContract implementation on Remix IDE

On the backend we have Ganache. It swiftly fires up a personal Ethereum blockchain or connects the back-end platform with public blockchain, which we can use to run tests, execute commands, inspect state of blockchain, and integrate decentralized applications to public blockchain.

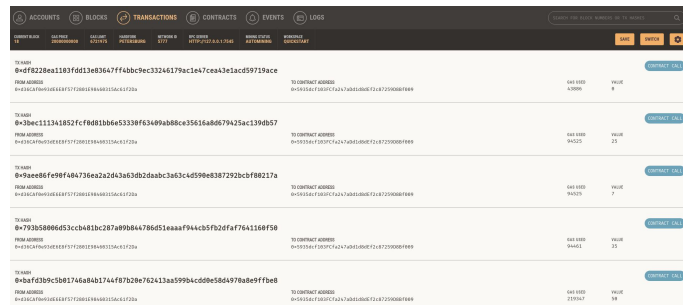


Figure 5.4: SmartContract implementation on Remix IDE

We are using Ganache (shown in figure5.4) to integrate web3.py Python

module with Ropsten blockchain and to manage and log transactions executed for the smartcontract.

### 5.3 Software Guard extensions

SGX module is placed between the SmartContract and SDN Controller API's server. Data coming off web3.py API of Ethereum is picked by SGX external interface and sent over to Enclave. Enclave encrypts the data and stores it in read-only memory space. This data in protected memory space is fed to the SDN controller through Pipe.

### 5.4 Testing scenario

A SmartContract is implemented on Ethereum virtual machine which provides customer the UI to interact with. Network topology is deployed using Mininet. All topology switches are connected to SDN Ryu controller's application through SSL connection on its TCP port 6633. Once the topology is loaded, handshaking process begins, and topology gets loaded in SDN's Python structure. Controller application will provision the Fast Lanes by running algorithm on stored topology with help of network awareness and delay calculator module once it receives the provisioning request on its API facing SGX. Invocation and revocation of Fast Lanes are implemented by controller app on the bases of request received from SmartContract via SGX module.

## 5.5 results

In the first step we initialize the Mininet topology by using Mininets custom application written on its Python's API. The Mininet application initialization is shown in figure 5.5.

```

munim@munim-VirtualBox:~/mininet$ sudo mn --custom custom/mesh.py --topo=mytopo --controller
remote --mac --switch ovs
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (h2, s4) (s2, s3) (s2, s4) (s3, s4) (s4, s5) (s5, s2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Starting CLI:
mininet>

```

Figure 5.5: Flow table entries in OpenFlow switch

After the topology implementation on Mininet gets completed we compute the SDN controller application on Ryu-Manager instance using following commands on a remote server. SDN controller application loads all the sub modules and components in the main instance shown in figure 5.6.

```

munim@munim-VirtualBox:~/ryu/ryu/app$ ryu-manager ofctl_rest.py watson/wk_shortest_forw
arding.py --k-paths=1 --weight=bw --observe-links
loading app ofctl_rest.py
loading app watson/wk_shortest_forwarding.py
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
creating context network_discovery
creating context dpset
creating context wsgi
creating context network_delay_calculator
creating context network_monitor
-----Topo Link-----
switch
(16785) wsgi starting up on http://0.0.0.0:8080

```

Figure 5.6: SDN Fast Lanes controller application and sub-modules initialization

Once application is up and running it retrieves the Mininet topology information and loads the topology in Python structure. Switches being

recognized by controller application are shown in figure 5.4.

```

----- Topo Link -----
switch
(17056) wsgi starting up on http://0.0.0.0:8080
switch:1 connected
switch:4 connected
switch:5 connected
switch:2 connected
switch:3 connected

```

Figure 5.7: switches and Ryu controller connection setup

After all switches' information is retrieved, controller instantiates the dispersion of LLDP packets back and forth between neighbor switches and construct a network-wide topology by using information arranged in switch connection matrix.

After the topology is learned, whole network is reachable to provision the Fast Lanes. Next step involves the request generation from SmartContract. In our project SmartContract also provide Getter functions which take the users values shown in figure 5.6 for Fast Lanes. User values in our case are source and destination IPs fastlane is going to be provisioned for, and payment receiver(ISP).

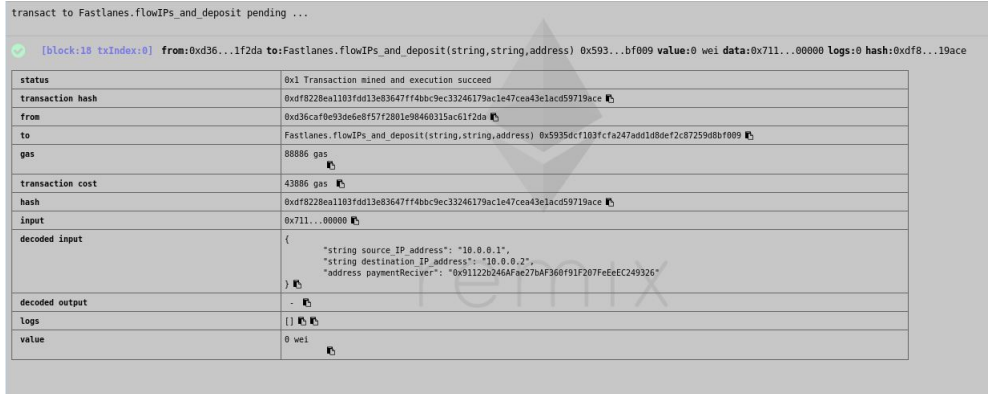
The screenshot shows a web interface titled "Deployed Contracts". A dropdown menu is open, showing a contract named "Fastlanes at 0x593...bf009 (blockchain)". Below the dropdown, there is a section titled "flowIPs\_and\_deposit". It contains several input fields: "source\_IP\_ad" with the value "10.0.0.1", "dress:" (likely a typo for "address:"), "destination\_IP" with the value "10.0.0.11", and "paymentReciver:" (likely a typo for "receiver:") with a long hexadecimal address. A "transact" button is located at the bottom right of the form.

Figure 5.8: SmartContract input functions on user-interface

SmartContract charges user some "wei" as a cost for Fast Lane. Once the transaction is fulfilled and users instructions meet the valid criteria following



dialog is shown for successful computation.



status	0x1 Transaction mined and execution succeed
transaction hash	0xdf8228ea1183fd13e83647ff4bbc9ec33246179ac1e47cea43elacd59719ace
from	0xd33caf9e93de6e8f572801e98468315ac61f2da
to	Fastlanes.flowIPs_and_deposit(string,string,address) 0x5935dcf103fca247add1d8def2c87259d8bf009
gas	88886 gas
transaction cost	43886 gas
hash	0xdf8228ea1183fd13e83647ff4bbc9ec33246179ac1e47cea43elacd59719ace
input	0x711...0000
decoded input	{ "string source_IP_address": "10.0.0.1", "string destination_IP_address": "10.0.0.2", "address paymentReceiver": "0x91122b246Afae27bAF368f91F207FeEeC249326" }
decoded output	-
logs	[]
value	0 wei

Figure 5.9: SmartContract successful transaction dialog

Using Web3.py API on a Python module, request to provision the Fast Lane is picked by the SGX. SGX will generate and apply signature key by using following command.

```
#Generating keypair for CPU and memory block
gen_key = "./opensgx -k "
subprocess.call(shlex.split(gen_key))
```

Figure 5.10: Generating key-pair for SGX Enclave

In the next step SGX encrypts the document and saves it in secure memory address space which is only accessible from the processor who is in possession of signature key.

```
#signing compiled version using key generated, it produce .CONF encrypted file
key_sign = "./opensgx -s user/demo/toSDN.sgx --key sign.key "
subprocess.call(shlex.split(key_sign))
```

Figure 5.11: Encrypting data using the key-pair and storing it in Enclave

the Fast Lane request data in secure memory blocks is immutable and accessed in read-only mode by script in main module (shown in figure5.8). Data

is accessed in read-only fashion and passed on to the SDN for provisioning of Fast Lanes on network. SGX terminal output is shown in Figure 5.8.

```

munim@munim-VirtualBox:~/opengsgx$ python sgxruntime.py
make: 'demo/toSDN.sgx' is up to date.
heap = 5010a000, size = 12bfff
{"10.0.0.1": "10.0.0.2", "10.0.0.3": "10.0.0.4"}

{"10.0.0.1": "10.0.0.2", "10.0.0.3": "10.0.0.4"}

-----
kern in count : 3
kern out count : 3
-----
encls count : 9675
ecreate count : 1
eadd count : 569
eextend count : 9104
einlt count : 1
eaug count : 0
-----
enclu count : 14
eenter count : 1
eresume count : 6
eexit count : 7
egetkey count : 0
ereport count : 0
eaccept count : 0
-----
mode switch count : 14
tlb flush count : 14
-----
TCS address : 4fffd000
Pre-allocated EPC SSA region : 0x2000
Pre-allocated EPC Heap region : 0x12c000
Later-Augmented EPC Heap region : 0x0
Total EPC Heap region : 0x12c000
heap = 5010a000, size = 12bfff
('FLOWS NEED TO BE ADDED', {'10.0.0.1': '10.0.0.2', '10.0.0.3': '10.0.0.4'})
<type 'dict'>


| % Total | % Received | % Xferd | Average Speed | Time   | Time     | Time     | Current  |
|---------|------------|---------|---------------|--------|----------|----------|----------|
|         |            |         | Dload         | Upload | Total    | Spent    | Left     |
| 100     | 12         | 100     | 129           | 0      | --:--:-- | --:--:-- | --:--:-- |
| 4       |            |         |               |        |          |          | 130      |


```

Figure 5.12: SGX terminal output for Enclave data (Fast Lanes provisioning request) is accessed

The data in the read-only address memory blocks is fed to the Ryu SDN controller application using its Rest API to provision a Fast Lane. An end-to-end Fast Lane is provisioned and shown in terminal window of SDN controller as shown in figure 5.14. Same steps are involved to revoke a Fast Lane.

```
[PATH]10.0.0.1<-->10.0.0.2: [1, 2, 4]
[PATH]10.0.0.2<-->10.0.0.1: [4, 2, 1]
-----Topo Link-----
  switch      1      2      3      4      5
    1          0      1      0      0      0
    2          1      0      1      1      1
    3          1      0      1      0      0
    4          1      1      0      0      1
    5          1      1      0      0      0
datapath      in-port  ip-dst  out-port packets  bytes  flow-speed(B/s)
-----
0000000000000001      1      10.0.0.2  2      0      0      17.9
0000000000000001      2      10.0.0.1  1      1      98      17.9
0000000000000002      3      10.0.0.2  4      1      98      17.9
0000000000000002      4      10.0.0.1  3      1      98      17.9
0000000000000004      1      10.0.0.1  4      0      0      17.9
0000000000000004      4      10.0.0.2  1      1      98      17.9
```

Figure 5.13: Provisioned Fast Lanes statistics on Ryu controller

# Chapter 6

## Conclusion

As the digitization is making its footprints in various domains, the demand of bandwidth per user is increasing day by day. To meet demand of increasing bandwidths, service providers must continuously expand their network infrastructure. The concept of Fast lanes has been in networking for many years. Traditional networks are basically combination of distributed entities which work on propriety rules and protocols and these networks provide bandwidth commitment and reservation using static methods, i.e. an engineer must provision a fast lane through all the nodes. The static method of provisioning takes time and due to the reservation of network resources, a factor of bandwidth goes unused when a Fast Lane's client is not sending traffic, while other users are having contention for rest of the bandwidth. Where on one hand this method is a hindrance in catering to the increasing demand of bandwidth per user in the era of digitization, it is also a time taking process to provision a Fast Lane, and hence becomes unsuitable for mission critical applications. The other major problem is a monetizing of

network services. Now the end-user's requirements have been evolved, they have different network requirements at different times depending on their platforms and end applications which becomes difficult to achieve with traditional monetizing platforms, as traditional monetizing platforms are mostly provided by banking sectors add an overhead in the process of acquiring a network service. It also gives the 3rd party some visibility over the network, and moreover service provider has to pay a huge sum to acquire monetizing platform for its network.

This research was focused on Software defined networking and Ethereum blockchains. Because of decoupling of control plane and data plane in SDN it becomes easy to manage and monitor the network centrally. The controller can be installed on a commodity hardware and network nodes are simply dump layer 2 boxes who forward packets on the bases of rules instructed by centralized controller. This paradigm not only comforts the service provider on revenue front, it also provides with the facility to program network functions and test them on a network slice before implementing them in production, on the same network. Blockchain has provided the pivotal role in cutting financial expenditures on service providers end. Smart Contract on distributed ledger has circumvented the middleman problem. In chapter 3 we have thoroughly discussed the role of Ethereum based monetizing platform. The execution of a program on Ethereum virtual machine has allowed us to provide the control of network directly to the customer and brought the provisioning time to a drastically low level. SDN and Ethereum network together can provide optimal solution for fast lanes provisioning as well as monetizing of network and due to their open nature, they can be programmed

and shaped to meet changing requirements.

Positioning SGX between SDN and Ethereum provides reliable communication for control traffic and can be used as trusted platform module for various functionalities. In chapter 5 we have shown step by step implementation of actualizing fast lanes using SDN and Ethereum. Multiple open source platforms have been put together to implement the prototype. Because of SDN and Ethereum flexibility and Open source approach the prototype is salable to thousand of devices with minimal changes.

# Chapter 7

## Future work

Although the blockchains have been around for many years but Ethereum has brought a different approach to decentralization by providing a virtual machine on to the blockchains. This platform can be anything depending on what we can program on it and gives the opportunity to create robust and agile decentralized applications. As in our case, we used Ethereum to write smart contract which is kind of a banking service as well as financial arbitration based on predefined terms. SDN and Ethereum in our prototype has a huge potential to expand and mold to meet various service provider network requirements and functionalities. In this on-demand fast lanes actualization through smart contract system there are many potential improvements which can enhance the capability and functionality of the system. The foremost areas of potential improvements are mentioned below.

## 7.1 OpenSGX compatibility

OpenSGX the emulation of Intel SGX has many hardware and operating system constraints as it is supported on few versions of Ubuntu x64 OS and it has limited community support. The computation time of SGX is also significantly high. We firmly believe, compatibility and computation time improvements will lead to better Enclave performance and research breakthroughs in various aspects.

## 7.2 Topology discovery

Our Fast Lane application on Ryu controller use the conventional LLDP dispersion logic to discover network topology which is naive to be implemented on production level. This can be improved by implementing loop avoidance mechanism along side the LLDP discovery function in the SDN controller application.

## 7.3 Smart contract

Since smart contract is a monetizing platform implemented on distributed ledger which cuts the role of financial arbitrator it also gives rise to another problem in our system. There is no conflict-resolve mechanism defined in our prototype. If client invokes a fast lane on using Ethereum platform the request goes through, and client starts paying fractions of ether for it, but if SDN on the side fails to provide service due to network outage or any other malfunctioning client is still paying as there is no feedback system between



smart contract and SDN. A system should be introduced in smart contract to check service delivery of the network.

# Bibliography

- [1] Benjamin B. Bauer and Andrew S. Patrick. A human factors extension to the seven-layer osi reference model. 2002.
- [2] Norbert Egi, Adam Greenhalgh, Mark Handley, Mickal Hoerdtd, Felipe Huici, and Laurent Mathy. Towards high performance virtual routers on commodity hardware. page 20, 01 2008.
- [3] 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.
- [4] Christian Esteve Rothenberg, Roy Chua, Josh Bailey, Martin Winter, Carlos N. A. Correa, Sidney C. de Lucena, Marcos Rogerio Salvador, and Thomas D. Nadeau. When open source meets network control planes. *Computer*, 47(11):46–54, November 2014.
- [5] Maciej Kuzniar, Peter Peresini, and Dejan Kostic. What you need to know about sdn flow tables. *Lecture Notes in Computer Science (LNCS)*, page 13, 2015.

- [6] ONF. Software-defined networking: The new norm for networks. Technical report, Open Networking Foundation, April 2012.
- [7] Alexander Shalimov, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov, and Ruslan Smeliansky. Advanced study of sdn/openflow controllers. In *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, CEE-SECR '13*, pages 1:1–1:6, New York, NY, USA, 2013. ACM.
- [8] The Open Networking Foundation. OpenFlow Switch Specification, Jun. 2012.
- [9] 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, March 2008.
- [10] AA Lazar, Koon-Seng Lim, and F. Marconcini. Realizing a foundation for programmability of atm networks with the binding architecture. *Selected Areas in Communications, IEEE Journal on*, 14(7):1214–1227, Sep 1996.
- [11] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '07*, pages 1–12, New York, NY, USA, 2007. ACM.

- [12] David Klingel, Rahamatullah Khondoker, Ronald Marx, and Kpatcha Bayarou. Security analysis of software defined networking architectures: Pce, 4d and sane. In *Proceedings of the AINTEC 2014 on Asian Internet Engineering Conference*, AINTEC '14, pages 15:15–15:22, New York, NY, USA, 2014. ACM.
- [13] 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, July 2008.
- [14] Martin Casado, Nate Foster, and Arjun Guha. Abstractions for software-defined networks. *Commun. ACM*, 57(10):86–95, September 2014.
- [15] J. Medved, R. Varga, A. Tkacik, and K. Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *2014 IEEE 15th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 1–6, Los Alamitos, CA, USA, jun 2014. IEEE Computer Society.
- [16] Chang Liu, Arun Raghuramu, Chen-Nee Chuah, and Balachander Krishnamurthy. Piggybacking network functions on sdn reactive routing: A feasibility study. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 34–40, New York, NY, USA, 2017. ACM.
- [17] Rob Sherwood, Glen Gibb, Kok kiong Yap, Martin Casado, Nick Mckeown, and Guru Parulkar. Flowvisor: A network virtualization layer. Technical report, 2009.

- [18] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, and Teresa Vazao. Towards programmable enterprise wlangs 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.
- [19] Wikipedia contributors. Network operating system — Wikipedia, the free encyclopedia, 2019. [Online; accessed 20-August-2019].
- [20] Jehad Ali, Seungwoon Lee, and Byeong-hee Roh. Performance analysis of pox and ryu with different sdn topologies. In *Proceedings of the 2018 International Conference on Information Science and System*, ICISS '18, pages 244–249, New York, NY, USA, 2018. ACM.
- [21] Rajat Jain and Rahamatullah Khondoker. *Security Analysis of SDN WAN Applications—B4 and IWAN*, pages 111–127. Springer International Publishing, Cham, 2018.
- [22] Justin Pettit, Ben Pfaff, Joe Stringer, Cheng-Chun Tu, Brenden Blanco, and Alex Tessmer. Bringing platform harmony to wmvare nsx. *SIGOPS Oper. Syst. Rev.*, 52(1):123–128, August 2018.
- [23] Jesse Yli-Huumo, Deokyoon Ko, Sujin Choi, Sooyong Park, and Kari Smolander. Where is current research on blockchain technology? a systematic review. *PLOS ONE*, 11(10):1–27, 10 2016.
- [24] Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. Blockchain. *Business Information Systems Engineering: The International Journal of WIRTSCHAFTSINFORMATIK*, 59(3):183–187, 2017.

- [25] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger eip-150 revision (759dccd - 2017-08-07), 2017. Accessed: 2018-01-03.
- [26] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.
- [27] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform, 2014. Accessed: 2016-08-22.
- [28] Piotr Piasecki. Gaming self-contained provably fair smart contract casinos. *Ledger*, 1(0):99–110, 2016.
- [29] Marko Vukolić. Rethinking permissioned blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, BCC '17, pages 3–7, New York, NY, USA, 2017. ACM.
- [30] Peter Robinson, David Hyland-Wood, Roberto Saltini, Sandra Johnson, and John Brainard. Atomic crosschain transactions for ethereum private sidechains. *CoRR*, abs/1904.12079, 2019.
- [31] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 270–282, New York, NY, USA, 2016. ACM.
- [32] Prerit Jain, Soham Desai, Seongmin Kim, Ming-Wei Shih, JaeHyuk Lee, Changho Choi, Youjung Shin, Taesoo Kim, Brent Byunghoon Kang, and Dongsu Han. OpenSGX: An Open Platform for SGX Research. In

*Proceedings of the Network and Distributed System Security Symposium*,  
San Diego, CA, February 2016.

- [33] Daniel Bartholomew. Qemu: A multihost, multitarget emulator. *Linux J.*, 2006(145):3–, May 2006.