

Aalto University
School of Science

Master's Degree Programme in Security and Mobile Computing

Rajat Kandoi

Deploying Software-Defined Networks: a Telco Perspective

Master's Thesis
Espoo, July 15, 2015

Supervisors: Professor Tuomas Aura, Aalto University, Finland
Professor Markus Hidell, KTH Royal Institute of Technology, Sweden
Advisors: Markku Antikainen M.Sc. (Tech.), Aalto University, Finland
Sumanta Saha M.Sc. (Tech.), Ericsson, Finland

Aalto University
School of Science
Master's Degree Programme in Security and Mobile Computing

ABSTRACT OF MASTER'S THESIS

Author:	Rajat Kandoi	
Title:	Deploying Software-Defined Networks: a Telco Perspective	
Date:	July 15, 2015	Pages: 77
Professorship:	Data Communication Software	Code: T-110
Supervisors:	Professor Tuomas Aura Professor Markus Hidell	
Advisors:	Markku Antikainen M.Sc. (Tech.) Sumanta Saha M.Sc. (Tech.)	
<p>Software-Defined Networking (SDN) proposes a new network architecture in which the control plane and forwarding plane are decoupled. SDN can improve network efficiency and ease of management through the centralization of the control and policy decisions. However, SDN deployments are currently limited to data-center and experimental environments. This thesis surveys the deployment of SDN from the perspective of a telecommunication network operator. We discuss the strategies which enable the operator to migrate to a network in which both SDN and legacy devices interoperate. As a synthesis of existing technologies and protocols, we formulate an automated process for the bootstrapping of newly deployed forwarding devices. Furthermore, we review solutions for programming the forwarding devices and for performing topology discovery. The functional correctness of the proposed bootstrapping process is evaluated in an emulated environment.</p>		
Keywords:	SDN, bootstrap, southbound protocols, topology discovery	
Language:	English	

Acknowledgements

First and foremost, I would like to thank my supervisors and advisors for their guidance and support. The weekly discussions with Tuomas Aura and Markku Antikainen were highly constructive and guided my research direction. Markus Hidell provided deep insights into the structure and content of the report. Sumanta Saha's guidance has allowed me to apply the knowledge I have acquired through the thesis immediately into my work at the industry.

I would also like to thank my manager, Antti Miettinen. He constantly followed up on my progress and was obliging in allowing me to work remotely from the university on several occasions.

Lastly, I would like to thank my family for their support; my wife who kept me motivated to complete the thesis work in time, and my parents, without whom none of this would have been possible.

Espoo, July 15, 2015

Rajat Kandoi

Abbreviations and Acronyms

ACL	Access Control List
API	Application Programming Interface
ARP	Address Resolution Protocol
AS	Autonomous System
BGP	Border Gateway Protocol
BGP-LS	Border Gateway Protocol Link State
BOOTP	Bootstrap Protocol
BDDP	Broadcast Domain Discovery Protocol
CA	Certificate Authority
CLI	Command Line Interface
CMP	Certificate Management Protocol
CRL	Certificate Revocation List
DHCP	Dynamic Host Configuration Protocol
DN	Distinguished Name
DSL	Digital Subscriber Line
DSLAM	Digital Subscriber Line Access Multiplexer
FIB	Forwarding Information Base
GMPLS	Generalized Multiprotocol Label Switching
HTTP	Hypertext Transfer Protocol
I2RS	Interface To Routing System
IETF	Internet Engineering Task Force
IAK	Initial Authentication Key
ICT	Information and Communications Technology
IGP	Interior Gateway Protocol
IP	Internet Protocol
IS-IS	Intermediate System - Intermediate System
IPSec	IP Security
L2	Layer 2 (of the network protocol stack)
L3	Layer 3 (of the network protocol stack)
LAN	Local Area Network

LLDP	Link Layer Discovery Protocol
LSP	Label Switched Path
LSPDB	Label Switched Path Database
LTE	Long-Term Evolution
MAC	Media Access Control
MIB	Management Information Database
MPLS	Multiprotocol Label Switching
NFV	Network Functions Virtualization
NLRI	Network Layer Reachability Information
NMS	Network Management System
NOC	Network Operations Center
OID	Object Identifier
OSPF	Open Shortest Path First
OTN	Optical Transport Network
OS	Operating System
PCC	Path Computation Client
PCE	Path Computation Element
PCEP	Path Computation Element Communication Protocol
PE	Provider Edge
PKI	Public Key Infrastructure
PKIX	Public Key Infrastructure X.509
PON	Passive Optical Network
PSK	Pre-shared key
RA	Registration Authority
SCEP	Simple Certificate Enrollment Protocol
SDN	Software-Defined Networking
SNMP	Simple Network Management Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
STP	Spanning Tree Protocol
TCP	Transmission Control Protocol
TED	Traffic Engineering Database
TFTP	Trivial File Transfer Protocol
TLS	Transport Layer Security
TLV	Type-Length-Value
URL	Uniform Resource Locator
USB	Universal Serial Bus
VLAN	Virtual Local Area Network
XML	Extensible Markup Language

Contents

Abbreviations and Acronyms	6
1 Introduction	11
1.1 Problem statement	11
1.2 Motivation	12
1.3 Contribution	12
1.4 Research methodology	13
1.5 Sustainable development aspects	13
1.6 Structure of the thesis	14
2 Background on network technologies	15
2.1 Introduction to SDN	15
2.2 Network sections	18
2.3 Public key infrastructure	18
2.3.1 PKI model	20
2.3.2 Certificate management protocols	21
2.3.3 Motivation for using digital certificates	22
2.4 Southbound protocols	22
2.4.1 OpenFlow	23
2.4.2 Path computation element communication protocol (PCEP)	25
2.4.3 Interface to routing system (I2RS)	25
2.4.4 BGP flow-spec	27
2.4.5 Simple network management protocol (SNMP)	27
2.4.6 NETCONF	28
2.5 Protocols for topology discovery	29
2.5.1 Simple network management protocol (SNMP)	30
2.5.2 Link layer discovery protocol (LLDP)	30
2.5.3 BGP link-state (BGP-LS)	31

3	SDN deployment solutions	33
3.1	Setting up new hardware	34
3.2	Bringing up a device	35
3.2.1	Enabling IP connectivity	35
3.2.2	Enabling Remote Management	37
3.3	Bootstrapping security	37
3.3.1	Installing the operator root certificate	37
3.3.2	Provisioning a domain certificate	38
3.3.3	Bringing up a remote device outside the secure opera- tor network	40
3.3.4	Enhanced security with logging servers	40
3.4	Topology discovery	42
3.4.1	Topology discovery in pure SDN networks	42
3.4.2	Topology discovery in pure legacy networks	44
3.4.3	Topology discovery in hybrid SDN networks	44
3.4.4	Limitations of discussed protocols	46
3.5	Packet forwarding	46
3.5.1	Forwarding in aggregation and core networks	46
3.5.2	Unified control of transport and IP networks	47
3.5.3	Centralized inter-domain routing	48
3.6	Summary	50
4	Implementation and evaluation	51
4.1	Emulation testbed	51
4.1.1	Emulation topology	52
4.1.2	Tools used	52
4.2	Bootstrapping process	53
4.2.1	Infrastructure preparation	54
4.2.2	Initial switch configuration	54
4.2.3	Integrating the switch into the network	54
4.3	Further improvements	55
5	Discussion	59
5.1	Meeting carrier grade requirements	59
5.2	Managing SDN networks after deployment	60
5.3	Reflections on the implementation	62
5.4	Future work	62
6	Conclusion	63

Chapter 1

Introduction

Software-defined Networking (SDN) is an emerging networking paradigm with a great potential to foster innovation through programmable networks. SDN networks are characterized by the separation of the control and data planes wherein a logically centralized controller performs routing decisions on behalf of forwarding elements. SDN has gained a lot of attention from network operators, equipment vendors and over-the-top application service providers. However, current deployments are limited to data-center and experimental university environments [62]. This thesis focuses on the deployment of SDN in *telecommunication networks*.

1.1 Problem statement

SDN offers operators many benefits such as centralized control, improved network efficiency and lower operational costs. However, telecommunication networks are complex in nature. They consist of heterogeneous legacy devices and support several different technologies. There is a demand for strong security as devices often communicate over insecure networks. Also, advanced administration and management support is required to operate the networks. Operators need to account for all these complexities before deploying SDN.

In this thesis, we answer the question “What are the considerations for deploying software-defined networks in telecommunication networks?”. We investigate the migration path of the network technologies, protocols and deployment process to an SDN-enabled network both from a theoretical and an engineering perspective.

1.2 Motivation

The work carried out in this thesis is motivated by the following factors:

- SDN requires programmable forwarding hardware. However, to protect their current investments, operators may be reluctant to replace all legacy network equipment at once [111]. It seems appropriate to investigate *incremental* SDN deployment in which SDN-enabled and legacy nodes need to interoperate. The goal of such a strategy is to allow realizing SDN's benefits as soon as the first new nodes are powered up.
- Network nodes are often placed in untrusted environments. However, security is often ignored due to complexity of the process of provisioning security material, or simply due to operational expenses [22]. In an SDN network, security associations are required between a controller and forwarding devices. It is worth investigating *automatic* certificate installation as it would be valuable in easing the deployment of security mechanisms.
- Communication between the SDN controller and forwarding devices occurs over so-called *southbound protocols*. With several southbound protocols available (e.g. OpenFlow, PCEP, NETCONF), it is worth studying the applicability and benefits of these protocols.
- Discovering the physical network topology is crucial in SDN networks as this knowledge is required to make routing decisions. It is important to study how such topology information can be obtained and if existing protocols can be used to achieve this.

1.3 Contribution

This thesis provides a comprehensive survey of key aspects for deploying SDN in telecommunication networks. First, we examine different strategies with which network operators can deploy the new SDN hardware equipment in their networks. We contrast clean-slate and incremental deployment strategies. Second, we investigate bringing up the newly installed devices. We present the engineering process and the associated infrastructure needed to perform this task. We also discuss how digital certificates can be installed in an automated manner through the use of a public key infrastructure and pre-installed device certificates. Third, we analyze different southbound protocols and investigate the applicability of these protocols to different parts of

the network. Finally, we discuss how existing protocols can be used to construct the network topology in a network comprising of both SDN and legacy devices. In addition to the literature survey, we provide a proof-of-concept implementation which demonstrates an automated process to bootstrap an SDN device and provision security material.

1.4 Research methodology

This thesis presents an engineering approach to the problem of SDN deployment. It builds upon existing protocols and technologies described in technical standards and research literature. We study a large number of protocols and solutions and design our solution using a subset of these. We then evaluate the validity of the proposed process through a proof-of-concept implementation and by testing it in an emulated network environment. This demonstrates how the presented approach enables SDN nodes and networks to be brought up and configured in an automated manner.

The approach used in this thesis can be classified as *experimental computer science* and is typical of computer networks and systems research. Experimental computer science aims at demonstrating and evaluating the feasibility of solutions to a problem [48, 56] through construction of prototype systems [44]. Our approach also resembles that of *constructive research* where the aim is solve a domain specific problem or to create knowledge about how the problem can be solved [43]. Additionally, our prototype implementation methodology can further be classified as *emulation* as we “build a set of synthetic experimental conditions for executing a real application” [61].

1.5 Sustainable development aspects

Telecommunication networks represent a significant portion of the global ICT power consumption [116]. Fig. 1.1 shows the increasing trend of power consumption in different parts of operator networks.

There is a growing need to design new network paradigms that enable the same ICT functionality while consuming lower amounts of energy in the future [110]. SDN can contribute significantly in achieving this goal. By moving control-plane decisions to a centralized location, forwarding devices no longer require the intelligence to perform computation tasks. Also, it is expected that SDN will allow improved utilization of the network resources. These factors will help decrease energy consumption of the network devices.

Furthermore, this thesis advocates for the incremental deployment of

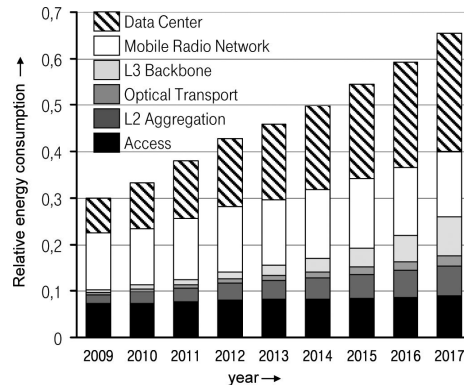


Figure 1.1: Energy consumption of operator's network [70]

SDN. With this approach, legacy hardware can be fully utilized till end of its effective lifetime. This method is in line with the green principle – reduce, reuse, recycle.

SDN also has far-reaching impact on economic sustainability of network operators. By moving functionality from hardware to software, SDN will accelerate service creation and reduce time to market. With this potential for new innovation, SDN will enable the creation of new business. Additionally, this thesis presents a method for the automated bootstrapping of SDN devices in the network. This will allow network operators to reduce their operational expenses, which leads to competitive advantages and cost savings to the service customers.

1.6 Structure of the thesis

The rest of the thesis is organized as follows: Chapter 2 provides the necessary background information to understand the work carried out in the thesis. It describes the network technologies and protocols used in the thesis. Chapter 3 contains our survey of the SDN deployment solutions. We provide the engineering process of deploying new network devices and also discuss how the devices can be configured for packet forwarding in an automatic manner. Chapter 4 provides details on the implementation of our prototype for the automatic integration of an SDN switch into the network. This demonstrates the functional correctness of our proposed approach. Chapter 5 discusses considerations for meeting carrier-grade requirements and presents network management tasks that can benefit from SDN. We also highlight lessons learned from the thesis and provide a direction for future work. Lastly, chapter 6 summarizes the thesis and provides concluding remarks.

Chapter 2

Background on network technologies

This chapter provides the required background information considered important to understand the work presented in this thesis. The thesis covers a broad range of issues related to the deployment of SDN in operator networks. This large scope is reflected by the diverse nature of the topics covered in this chapter. First we present the concept of software-defined networking and discuss how SDN differs from legacy networks. Then we discuss the architecture and network segments of a telecommunication network. Subsequently, we review Public Key Infrastructure (PKI) and describe the PKI model used in the thesis. We then examine various protocols the SDN controllers and forwarding devices use to communicate with each other. Lastly, we discuss protocols that enable gathering network topology information.

2.1 Introduction to SDN

Software-defined networking is a new paradigm which revolutionizes network architecture through the introduction of a software-controlled, programmable forwarding plane. Traditional networking devices are typically autonomous in nature. Each device hosts its own operating system, runs distributed control-plane protocols and builds a local network state. The operating system, which is often proprietary, consults the local network state and configures specialized forwarding hardware through proprietary application programming interfaces (API) [40]. SDN, on the other hand, eliminates these control-plane operations from network devices and moves the operating system to a logically centralized *controller*, also referred to as the network operating system. The controller exposes the network state learned from the

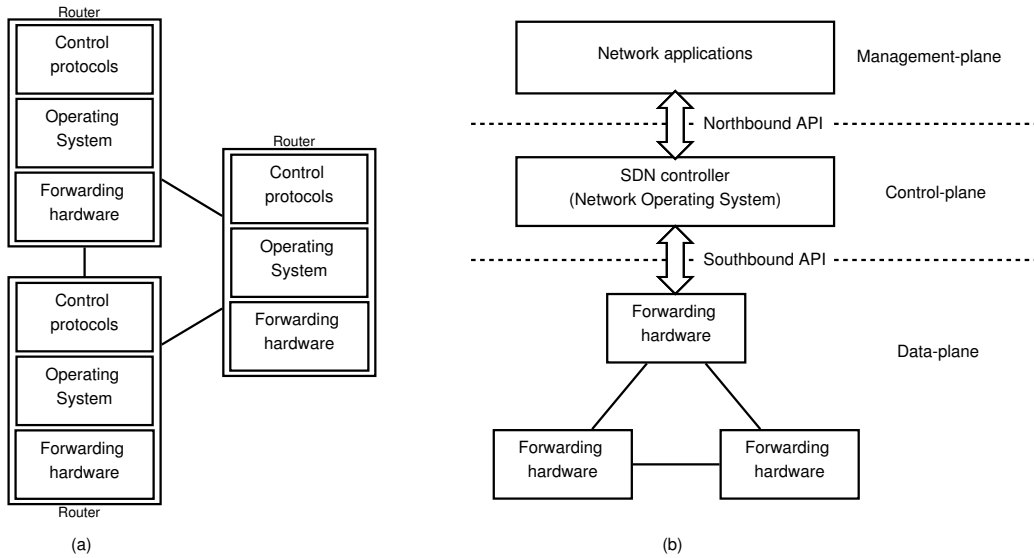


Figure 2.1: (a) Traditional network and (b) SDN network

forwarding devices to software-based *network applications*. Routing decisions are made by the applications and communicated to the controller, which in turn translates these decisions in to forwarding rules and programs the appropriate devices. Forwarding devices then perform packet header matching against these rules to determine the port on which to send a packet out.

Communication between the network applications and the controller occurs over so called *northbound* APIs. Communication between the controller and forwarding devices occurs over *southbound* APIs. The forwarding devices constitute the *data-plane*, the controller constitutes the *control-plane*, and the networks applications form the *management-plane* [69]. Fig. 2.1 depicts the difference between traditional networks and SDN networks.

SDN can also be defined in terms of three abstractions: forwarding abstraction, distribution abstraction, and specification abstraction [69, 98].

- The *forwarding abstraction* allows network applications to make routing decisions without knowing any details of the underlying hardware. This is achieved through the use of open and standardized protocols for the communication with the forwarding devices.
- The controller implements the *distribution abstraction*. This abstraction is essentially responsible for two tasks. First, it is responsible for installing forwarding rules on the network devices. Secondly, it gathers

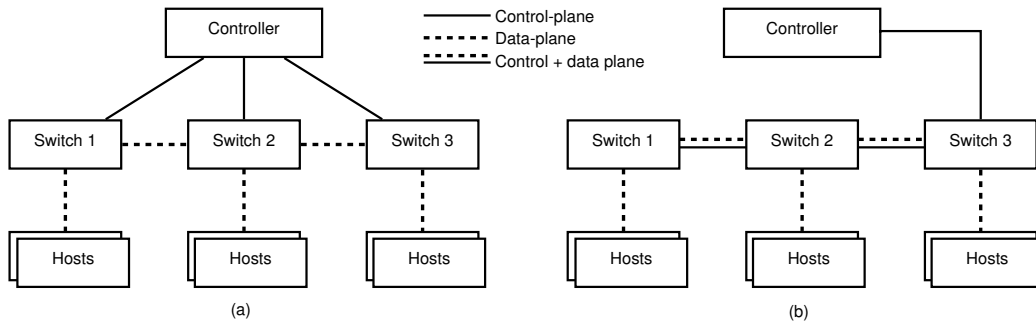


Figure 2.2: (a) Out-of-band and (b) in-band control plane

information about the forwarding layer and exposes this state information to network applications thereby allowing them to build a global network view.

- The *specification abstraction* allows network applications to express desired network behavior without being responsible for the actual implementation of the behavior itself.

Physical connections between the forwarding devices and the controller can be set up in two ways: out-of-band and in-band [16]. In the out-of-band scheme, each forwarding device has a dedicated physical connection to the controller whereas in the in-band scheme, control-plane information is carried over existing data-plane connections between the forwarding devices. These scenarios are depicted in Fig. 2.2. Note that forwarding devices as referred to as switches which in computer networks terminology are typically layer-2 devices. In the context of SDN, a switch is a forwarding device which performs packet header matching regardless of the layer to which the header belongs; for instance forwarding can be based on layer-2 MAC addresses, layer-3 IP addresses, layer-4 port numbers, or a combination of all these fields. In SDN literature, the forwarding devices are nevertheless called switches as a reference to the single forwarding function that they perform.

SDN offers network operators many advantages. *Decoupling* the control and data planes allows the forwarding devices to be manufactured at lower costs since they no longer require the computing intelligence to perform control-plane processing [16]. The *centralized* control allows the controller to maintain an up-to-date view of the full network topology. The exposing of this network state to software applications enables better informed forwarding decisions [69]. *Softwarization* of the forwarding decisions accelerates innovation and service creation. Network operators no longer need to wait for standardization and implementation of new protocols. Instead, new functionality can be deployed as plug-and-play software modules [66].

2.2 Network sections

An operator's telecommunication network is typically structured into the access network, aggregation network and core network [70]. These networks operate on packets of data which represent layer-2 and above of the network protocol stack. The data is carried over the transport network, which consists of optical switches and optical routers, and represents the physical layer (layer-1). These network sections are depicted in Fig. 2.3.

- The *access* network, as the name suggests, is part of the network that provides homes and enterprises with access to network services such as voice, video and data. Access technologies include xDSL (e.g. ADSL2, VDSL), optical access (e.g. PON, point-to-point fiber) and wireless (e.g. 3G, LTE). Connections from end-users terminate at an Ethernet switch, which we refer to as the *access edge switch*. It represents the first point of entry for user traffic in to the operator network.
- The *aggregation* network provides traffic aggregation from the access network. It consists of layer-2 Carrier Grade Ethernet and an underlying optical transport network (OTN). Typically, the logical network topology is a tree-like arrangement of Ethernet switches [70], which are physically interconnected via metro or region OTN rings [9]. The aggregation network uses VLAN or Multiprotocol Label Switching (MPLS) based technologies to route the traffic to the core network [11, 84].
- The *core* network constitutes a layer-3 IP-MPLS backbone. The logical topology is typically a partial mesh network and the underlying OTN can be a ring or full mesh network [9]. We refer to the border node which interfaces with the global Internet or other peer networks as a *provider edge (PE) router*.

2.3 Public key infrastructure

Public key infrastructure is a set of technical mechanisms, procedures and policies that collectively enable deployment of security services [106]. PKI is built on public key cryptography and digital certificates. Trusted nodes known as Certificate Authorities (CA) issue digital certificates to clients of the PKI (a human user, a server or a client machine). This allows clients to learn other clients' public keys securely. This is achieved as follows. The CA typically has a self-signed certificate, i.e., the issuer and subject fields of the certificate are the same and contain the CA's distinguished name. The

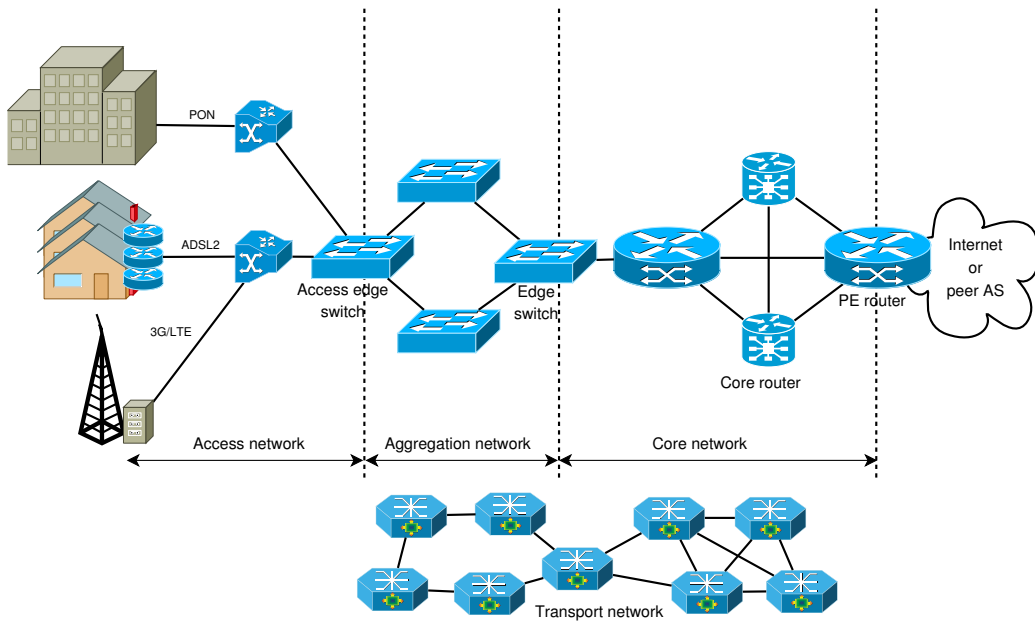


Figure 2.3: Network sections in a telecommunication network

public key field contains the CA’s public key and the certificate is signed with its private key. A client certificate issued by the CA contains the client’s distinguished name in the subject field and the CA’s distinguished name in the issuer field. The client’s public key is inserted in the public key field and the certificate is signed with the CA’s private key. In this way, a digital certificate creates a mapping between names and public keys [89]. Fig. 2.4 depicts the fields and their values as described above.

Now let us assume Alice wants to prove her identity to Bob. In order to do so, Alice presents her CA-issued certificate to Bob. We assume that Bob trusts this CA and possesses a copy of the CA’s certificate (and thereby

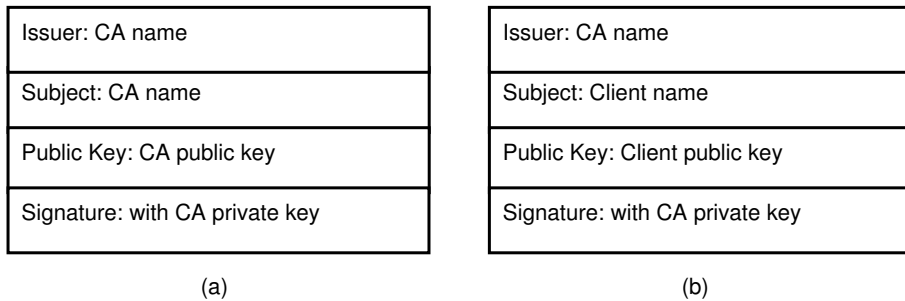


Figure 2.4: (a) Self-signed CA certificate and (b) CA-issued client certificate

the CA's public key). When Bob verifies the signature on Alice's certificate using the CA's public key, he can be sure that the correct CA (the one he trusts) has issued the certificate. Consequently, he trusts Alice's public key.

In a PKI, registration authorities are used to offload some of the work handled by CAs. However, for simplicity we omit them from our discussion. A more detailed discussion on PKI can be found in [65] and [24]. In the following sections, we introduce the PKI model used in this thesis and provide an overview of certificate management protocols.

2.3.1 PKI model

The PKI model considered in this thesis consists of the following actors and certificate profiles (depicted in Fig. 2.5):

- Vendor CA – The vendor CA is controlled by the manufacturer of the network equipment. It has a self-signed certificate which we refer to as the *vendor root certificate*. It issues a *device certificate* to every manufactured device. The vendor is responsible for installing both the vendor root certificate and the device certificate on the manufactured devices.
- Operator CA – The operator CA is controlled by the network operator and has a self-signed certificate, which we refer to as the *operator root certificate*. The operator CA is responsible for making decisions to allow or reject a new device from joining the network domain. If the device is allowed to join the domain, it issues a *domain certificate* to the device.
- New device – This is the device that wants to connect to the network. It is pre-installed with the vendor root certificate and a device certificate. When a device wants to join the network, it first obtains a copy of the operator root certificate. Then it requests the operator CA to issue a domain certificate to itself.

We assume that the new device connects to other nodes (for instance, the SDN controller) which are under control of the same network operator, i.e. the common trust anchor for all devices is the operator CA. Furthermore, we assume these nodes have already been registered to the network and possess a copy of the operator root certificate and a domain certificate signed by the operator CA.

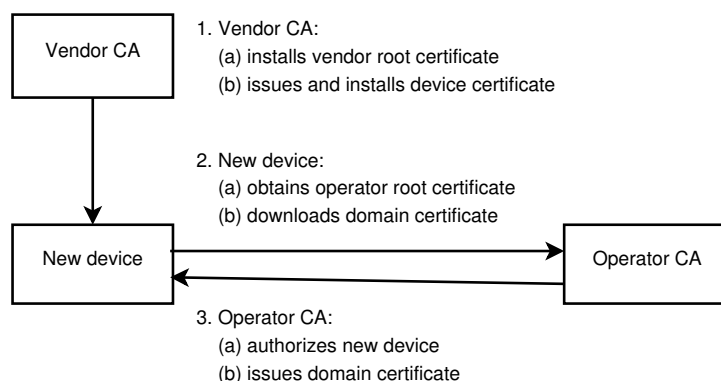


Figure 2.5: PKI actors and certificate profiles

2.3.2 Certificate management protocols

Certificate management protocols are used to support online exchanges of messages between clients and the CA in order to facilitate functions such as certificate enrollment, certificate revocation, key pair update, and key pair recovery. Several certificate management protocols have been developed over the years. The effort has mainly been driven by the Internet Engineering Task Force (IETF) Public Key Infrastructure X.509 working group and has been realized in the form of two management protocols – Certificate Management Protocol (CMP) [13] and Certificate Management Messages over CMS (CMC) [96]. The basic functionality of these two protocols is essentially the same [6] although CMP is the more comprehensive and widely deployed protocol of the two. In addition to these protocols, Cisco has developed the Simple Certificate Enrollment Protocol (SCEP) [90]. Despite being an Internet draft for almost a decade, the protocol has seen widespread deployment.

In this section, we discuss how the CA can automatically authorize client certificate requests. In order to do so, CMP and SCEP make use of an out-of-band shared secret, referred to as the *challengePassword* in SCEP and the *initial authentication key* in CMP. The CA generates a shared secret and saves a binding between the secret and the requesting client's name. The shared secret is communicated securely to the client before it makes the request.

In SCEP, the entire request is encrypted using the CA's public key and the challengePassword is included. Upon decrypting the request and comparing the challengePassword with its database, the CA authorizes the certificate request if the passwords match. In CMP, the initial authentication key is used to protect the certificate request. If the CA is able to verify the message hash using its own copy of the initial authentication key, it can then automatically

authorize the request.

Both protocols also provide mechanisms to automatically authenticate the clients based on certificates issued by other CAs; for instance, the operator CA can authorize a device based on the device certificate. However, a trust relation between the operator CA and the vendor CA must be established in advance. Using device certificates to automatically authorize devices is not sufficient as one vendor may sell devices to several operators. Hence the operator must provision a white-list of the devices that are allowed to connect to its network. After successful verification of the device certificate, the CA consults the white-list, ensures that this device is allowed to join the network and only then authorizes the certificate request.

2.3.3 Motivation for using digital certificates

IP Security (IPSec) [50] and Transport Layer Security (TLS) [41] are the most notable protocols used for securing communication in the Internet today [15]. Both protocols support authentication through pre-shared keys (PSKs) as well as digital certificates.

A PSK is a string known by both communicating end-points. The PSK is never actually transmitted over the network; rather it is used to derive the keying material for the session. Configuring secure communication with PSKs is easier than with digital certificates as digital certificates require an associated public key infrastructure. However, digital certificates are considered a stronger authentication method [81] and are also a more scalable authentication solution [7]. In a PSK scheme, the server needs to maintain a mapping between the user and the PSK for every user. This is not required in the digital certificate scheme. Unlike with PSK, a digital certificate's credentials can be revoked by placing them on certificate revocation lists (CRLs). A certificate placed on the revocation list is no longer valid and the owner can no longer authenticate itself with this certificate.

In the rest of this thesis, we consider only digital certificate based schemes for securing communication.

2.4 Southbound protocols

Southbound interfaces enable the separation of control and data planes in SDN networks. They include the communication protocol that forwarding devices and the controller use to interact. Currently, considerable effort is being spent on standardizing these protocols to promote openness and interoperability [69]. OpenFlow [83] has emerged as the predominant southbound

protocol. In addition to OpenFlow, several existing protocols such as PCEP [109], BGP [92] and NETCONF [46] can, with suitable modifications or extensions, also be used as southbound protocols. It is worth mentioning here that networks are likely to support several protocols simultaneously.

In this section, we introduce some southbound protocols. We do not provide an exhaustive list and limit the discussion to protocols well suited to operator networks. Section 3.5 discusses the use cases and deployment models of these protocols.

2.4.1 OpenFlow

OpenFlow was first proposed by McKeown et al. [83] in 2008 and has since then gained industry-wide importance. Several versions of the OpenFlow specifications have been released, the latest version being 1.5.0 released in January, 2015. New versions of OpenFlow add new features to improve the protocol but the inherent methods to program the forwarding devices remain more or less the same.

An OpenFlow switch performs packet forwarding by consulting its *flow table* to determine the output port on which to send the packet. Each entry in the flow table (called a flow rule or flow entry) consists of the packet header fields to match, the actions to apply on matching packets, and the corresponding counters to update. When a switch receives a packet which cannot be matched to any installed flow rule the switch typically first buffers the packet and then requests a new flow rule from the controller with an OFPT_PACKET_IN message. This message includes the packet's header fields. The controller then responds with an OFPT_FLOW_MOD message, which contains a rule for handling the packet and the duration for which to keep the flow rule in its flow table. This duration is called a timeout. Each flow rule has two associated timeout values, an idle timeout value (or soft timeout), which is triggered when the flow remains inactive, and a hard timeout, which is triggered regardless of the flow's activity. When either of these timers expires, the switch removes the corresponding flow entry from its flow table and sends an OFPT_FLOW_REMOVED message to the controller. This mechanism of flow rule installation is *reactive* (depicted in Fig. 2.6) meaning that rules are requested by the switch only upon receiving data packets. Controllers may also behave *proactively* by installing rules to handle expected data traffic [25] along the path of a flow when the packet is seen at the ingress switch.

As mentioned above, when a switch first receives a packet belonging to a new flow, it buffers the packet before sending an OFPT_PACKET_IN message to the controller. This message includes a maximum of 128 bytes of the

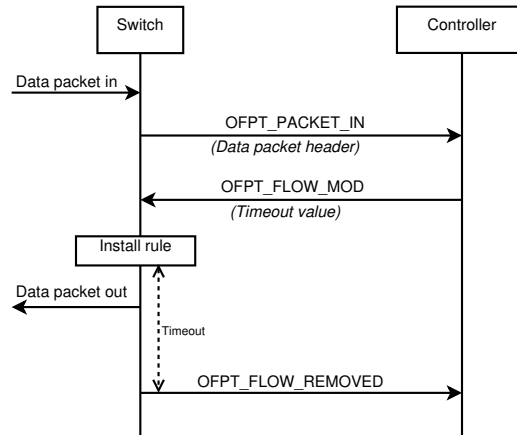


Figure 2.6: Reactive flow installation in OpenFlow

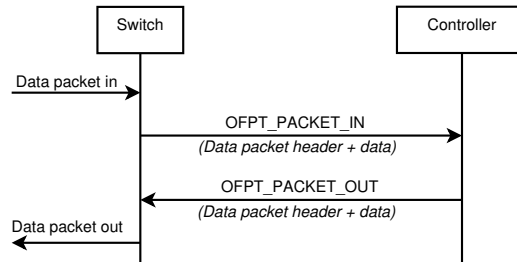


Figure 2.7: Behavior when switch cannot buffer packets

received packet header. However, if the switch does not have buffering capabilities or if the input buffer is full, the OpenFlow specification [5] mandates the switch to send the entire packet to the controller encapsulated within the `OFPT_PACKET_IN` message. In this case, the controller responds with an `OFPT_PACKET_OUT` message which also includes the entire data packet. No flow rule is installed and the switch simply performs the associated action (which is typically to forward the packet out on the specified port). This scenario is depicted in Fig. 2.7.

OpenFlow also provides additional methods such as those used to exchange capability information (e.g. supported version of the protocol), port status information (e.g. when a port comes up or goes down), and error messages (e.g. when a flow rule addition fails).

2.4.2 Path computation element communication protocol (PCEP)

PCEP [109] was designed to enable simpler and more efficient MPLS and GMPLS path computation in large, multi-domain networks. The main idea is to decouple path computation from network devices, and move the functionality to dedicated entities with communication occurring over a standardized protocol. The network devices are referred to as Path Computation Clients (PCCs), path-decisions are made by the Path Computation Element (PCE), and the communication protocol is called PCEP. Fig. 2.8 shows the normal call-flow between the PCC and the PCE. The diagram depicts a passive PCE, wherein a path is computed based on a received request. The PCC sends to the PCE the Label Switched Path (LSP) path parameters which may include the source IP address, the destination IP address, and the required bandwidth. The PCE either replies positively and provides the MPLS label to be used for the path, or it can reply negatively, for instance, if the requested bandwidth is unavailable. PCEs may also behave actively by installing paths dynamically when network changes are detected. Furthermore, PCEs can be stateless or stateful. Stateless PCEs do not maintain a database of allocated resources and hence may create suboptimal paths.

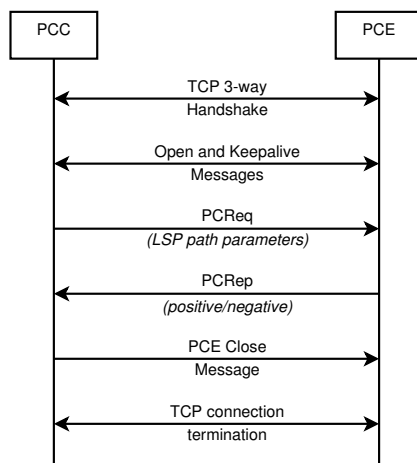


Figure 2.8: A PCEP session

2.4.3 Interface to routing system (I2RS)

Routers build their Routing Information Base (RIB) by participating in interior and exterior gateway routing protocols. This database is built solely

from control-channel communication with other nodes. Based on the learned topology, routers populate their Forwarding Information Base (FIB), which the data-plane references to perform packet-matching and forwarding. As discussed in Section 2.4.1, OpenFlow allows programming the FIB. I2RS [1], on the other hand, is an effort to standardize interfaces and data models to allow programmability of the RIB. The proposed architecture includes I2RS-Agents, I2RS-Clients and I2RS-Services [18] as depicted in Fig. 2.9. Agents run on network devices and provide services such as accessing the RIB manager and topology database. Applications access services through I2RS-Clients. The protocol for communication between the client and the agent is the I2RS protocol. I2RS is work in progress and there are no official standards proposed yet. However, the IETF’s I2RS working group is driving an effort to standardize a data model, which can be used to define an interface to the network device’s RIB [20].

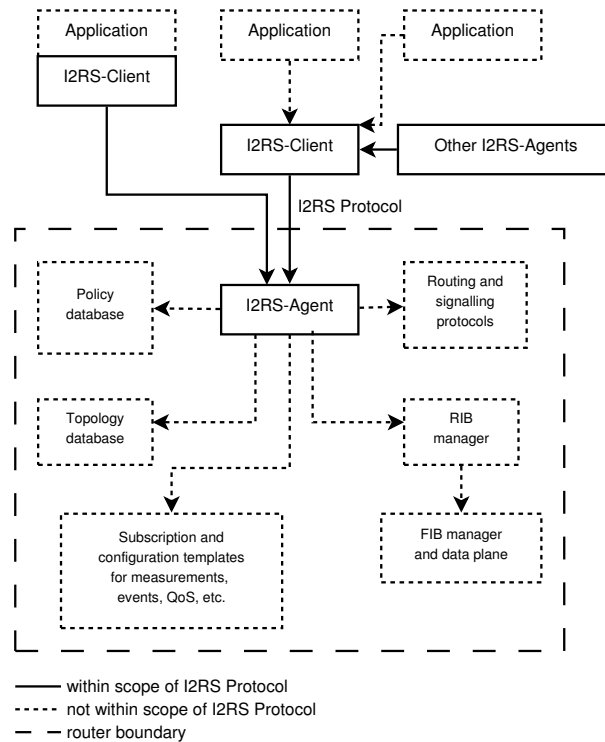


Figure 2.9: I2RS architecture [19]

2.4.4 BGP flow-spec

BGP flow-spec [80] is an extension to the Border Gateway Protocol (BGP) to enable dissemination of traffic flow-specifications (flow-spec) to BGP peers. RFC 5575 defines a new BGP Network Layer Reachability Information (NLRI) address family. This NLRI allows encoding flow-specifications using the multi-protocol extensions for BGP [21]. A flow-specification rule is like an access control list (ACL) rule and consists of the criteria to match aggregated traffic flows based on elements such as the IP destination prefix, IP source prefix, and layer-4 port numbers. The actions to apply to matched flows include accept, discard, rate-limit and redirect and they are specified in the extended communities attribute [58] of the BGP message. In the context of SDN, the controller may act as the flow-specification originator and update policies dynamically on the network devices. The network devices can then propagate this information to other BGP peers within the operator's network or to another operator's network depending on the applied configuration.

2.4.5 Simple network management protocol (SNMP)

SNMP was first conceived in 1988 and has since then undergone several revisions, the latest being SNMPv3 [32]. The SNMP architecture follows a manager-agent model. The manager resides on the Network Management System (NMS) and issues SET or GET requests to an agent running on the managed device. The agent interacts with the device database, known as the Management Information Base (MIB), and carries out the desired task. Additionally, agents can also notify the manager on occurrence of certain events through messages called SNMP traps.

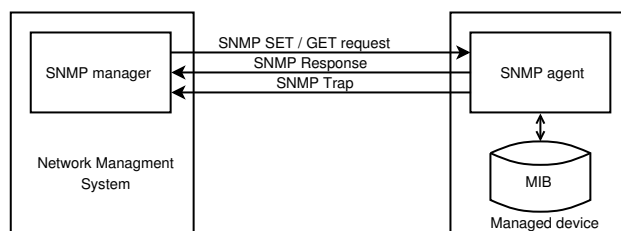


Figure 2.10: SNMP

SNMP is a widely recognized protocol and is implemented extensively across all sorts of network devices. Although SNMP is a good candidate for a southbound interface, it has several practical shortcomings. These are discussed in [99] and include the following:

- SNMP performance of reading large amounts of data (such as router’s routing table) is very poor.
- Network administrators view configuration of devices as a task sequence. However SNMP’s view is data-centric wherein configuration implies changing state of data objects. Hence, there is a need for additional translation code on the management application.
- Writeable SNMP objects are not widely deployed. Furthermore, standard MIB modules do not always provide the writeable objects that would be needed for SDN. Many such objects are contained only in proprietary modules. Therefore SNMP cannot easily be used to program forwarding rules on the network devices.

2.4.6 NETCONF

NETCONF was developed by the IETF to address the shortcomings of SNMP. Like SNMP, it is a network management protocol and “provides mechanisms to install, manipulate, and delete the configuration of network devices” [46]. The NETCONF protocol is divided into four layers as depicted in Fig. 2.11. Messages and configuration data exchanged between the NMS (or client) and the network device (or server) are encoded using Extensible Markup Language (XML) [28]. The protocol operations are carried out as Remote Procedure Calls (RPCs).

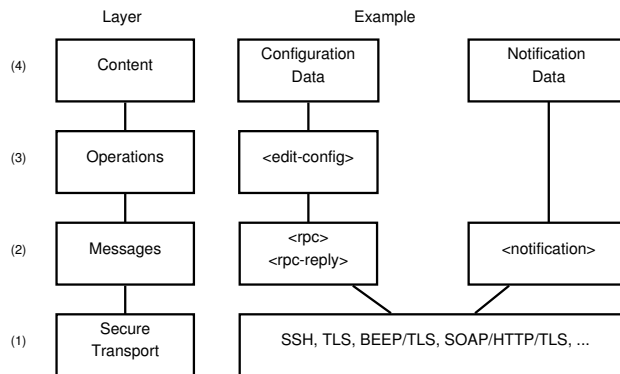


Figure 2.11: NETCONF protocol layers [46]

- (1) The Secure Transport layer provides a communication path between the client and the server. NETCONF requires this connection to be persistent, connection-oriented and provide authentication, data integrity,

confidentiality, and replay protection. NETCONF implementations are required to support the SSH protocol.

- (2) The Message layer provides mechanisms to encode the RPCs and notifications in a transport-independent manner.
- (3) The Operation layer specifies a set of operations to manage devices and retrieve state information. Some examples include get, get-config, edit-config, lock, and close-session. The operations may be extended depending on the device's capabilities.
- (4) The Content layer is needed to model the configuration and state data of the devices. NETCONF itself does not provide such a model. YANG [27] has emerged as the leading data-modelling language to be used in conjunction with NETCONF.

NETCONF is a proven standard for writing network configurations. Its features include domain-specific knowledge, support for transactions, and vendor independence. NETCONF can be used to configure the devices through the edit-config operation. Thus it enables programmability of the forwarding devices. NETCONF is the configuration protocol used by OFCONFIG [4] to program OpenFlow devices.

2.5 Protocols for topology discovery

Topology discovery involves learning how network devices are interconnected. The discovered topology can represent either logical or physical connections. A *logical* view of the topology is constructed from layer-3 information. With this method, layer-2 devices are not discovered [23]. It only determines connections between devices without knowing details of the physical links. On the other hand, a *physical* view of the topology is constructed using layer-2 mechanisms and this method exposes the physical interconnections between devices. Peers in the logical topology are routers that are one hop from each other, whereas peers in the physical topology are directly connected ports [105]. Topology discovery is a crucial component in SDN networks as network applications depend on this information to make routing decisions. Furthermore, applications need to know about the complete network topology in order to make optimal routing decisions.

In this section we introduce some of the protocols used for topology discovery in conventional networks. The same protocols can be used for topology discovery in SDN networks as will be explained in Section 3.4.

2.5.1 Simple network management protocol (SNMP)

SNMP has been presented in Section 2.4.5 as a southbound protocol. It can also be used for the purpose of topology discovery. As previously mentioned, devices with SNMP support store all information in Management Information Bases (MIB).

The information on the devices is keyed by hierarchical Object Identifiers (OIDs). There are both standard and proprietary MIBs. As an example of an OID, the BGP version of a node is represented as *iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).bgp(15).bgpVersion(0)* or simply *1.3.6.1.2.1.15.1.0*. The value returned by an SNMP query with this OID will be 4 if the IPv4 version of the BGP protocol is being used by the queried device.

Other information stored as OIDs includes details regarding the device's ports and the neighbors to which it is connected. Control-plane protocols such as IS-IS [59], OSPF [36, 85] and BGP [92] have standardized MIBs [54, 63, 88]. By querying each device with the SNMP GET method, topology information can be extracted from the devices and then pieced together to form a complete view of the network topology.

2.5.2 Link layer discovery protocol (LLDP)

LLDP¹ was standardized by the IEEE in the standards document 802.1AB [10]. As the name suggests, it is a link-layer (layer-2) neighbor discovery protocol that enables adjacent switches to identify each other and exchange information regarding their capabilities. The protocol was developed to supplant proprietary protocols, such as the Cisco Discovery Protocol and Extreme Discovery Protocol, and hence it was designed to work in a multi-vendor environment.

LLDP works in two modes, the transmit mode and receive mode. In the transmit mode, a switch periodically sends out Ethernet frames called LLDP Data Units (LLDPDU) from each of its interfaces. Each LLDPDU contains the switch identifier and the identifier of the port from which the frame is sent. The destination MAC is set to a special multicast address. Upon receiving an LLDPDU (in the receive mode), a switch parses the LLDPDU and stores the switch identifier and port identifier in to the Physical Topology Management Information Base (PTOPO-MIB) [26]. Devices are required to consume the LLDPDU, i.e., they should not forward it to any other port [8].

LLDP allows switches to exchange information with their adjacent nodes, and hence they are able to learn only about their immediate neighbors. A

¹LLDP is formally referred to by the IEEE as Station and Media Access Control Connectivity Discovery

system administrator can, via the network management system and SNMP, retrieve a switch's PTOPO-MIB [26]. By querying all the switches in the network and thus gathering each switch's neighbor information, the entire network's *physical* topology can be constructed.

2.5.3 BGP link-state (BGP-LS)

With interior gateway routing protocols (IGP), such as OSPF and IS-IS, the network devices' topology knowledge is limited to the area or domain to which they belong to. Although it is possible to make the SDN controller a passive IGP listener and thereby obtain topology information, this method does not scale well. The controller needs to participate in the IGP at several areas and piece together the gathered information to form the full network topology. BGP-LS [52] has been proposed as a solution to this.

BGP-LS is an extension to BGP enabling it to carry link-state information gathered by the IGPs. A BGP message contains one or more Link-State NLRIs, which are further classified as Node NLRIs, Link NLRIs and Prefix NLRIs. A Node NLRI uniquely identifies the router, a Link NLRI uniquely identifies a link, and a Prefix NLRI uniquely identifies an IPv4 or IPv6 Prefix originated by the BGP speaker. Each domain must have at least one BGP speaker and the IGP information is redistributed to this node. The SDN controller (or another dedicated network node with which the controller interfaces) can act as a BGP route reflector, and thereby learn the entire network's topology. This scenario is depicted in Fig. 2.12.

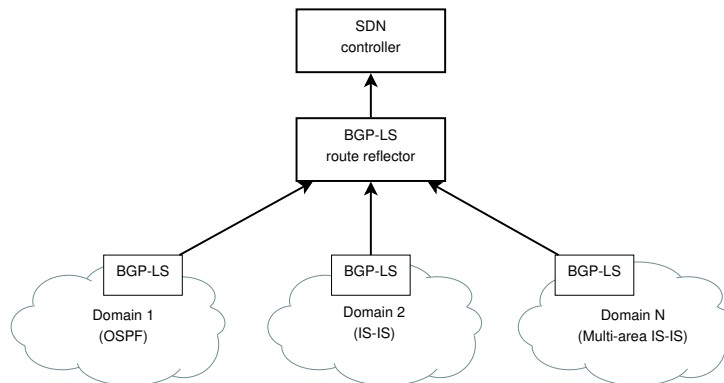


Figure 2.12: BGP-LS – gathering topology information [107]

Chapter 3

SDN deployment solutions

This chapter presents our survey and analysis of deploying SDN in telecommunication networks. To build a new network or to upgrade an existing one, the operator needs to determine the services that it wants to provide, analyse the associated network requirements, and procure the network equipment. The operator then needs to work out a deployment strategy and begin installing the devices in the network. Each installed device must be configured with a management interface and have the appropriate protocols enabled. In addition, the operator must configure the mechanisms and access rights for performing network management tasks such as traffic engineering, network monitoring, and troubleshooting. These jobs are quite diverse and automating them is crucial in order to be able to scale the deployment process and enable ease of management.

In this chapter, we discuss some of these deployment considerations in the context of SDN. We broadly classify them as follows:

- Setting up new hardware – strategies for transitioning to an SDN network
- Bringing up a device – connecting a device to the network
- Bootstrapping security – provisioning security keys and configuration on the device to enable secure control-plane communication
- Performing topology discovery – constructing a view of the network’s physical topology
- Packet forwarding – southbound protocols and network applications for installing packet forwarding rules

3.1 Setting up new hardware

In order to reap the benefits of SDN, network operators first need to install programmable network devices in the network. A straightforward strategy is to simply swap-out existing network devices and replace them with SDN devices. This approach is viable when rolling out a new network or deploying SDN in closed environments such data-centers and campus networks [101]. For example, such a *clean-state* strategy was employed by Google in deploying their inter data-center SDN network [60]. However, the deployment took several years and the benefits were visible only after the entire switching hardware was upgraded [72].

A complete network overhaul is not always a feasible solution. Network operators want to protect their current investments, and budget constraints may restrict purchase of large volumes of new equipment at one time. Hence, another approach is to deploy SDN *incrementally* alongside the existing networks. We call a heterogeneous network with legacy hardware and SDN hardware a *hybrid network*.

Another possibility to deploy SDN incrementally is through a *dual-stack* approach, wherein a network device runs both legacy protocols and SDN protocols in parallel. We call such a device a *hybrid switch*. The implementation of a hybrid switch can be twofold. In the basic hybrid switch, packets tagged with certain VLAN tag numbers are processed by SDN methods whereas other packets are handled by traditional methods. A variant of the hybrid switch method, which is gaining popularity due to its inclusion in the OpenFlow 1.3 specification [5], is one where the forwarding pipeline supports both traditional processing and SDN processing. In such an implementation, the controller maintains control over all the switch ports and VLANs. By inserting appropriate rules in the forwarding table, the controller delegates control decisions to the switch itself. For instance, the controller may install a rule to forward all DNS requests to it for inspection but instruct the switch to handle all other traffic in the traditional way. In this manner, the controller (and network applications) can be light-weight modules that know only about DNS and not any other kind of traffic. Furthermore, not all traffic needs to be forwarded to the controller, thereby saving computing resources and network bandwidth. The main advantage of the dual-stack deployment is that this method allows for an overnight change in the forwarding hardware. When a hybrid switch is installed in the network, the controller instructs it to handle all traffic in the traditional way. Thus, it leads to a seamless transition to an SDN-ready network where SDN functionality can slowly be switched on. Fig. 3.1 depicts the three deployment strategies discussed in this section.

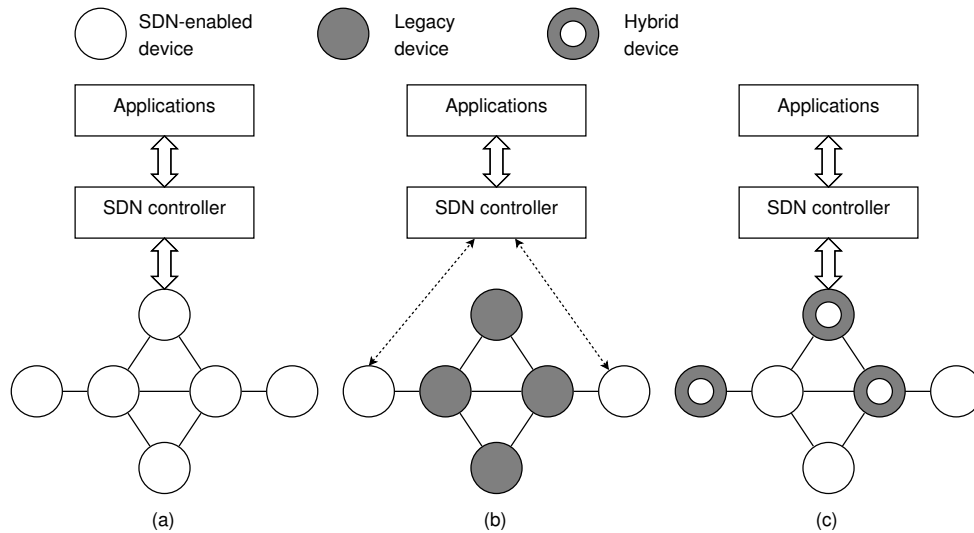


Figure 3.1: SDN deployment models (a) clean-slate (b) hybrid network (c) dual-stack

3.2 Bringing up a device

Bringing up a network device involves providing IP connectivity to the device and enabling remote management of the device. In this phase of deployment, the device behaves as an *end-host* rather than as a forwarding device, i.e., the device cannot forward any packets yet. We assume the device supports the normal networking stack and protocols that are supported by end-hosts; for example the Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS). The initial configuration operation can be performed manually using the Command Line Interface (CLI) or in an automated manner. In this section, we discuss the configuration parameters and associated network infrastructure required to connect a device to the network.

3.2.1 Enabling IP connectivity

When manually bringing up the device, an engineer present at the installation site performs the initial configuration tasks. The engineer connects one end of a serial cable to a laptop and the other end to the console port of the device and starts a terminal emulation software on the laptop. Now the device is ready to be configured through vendor-specific CLI instructions. The first step is to enable IP connectivity by configuring one or more interfaces with IP addresses. A layer-2 switch has only a virtual interface associated with an IP address. A layer-3 router has an IP address configured on each of its

physical interfaces in addition to one or more virtual interfaces. Furthermore, the address of the default gateway router, DNS server, network time protocol server, and the device host name are also configured on the device.

Alternatively, the device can obtain some or all of the mentioned configuration parameters via DHCP [45] and BOOTP vendor extensions [14]. The process of automatically enabling IP connectivity is depicted in the form of a flow chart in Fig. 3.2. The operator must set up the DHCP server before the device starts the DHCP protocol. A reservation (mapping of DHCP parameters to a specific device) should be added on the DHCP server based on either the MAC address of the interface or device client identifier which will be included in the DHCP request. This is required so that the DHCP server can allocate to the device the correct host name and the Trivial File Transfer Protocol (TFTP) server address from which to fetch subsequent configuration. In case the host name is not learned via DHCP, the device performs a reverse DNS lookup on the IP address it received via DHCP and thus learns its host name. This means that the DNS server must also be set up in advance and should be reachable from the device. Alternatively, there may be a file on the TFTP server containing a mapping of the host name and IP address. The device downloads this file and chooses the correct host name. For the device and DHCP server to communicate over DHCP, the DHCP server should reside on the same LAN as the device. If this is not the case, there should be a DHCP relay agent which forwards requests and replies between the device and the DHCP server.

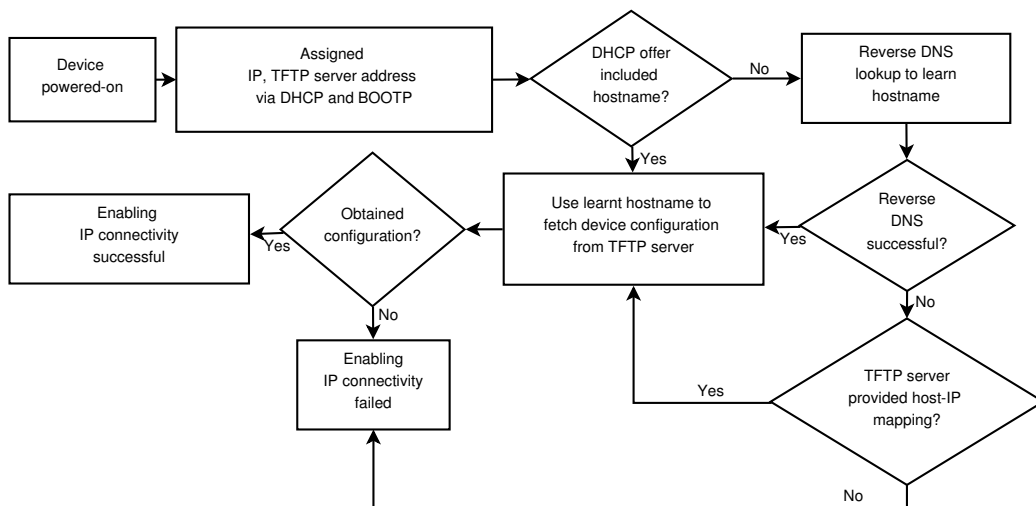


Figure 3.2: Enabling IP connectivity for a network device

3.2.2 Enabling Remote Management

For remote management of the device, Telnet and/or SSH need to be enabled. This may be configured manually or may be part of the initial configuration file downloaded from the TFTP server. Telnet can be used when connecting to the device from within an isolated secure environment. However, Telnet offers no encryption or server authentication. Also, when a Telnet connection is made, the user name and password are sent in clear text over the network. Since the device does not know what kind of environment it is in, Telnet should be disabled by default. SSH is the most commonly used secure remote login protocol and provides public-key based authentication as well as password-based authentication. Password-based authentication requires the allowed user name and password to be provisioned on the device, for example, as a part of the initial configuration file. However, as discussed in Section 2.3.3, this method is not as secure as the public-key authentication methods and does not scale well. For public-key authentication, the device needs to generate a key pair. This can be triggered manually through the CLI, or the device itself can generate keys at the first boot up. The keys, once created, are stored in the device's non-volatile memory.

3.3 Bootstrapping security

The most popular secure communication protocols, namely TLS/SSL and IPSec, use digital certificates for authentication. In this section we discuss mechanisms for:

- (a) installing the operator root certificate
- (b) provisioning a domain certificate signed by the operator CA

Please refer to Section 2.3 for details on the PKI actors and certificate profiles.

3.3.1 Installing the operator root certificate

Each network device needs the operator root certificate in order to verify the domain certificates presented by the SDN controller and other devices to which it connects. It also uses the operator root certificate to verify the authenticity of its own domain certificate during enrollment. The operator root certificate can be installed on the device through the CLI or by installing it from a USB disk. This method is secure as long as only authorized individuals have access to the CLI of the device. Alternatively, the device itself

can download the required certificate directly from the CA or another public certificate repository. The device needs to be provided with the URL from where to download the certificate. However, manual intervention is required to verify the authenticity of the certificate. The device or the device administrator must compare the fingerprint of the received certificate with an authentic fingerprint received by out-of-band methods. This fingerprint can also be pre-configured on the device, in which case the interactive verification of the certificate is no longer required. Fig. 3.3 depicts the steps to install the operator root certificate.

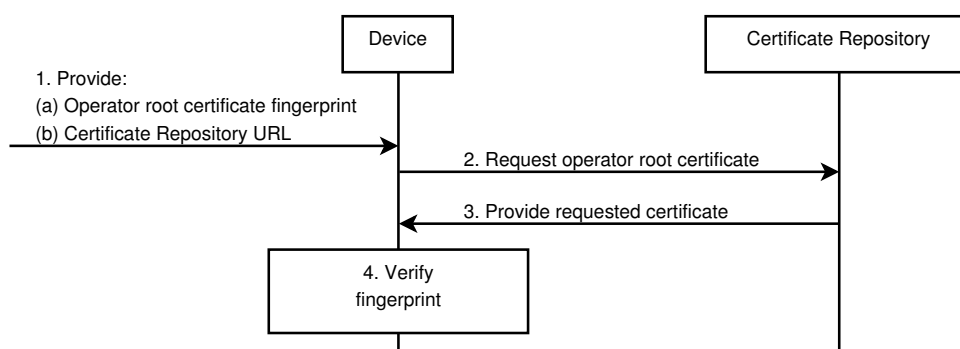


Figure 3.3: Installing the operator root certificate

3.3.2 Provisioning a domain certificate

The device can register with the network through a manual process or in an automated manner. The basic steps (depicted in Fig. 3.4) involved are as follows:

1. Device creates a certificate signing request (CSR) and sends it to the operator CA
2. Operator CA authorizes the request
3. Operator CA creates a signed domain certificate and makes it available to the device

We first assume that the device has had no prior contact with the PKI system and owns no digital certificates, i.e., it does not possess even a device certificate. Additionally, we assume that the device has a key pair which it wishes to have certified by the CA. In the manual mode, an engineer generates a certificate signing request by entering CLI commands. The engineer then transfers this request to the CA by any appropriate method, for example,

by email, in a file, or with a copy-paste operation from the user interface. The engineer must ensure that the request is transferred to the CA without possibility of malicious modification and then manually authorize the CA to issue the certificate. If the CA accepts the request and generates a certificate, the engineer again transfers and installs the certificate onto the device.

In an automated method, the device uses certificate enrollment protocols such as SCEP or CMP to obtain the digital certificate. In order to authorize the request, the CA generates a password and creates a binding between the device and the password. This operation is manually performed and the password is configured on the device using out-of-band methods. The certificate request is authenticated using this shared secret and hence the CA can automatically authorize the request. There is no need for the device to authenticate the CA as that can be done once the digital certificate has been received. The device can verify if the expected CA created the certificate by verifying the signature on the issued certificate.

When the device possesses a digital certificate signed by a third-party CA (such as the vendor CA), authorization can be simplified (refer to section 2.3.2). The operator CA can authorize requests based on the device certificate presented by the device. However, one vendor may sell network devices to several operators. So the operator needs to configure a white-list with the serial number of the devices that are allowed to join the network. Upon receiving a request, the CA checks this list for the presence of the serial number and then proceeds to authenticate the device. In this case, the CA needs to have a list of the trusted vendor root certificates in order to verify the device's identity.

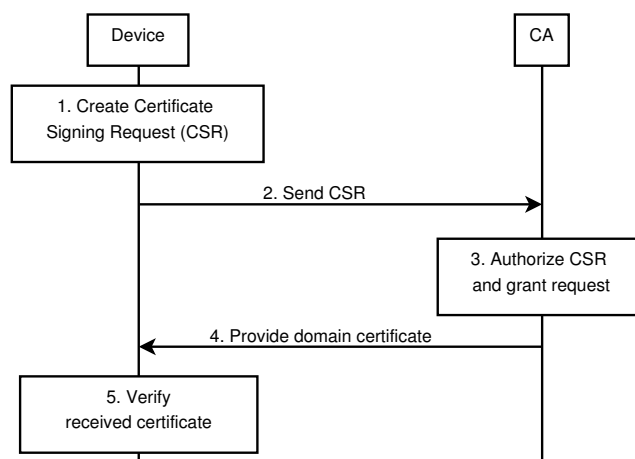


Figure 3.4: Provisioning a domain certificate

3.3.3 Bringing up a remote device outside the secure operator network

A network device, for example one that is located in the access network, may need to connect to the servers (e.g. the SDN controller) located in the secure operator network over an insecure public Internet. In order to bring up the device from such an insecure environment, all control-plane communication with the device must be protected. IPSec is a suitable protocol for such an environment as it enables encryption at the IP layer and thereby all traffic, regardless of the transport protocol or service, can be protected. In this case, the device first obtains an IP address via DHCP and also learns the IP address of the IPSec gateway for the trusted domain. Then the device performs certificate enrollment with the operator CA as discussed in the section above and obtains the domain certificates for IPSec authentication. It can then form an IPSec tunnel through the untrusted network and connect securely to the other servers, for instance the TFTP server. If the operator CA resides within the secure operator network, appropriate firewall rules and policies must be configured on the IPSec gateway to allow traffic between the device and the CA before the device has formed an IPSec security association.

3.3.4 Enhanced security with logging servers

As discussed earlier, the DHCP server or relay must be located on the same LAN as the new device. Thus, if the new device is brought up in an untrusted network, the DHCP server or relay will also be located in the untrusted network. In this case, the device can be hijacked with a rogue DHCP server so that its deployment to the operator network fails. Hence, the network operator needs a mechanism to detect the successful or failed integration of the device into the operator domain. This can be achieved through the use of preconfigured event logging. We describe our assumptions and the functionality below.

1. The device allows only one domain certificate, i.e., the domain certificate identifies the operator network to which the device is connected. We assume here that the device is controlled only by one network operator at a time.
2. The network equipment vendor knows which devices are sold to which network operator.
3. The vendor provides a service via which operators can login to their

accounts and view the list of devices. In addition to providing the list of devices, the service also stores logs received from the device.

4. The device has fixed pre-configured fully qualified domain names for the logging servers.
5. The operator configures its DNS servers to resolve the IP addresses of these logging servers and its firewall to allow network devices to access them.

Once the device obtains IP connectivity, it periodically sends the logging servers its certificate details and the IP addresses of the SDN controllers to which the device is connected. By manually observing logs from the new device, the network operator can check whether the device has connected to its domain as expected. The operator needs to verify the device's identity, the certificate issuer's name and the details of the SDN controllers. If any of the information is wrong, the operator can reset the device and start afresh. On the other hand, if no events are seen on the logging server, it means that the device may have obtained false configuration information from DHCP and has been hijacked. Again, the device should be reset and the process started afresh. These scenarios are shown in Fig. 3.5.

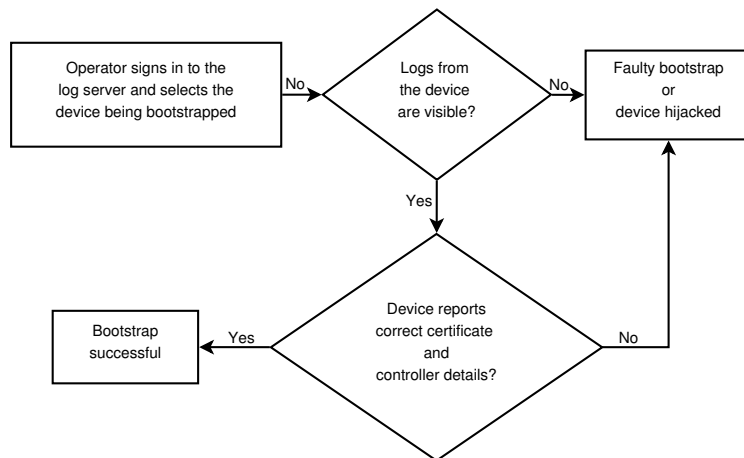


Figure 3.5: Detecting bootstrap result from log server

We propose that there are at least two logging servers – one vendor controlled and the other operator controlled. The vendor's logging service may provide only a minimal service; for example the operator can list devices and view logs from the device. This allows the operator to rely on the vendor provided service initially, before the device has been configured to use the operator's own logging service. Additionally, the vendor logging server can

make use of secure TLS connections through the use of device certificates. If the operator does not wish to expose device information to the vendor, the vendor's logging service need not be resolved on the DNS server. Having an operator-controlled logging server allows the operator to create monitoring applications on top of the service. For instance, the logging server could send email notifications or SNMP traps if the state of the device changes.

3.4 Topology discovery

Discovering the complete *physical* network topology is crucial in SDN networks as SDN applications require this information to make optimal routing decisions. However, as learned in Section 3.1, operators are likely to employ an incremental deployment strategy in which legacy and SDN devices need to interoperate. This section considers how the full topology of such networks can be learned. In order to simplify the discussion, we first start with topology discovery mechanisms in pure SDN networks followed with those used in pure legacy networks. We then expand our discussion to the case of hybrid SDN networks.

3.4.1 Topology discovery in pure SDN networks

OpenFlow messages combined with the Link Layer Discovery Protocol (LLDP) can be used to construct the physical topology of pure SDN networks. We assume here that all the devices in the network speak the OpenFlow protocol. To explain how discovery works we consider the simple topology as illustrated in Fig 3.6.

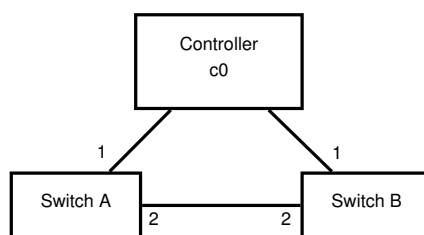


Figure 3.6: Simple topology to explain how link discovery works

Upon initial connection establishment with each switch, the controller installs a flow rule to forward LLDP packets to it. It then crafts an LLDP packet containing its own controller identifier (c0) and the device identifier of the switch to which the packet is destined. Let us assume the packet is sent to switch A. The controller encapsulates the LLDP packet into a

PACKET_OUT message and instructs switch A to flood the packet. Switch A then sends the LLDP packet out on all its data-plane ports, in this case only port 2. When switch B receives this packet, a rule match for LLDP is found. Switch B encapsulates the received packet into a PACKET_IN message and sends it to the controller. The PACKET_IN message includes the number of the port on which the LLDP packet was received (port 2 in this case).

The controller parses the packet and determines that it had initially sent this packet to switch A (based on the switch identifier). Hence it learns that port 2 of switch B is connected to some port of switch A. It repeats this process by sending an LLDP packet to switch B and when the same LLDP packet is received in the PACKET_IN message from Switch A, the controller can learn that port 2 of switch A is connected to switch B. By combining these two pieces of information, the controller now determines that port 2 of switch A is connected to port 2 of switch B.

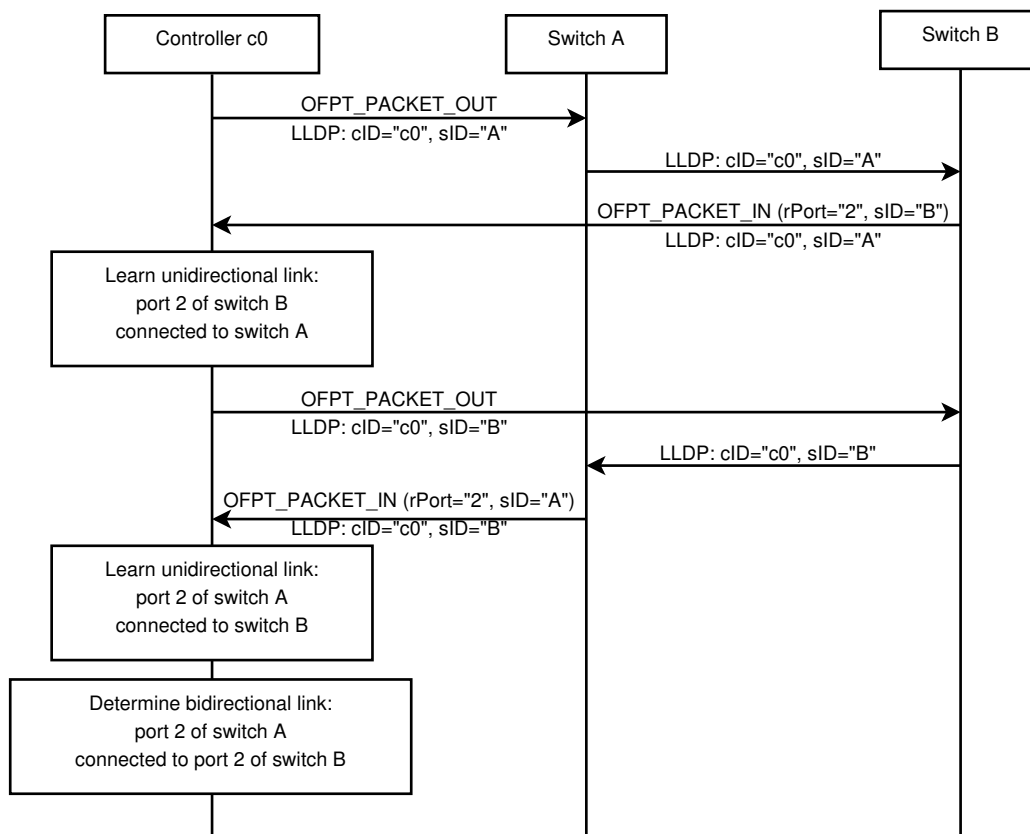


Figure 3.7: Topology discovery using OpenFlow and LLDP

The above described method of topology discovery is depicted in Fig.

3.7. For a more detailed discussion, please refer to the work presented by Sharma et al. [104] where they use the DHCP, ARP and LLDP protocols with OpenFlow to bootstrap a pure SDN network. It is worth noting that the OpenFlow standard [5] does not propose any topology discovery mechanism or protocol but a white paper by the Open Networking Foundation [12] cites the above mechanism.

3.4.2 Topology discovery in pure legacy networks

Legacy networks do not require knowledge of the full layer-2 topology in order to forward packets. Legacy devices employ MAC learning and also use control-plane protocols to build their topology database. MAC learning is a process where switches examine received packets and save a mapping of the source MAC address and the input port in their FIB. In this way, they learn the direction, i.e. the port, via which the MAC address can be reached. However, due to the local nature of MAC learning and the distributed nature of control-plane protocols, the database on each switch represents only a small subset of the entire network topology.

Building a global view of the layer-2 topology has been under active research for several years. The most common method is to use SNMP to query every network device individually and then piece together the complete topology information [23, 29, 78, 87, 97]. One way is to retrieve entries in the forwarding information databases (FIB) of all the devices and construct a view of the topology based on the common entries in the forwarding tables. Additionally, the most commonly used control-plane protocols provide their own MIBs (refer to Section 2.5.1.) Information regarding neighbors and link states can also be obtained from these MIBs.

3.4.3 Topology discovery in hybrid SDN networks

Neither of the schemes presented thus far can be used alone to construct a view of the full network topology of a hybrid SDN network; instead, both schemes need to be jointly used. Let us consider the topology depicted in Fig. 3.8, wherein two SDN domains are separated by a legacy domain. In this network, each SDN domain has its own controller. The topology of each of the domains (referred to as Local TopoDB) can be discovered using the mechanisms described above. A network application can then query each of the local databases and construct a global topology.

However, this does not address discovering the links between the domains. If the legacy domain uses LLDP for link discovery, the PTOPO-MIB of the legacy device will contain information about the SDN device to which it is

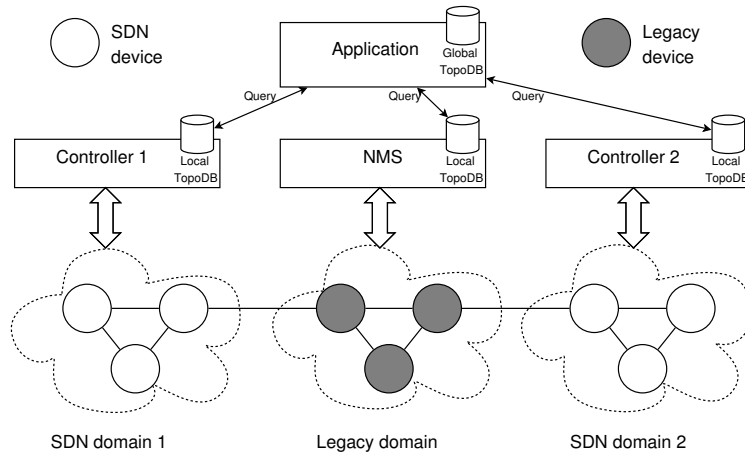


Figure 3.8: Topology discovery in hybrid SDN network

connected. Furthermore, when the legacy device sends an LLDP packet to the SDN device, the controller can determine that the device is connected to a legacy domain as it does not find a controller identifier in the LLDP packet. The network application can use these bits of information to correctly construct the topology information.

If the legacy domain does not use LLDP but some other protocol, for instance IS-IS, then it may not be possible to accurately determine the exact network topology. In the worst case, all adjacent legacy switches appear as one switch. However, it is still viable to determine the devices (and ports) through which the SDN domains are connected to each other. This is achieved through the use of broadcast or other multicast MAC addresses in the LLDP packet crafted by the controller instead of the MAC address specified by the LLDP standard. In this case, legacy switches should honour the request for broadcast and the LLDP packet will eventually reach the other SDN domain. This way, the controller can detect connections between the SDN islands. The Floodlight controller utilises this mechanism and refers to the protocol as the broadcast domain discovery protocol [49]. However, it may not always be possible to determine which legacy device (and port) the SDN domains are connected to. If the legacy devices send any control-plane information which consists of the device identifier and the controller is able to parse this message, then it could be possible to construct a view of the full topology.

3.4.4 Limitations of discussed protocols

The discussed methods of building a picture of the physical topology may not always work in practical environments. Schafer et al. [97] discuss their experience of using LLDP and SNMP. They highlight the following problems:

- Not all network devices support LLDP.
- A switch that is not 802.1D compliant, upon receiving an LLDPDU, forwards the frame through all its ports. Hence, switches connected to this switch falsely believe that they are directly connected to the switches they are actually not.
- The LLDP specification does not mandate the support for SNMP. Hence, although most vendors support SNMP, this is not guaranteed.
- Different network management systems may interpret the SNMP values differently. For instance the network management system used by the authors only supported the mandatory TLVs and was unable to parse the optional TLVs. Furthermore, one device reported wrong port identities.

3.5 Packet forwarding

Once the network topology is known, applications make the routing decisions in hybrid networks and communicate them to the controller. The controller then translates these decisions into forwarding rules and configures the network devices, i.e. the SDN switches, via southbound protocols. The switches can now match packet headers against these rules and forward data-plane traffic correctly.

In a hybrid SDN network, SDN devices and legacy devices need to interoperate. This section discusses three different approaches that have been proposed for enabling the packet forwarding functionality in hybrid networks. These approaches differ considerably depending on the network section and the legacy protocols that will be used alongside the new SDN protocols.

3.5.1 Forwarding in aggregation and core networks

Aggregation and core networks typically forward packets based on MPLS labels. OpenFlow is a good southbound protocol candidate in these networks as OpenFlow switches can match packets based on MPLS labels, and can also push, pop, and swap the MPLS labels.

If all the switches in the MPLS network speak the OpenFlow protocol, the controller has two options to configure the bindings between forwarding equivalence classes and MPLS labels on the routers. The first method involves a reactive controller. The controller provides switches with MPLS labels and the output port on which to forward the packet hop-by-hop. In other words, each switch forwards the first packet of a new flow to the controller and the controller responds with the forwarding rule for that flow. Alternatively, the controller may be proactive. In this case, when the ingress switch requests an MPLS label for a new flow, the controller calculates the entire label-switched path and configures all the switches on the calculated path right away. This method reduces the number of control messages received by the controller as only the ingress switch needs to send the first packet of a new flow to the controller.

Operators can harness the benefits of SDN without upgrading their entire MPLS network by deploying SDN switches only at the access edge [31]. They can enforce Quality of Service (QoS) requirements and other access control policies at the point user traffic enters the operator network [79]. However, if only the edge nodes support OpenFlow and the other nodes are legacy MPLS routers, network applications become more complex. They need to implement the MPLS protocols for label distribution and also assist legacy switches with building their topology databases.

The work presented in [103] demonstrates MPLS in a pure OpenFlow network whereas [64] demonstrates MPLS in a hybrid network.

3.5.2 Unified control of transport and IP networks

Traditionally, the IP network and the transport network have been managed independently. With SDN, there is an opportunity to centralize the control of both these networks [38]. Generalized Multiprotocol Label Switching (GMPLS) is the de facto protocol used in wavelength switched optical to set up and release lightpaths, and PCEP enables centralized control of these networks. In order to provide unified control of transport and IP networks, several works have proposed the integration of the OpenFlow and PCEP protocols [33, 34, 74, 82] while others have suggested extensions to the OpenFlow protocol in order to provide the unified control [39, 75–77]. However, integrating OpenFlow and PCEP may be beneficial to operators as they can utilize existing path calculation algorithms developed for PCEP [86] and therefore do not need to develop new software modules for OpenFlow specifically.

We now discuss three schemes by which OpenFlow and PCEP can be integrated [34, 86]. The first scheme represented in Fig. 3.9(a) is perhaps the most straightforward one. The PCE exists as an external entity and

the OpenFlow controller behaves as the PCC. The controller requests a path with the PCEP protocol and communicates the calculated path back to the switches with the OpenFlow protocol. In the second scheme (Fig. 3.9(b)), the PCE runs as an application on top of the OpenFlow controller. There are two independent instances of the link state database (LSPDB) and the topology engineering database (TED) in both of these cases. Mechanisms to allow the PCE to query the databases and synchronize them need to be developed. The last scheme, depicted in Fig. 3.9(c), represents an integrated PCE and OpenFlow controller. In this case, the controllers share the same LSPDB and TED instances with a shared path computation application calculating the required path. A full integration allows the application to maintain common data structures and state, and thereby simplifies concurrent access.

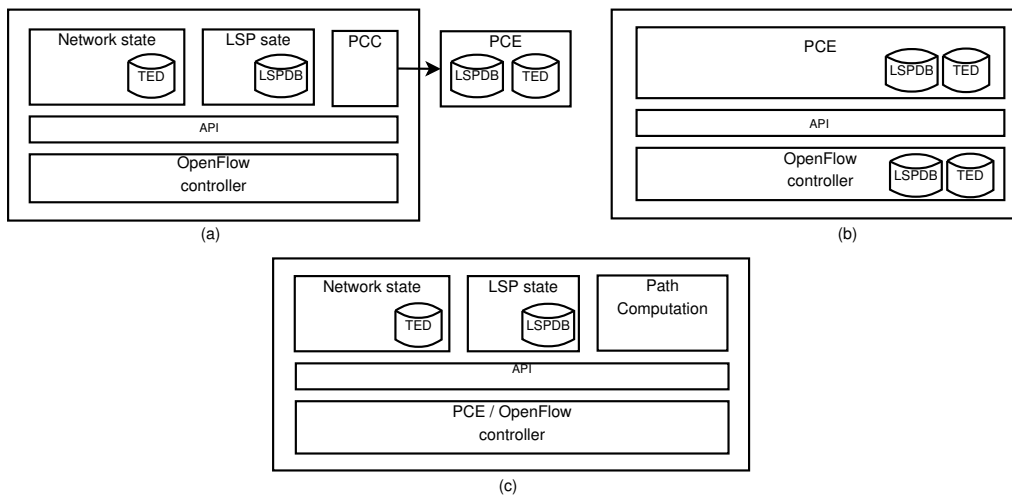


Figure 3.9: (a) Application, (b) external, and (c) integrated PCEP in SDN [86]

3.5.3 Centralized inter-domain routing

BGP [92] is the de facto protocol for inter-domain routing. BGP comprises of two protocols: external BGP (eBGP) and internal BGP. eBGP is used for peering with external autonomous systems (AS) whereas iBGP is used to distribute the routes learned via eBGP within the AS. Centralizing BGP control can improve routing convergence time [51] and avoid forwarding loops and path oscillations [47] in addition to providing typical SDN advantages such as simplified architecture and easier management.

Feamster et al. [47] describes three phases for the evolution to a BGP-free edge as represented in Fig. 3.10. Their routing control platform can be compared to an SDN controller. In the first phase, the controller forms iBGP

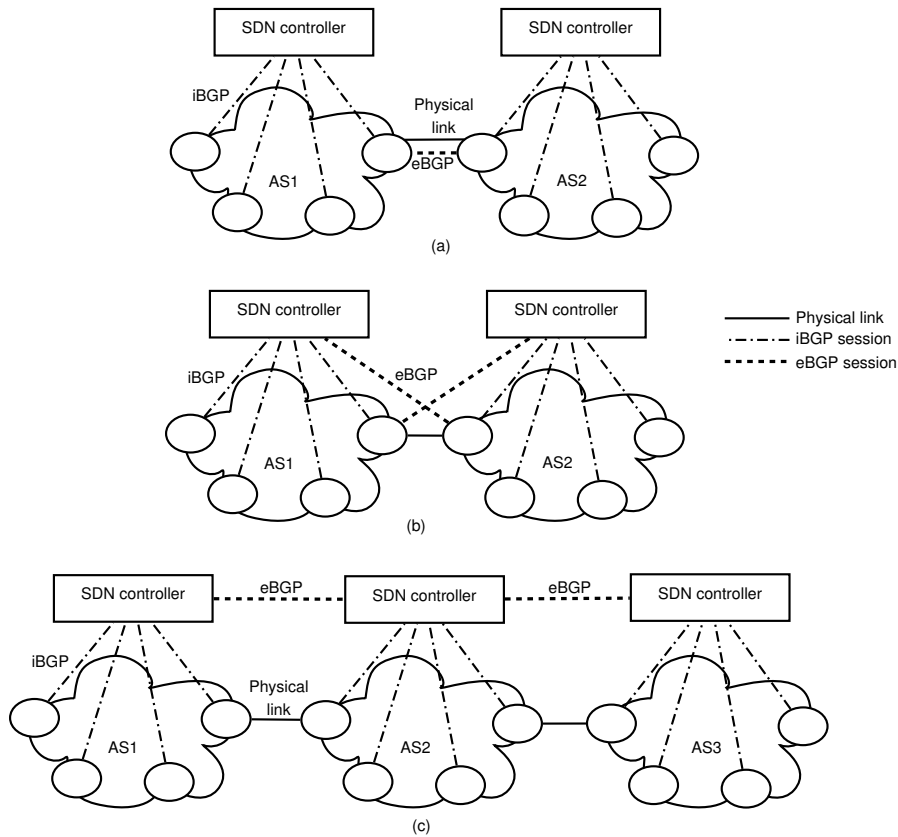


Figure 3.10: Evolution of centralized BGP control (a) phase 1 (b) phase 2 (c) phase 3 [47]

sessions with all the border routers within the AS. It learns eBGP routes from each border router within the AS over the iBGP interface and also distributes the routes to the other BGP peers in the same AS. This architecture is similar to that of iBGP with route reflector but differs in the sense that the SDN controller can choose the best router for each of the BGP peers individually. In the second phase, in addition to performing its role as a route reflector, the controller also interfaces with external ASes forming eBGP sessions. Now the controller does not need to learn routes over iBGP sessions but only distributes the routes that it has learned directly from external ASes. Lastly, in the third phase, multiple domains exchange inter-domain routing information through their SDN controllers. The controllers could use eBGP or some other protocol for communication between themselves.

The first phase of deployment has been experimentally demonstrated in [73] and the second phase in [94]. Both these works replace the border nodes with OpenFlow switches and use the OpenFlow protocol instead of iBGP

for communication with the controller. They also develop BGP applications which implement the logic to form eBGP sessions with external domains, perform the necessary route computations, and configure the switches with forwarding rules. [68] presents an implementation proposal to realize phase 3 of deployment using OpenFlow.

3.6 Summary

This chapter has reviewed the key aspects in deploying SDN devices into the operator network. An incremental deployment of SDN hardware is more feasible than a clean-slate approach. Operators are likely to deploy network devices that support both the SDN protocols and the legacy protocols. This approach allows operators to migrate to an SDN-ready network and switch on SDN functionality gradually.

We have presented an automated process by which newly deployed forwarding devices can be bootstrapped into the network. The proposed process uses DHCP to obtain the IP address of the switch's management interface, TFTP to obtain subsequent device-specific configuration, and a public-key infrastructure to provision the domain certificate on the device. Once the device has its domain certificate, it connects to the SDN controller with the TLS protocol. Now the device can obtain forwarding rules from the controller.

To provide these rules, the network applications need to know the physical topology of the full network. SNMP crawling can be used to construct a view of the connections between the legacy devices, and LLDP can be used to construct a view of the connections between the SDN devices. In order to learn about the physical connections between the legacy and SDN devices, a broadcast version of the LLDP protocol can be used.

Lastly, we have discussed how existing protocols, such as PCEP and BGP, can be combined with SDN protocols, such as OpenFlow, and how they can be used to communicate rules to the forwarding devices and thereby enable packet forwarding in the mixed networks.

Chapter 4

Implementation and evaluation

This chapter provides details about the proof-of-concept prototype developed in order to demonstrate automatic bootstrapping of SDN switches. The implementation corresponds to the processes described in Section 3.2 and Section 3.3. This chapter is organized as follows: Section 4.1 describes the emulation testbed and the tools used. Section 4.2 outlines the implementation wherein we describe the initial configuration tasks and the bootstrap process. Lastly, Section 4.3 discusses the limitations of the implementation and suggests some improvements.

4.1 Emulation testbed

We use Mininet¹ [71] as the emulation platform for our implementation and run our tests on a computer with Ubuntu 12.04. Mininet is written in Python and provides a Python based API for the creation of hosts, switches, and controllers. All the network nodes run on the same machine as lightweight virtual machines, which are essentially processes running in their own name spaces. However, all the processes share the computer's file system. Mininet also emulates links between the nodes as virtual Ethernet pairs. In our emulation, we do not use the switch and controller classes provided by Mininet. Instead, all the nodes created are based on the `mininet.node.Host` class. We write our own Python code to create switch and controller processes on the hosts. This is needed in order to have a fine-grained control over each of these processes. Furthermore, Mininet allows setting host properties, such as maximum CPU, and link metrics, such as latency and bandwidth. However, we do not use any of these properties as the aim of our prototype is not to measure performance but to demonstrate the functional correctness of the

¹<http://mininet.org>

proposed bootstrapping process. It is worth mentioning that testing in an emulated environment such as Mininet does not guarantee correctness of the principles or the implementation. However, it can expose missing parts or inadequacies in the process. Furthermore, experience shows that such testing will often detect flaws that escape theoretical analysis (which may be based on false assumptions) or go undetected in experimental deployment.

4.1.1 Emulation topology

We emulate the topology of NORDUnet [3] obtained from the Internet topology database [2]. NORDUnet provides IP/MPLS connectivity between research and educational networks of the Nordic countries. As depicted in Fig. 4.1(a), the NORDUnet backbone consists of 6 switches located in Helsinki, Stockholm, Copenhagen, Oslo, Reykjavik, and Hamburg. In this rest of the chapter, we refer to the backbone switches by the name of the city; for instance the switch located at Helsinki is referred to as the Helsinki switch. In addition to the backbone switches, we add another switch and place it at the network operation center (NOC) in Stockholm. We refer to this switch as the NOC switch. It interfaces externally with the backbone network, and internally with the servers required for bootstrapping the switches in the NORDUnet network. The NOC network topology is depicted in Fig. 4.1(b).

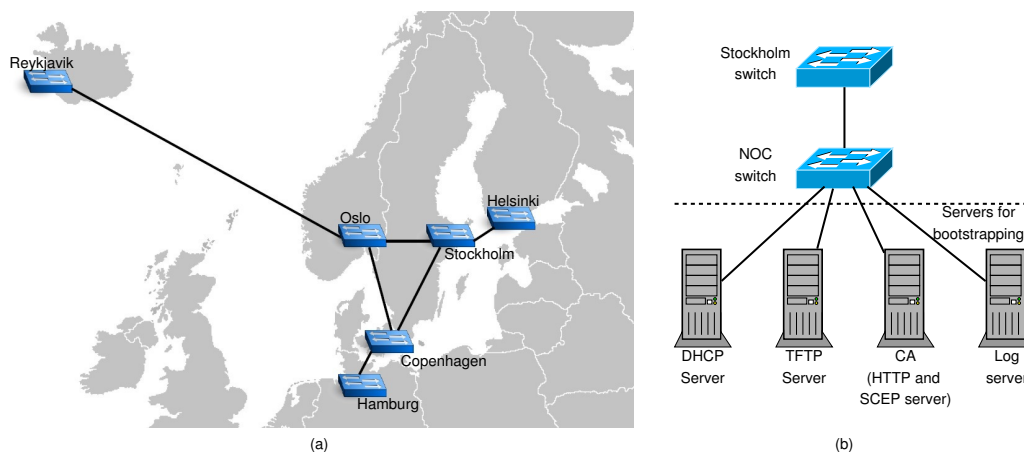


Figure 4.1: Emulated topology of NORDUnet

4.1.2 Tools used

In order to implement the SDN bootstrapping process, the following software was used:

- The DHCP server is implemented using `isc-dhcpd-4.2.8` and the switches run `isc-dhclient-4.2.8`.
- The TFTP server and client are implemented using `tftp-hpa 5.2`.
- The CA runs `OpenSCEP2` server version 0.4.2 to provide certificate enrollment to switches. `OpenSCEP` uses `OpenSSL` to perform PKI operations. We had to make a few modifications to the `OpenSCEP` server to make it compatible with the newer version of `OpenSSL` (`OpenSSL 1.0.1`).
- The CA also serves as the certificate repository and provides clients with HTTP access to download its root certificate. The HTTP server used is `Apache/2.2.22`. A fingerprint is used to check the integrity of the downloaded root certificate.
- The log server runs `syslog-ng 3.6.2` and the clients use a program written in C to periodically report logs to the server.
- Legacy switches and the SDN switches are both implemented using `ovs-vsitchd 2.1.0`. This version of the Open vSwitch supports the `OpenFlow` protocol version 1.3 (and earlier).
- The controller used is the test-controller provided along with Open vSwitch. The controller was earlier called `ovs-controller` but was renamed by the developers of Open vSwitch to represent the fact that it is not ready for use in a production environment. However, it provides the required functionality for our prototype.

It should be noted that the above software and servers are only used for bootstrapping the backbone switches in the NORDUnet network.

4.2 Bootstrapping process

This section is divided into three parts; Section 4.2.1 describes the preparation of the infrastructure needed for the bootstrapping process. Section 4.2.2 describes the initial configuration of the switches. Lastly, Section 4.2.3 describes the steps performed by the SDN switch during the bootstrapping process. We demonstrate the automatic bootstrapping of the Helsinki switch. All the other switches are legacy switches.

²<http://openscep.othello.ch/>

4.2.1 Infrastructure preparation

Before the bootstrap process begins, the DHCP server, TFTP server and CA are configured manually. These are the tasks that an operator should perform before powering on the SDN device. First the DHCP server is configured with the Helsinki switch's host name, the IP address to allocate to the switch, and the IP addresses (and port numbers) of the TFTP server, log server and OpenFlow controller. Then a configuration file is created on the TFTP server. The contents of this file include the URL of the repository containing the operator root certificate, the fingerprint of this certificate, and the SCEP enrollment URL. Lastly, the serial number of the device is added to the white-list saved on the CA. In the emulation, we use the same string as the host name and the device serial number. Additionally, the controller is also configured with the operator root certificate and the controller's domain certificate.

4.2.2 Initial switch configuration

Both the SDN and legacy switches in the emulation are implemented using Open vSwitch. This is achieved as follows: the OpenFlow specification [5] specifies two modes of operation when an OpenFlow switch loses connectivity with the controller. In the first mode called *fail secure*, the switch behaves like an OpenFlow switch and requires rules from the controller to forward packets. However, in the second mode called *fail standalone*, the switch behaves like a normal Ethernet switch. We set up all the legacy switches in the fail standalone mode from the start of the emulation and thereby achieve legacy switching behaviour.

Our demonstration starts with the Helsinki switch also in the fail standalone mode. However, we set the switch to not forward any packets by setting the kernel value `net.ipv4.ip_forward` to 0. Therefore the switch behaves like an end-host.

4.2.3 Integrating the switch into the network

The steps performed by the Helsinki switch in order to integrate into the network are described below. Fig. 4.2 also depicts these steps and provides a list of the values obtained in each step of the process.

- Step 1 – The switch starts the DHCP client process. The server identifies the switch based on the client identifier and returns the switch's

host name, the IP address of the management interface, and the IP addresses (and port numbers) of the TFTP server, log server and OpenFlow controller.

- Step 2 – The switch then connects to the log server and reports its existence by logging its host name.
- Step 3 – Using the TFTP server address obtained via TFTP, the switch downloads a host-specific configuration file. The file is identified by the host name received through DHCP.
- Step 4 – The switch reads the certificate repository URL and the expected fingerprint from the TFTP configuration file. It then downloads the operator root certificate and verifies that the certificate fingerprint matches the expected one. Then the switch performs certificate enrollment over SCEP and obtains the domain certificate when the request is approved by the CA. It now reports its certificate details to the log server.
- Step 5 – The switch’s mode is changed to fail secure and forwarding is enabled. The switch then establishes a mutually authenticated TLS connection with the controller and starts to receive forwarding rules over the OpenFlow protocol. It now includes the controller details in the messages to the log server.

At this point, the switch is connected to the network and SDN functionality is enabled. A snippet of the output from the emulation is shown in Fig. 4.3.

4.3 Further improvements

The prototype system is only in its first stage of implementation and there is much scope for improvement. There is no error handling incorporated in the emulation. For instance, there is currently no provision to handle cases when the switch fails to obtain an IP address via DHCP or to download the configuration file from the TFTP server. There should be appropriate timeout values, and upon failure, debug messages should be displayed to the user. Also, the demonstration is sequential in nature. The CA grants the request immediately upon receiving the request. However in practical deployments this is typically not the case. Hence, there should be mechanisms incorporated into the emulation to test the switch and server behaviour when there are delays at different stages of the process. Furthermore, the private keys are stored as text files on the file system of the computer running the

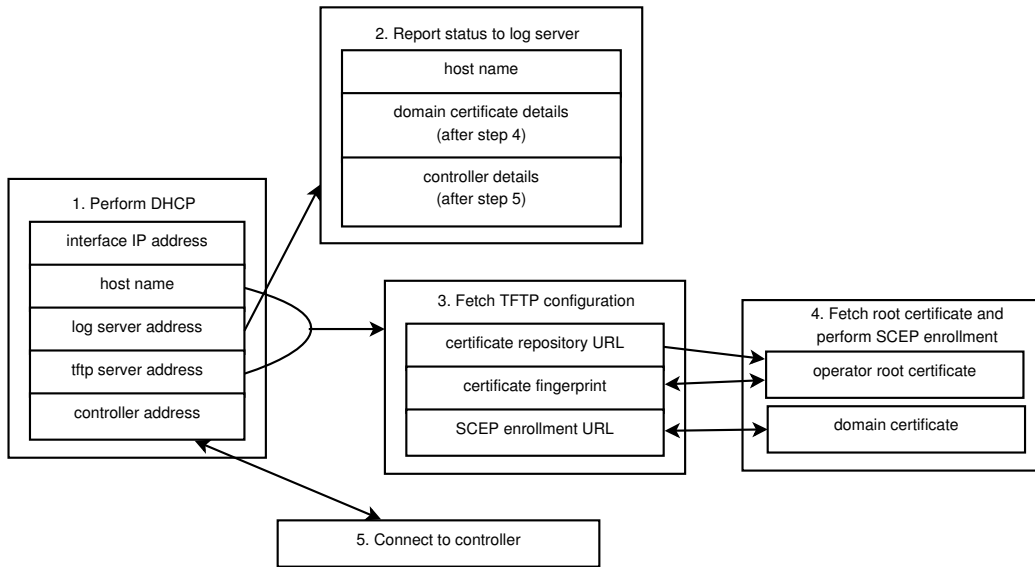
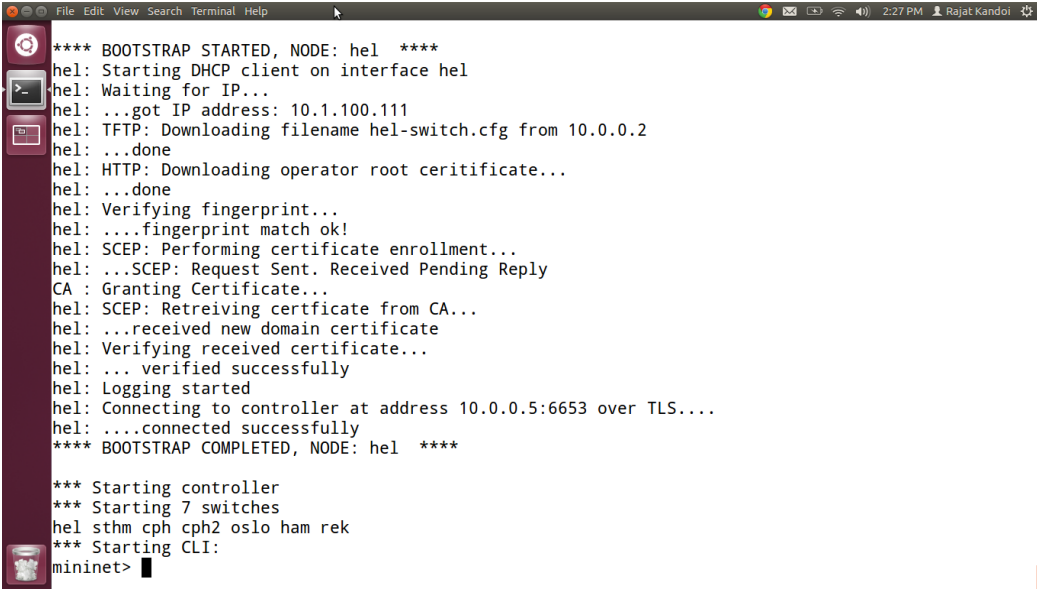


Figure 4.2: Bootstrap process

emulation. Eventually, secure hardware storage such as the trusted platform module should be used for secure confidential storage. Lastly, we have not implemented topology discovery and network applications in our prototype. Instead, the controller uses ARP flooding and MAC learning to learn the location of the destination and to configure the switch with forwarding rules. Such a method is not practical in backbone networks like NORDUnet. It would also be valuable to extend the implementation and include MPLS based forwarding into the demonstration.



```
**** BOOTSTRAP STARTED, NODE: hel ****
hel: Starting DHCP client on interface hel
hel: Waiting for IP...
hel: ..got IP address: 10.1.100.111
hel: TFTP: Downloading filename hel-switch.cfg from 10.0.0.2
hel: ..done
hel: HTTP: Downloading operator root certificate...
hel: ..done
hel: Verifying fingerprint...
hel: ...fingerprint match ok!
hel: SCEP: Performing certificate enrollment...
hel: ..SCEP: Request Sent. Received Pending Reply
CA : Granting Certificate...
hel: SCEP: Retrieving certificate from CA...
hel: ..received new domain certificate
hel: Verifying received certificate...
hel: ... verified successfully
hel: Logging started
hel: Connecting to controller at address 10.0.0.5:6653 over TLS...
hel: ...connected successfully
**** BOOTSTRAP COMPLETED, NODE: hel ****

*** Starting controller
*** Starting 7 switches
hel sthm cph cph2 oslo ham rek
*** Starting CLI:
mininet>
```

Figure 4.3: Screenshot of emulation output

Chapter 5

Discussion

This chapter is organized as follows: Section 5.1 discusses aspects that operators must consider in order to achieve carrier grade performance, Section 5.2 reviews management tasks that can benefit from SDN deployments in telecommunication networks, Section 5.3 describes the lessons learned while implementing the prototype, and lastly, Section 5.4 discusses ideas for future research.

5.1 Meeting carrier grade requirements

SDN technologies for telecommunication networks must provide high performance, be highly available and also achieve scalability. In this section, we discuss how design decisions such as flow rule installation strategies, number of controllers deployed, and the placement of these controllers can influence meeting these carrier grade requirements.

Flow rule installation can be performed reactively or proactively. A reactive strategy allows for fine-grained control over traffic flows but it increases the number of routing decisions the controller needs to make. In large scale networks, a controller may need to process millions of flows per second and, thereby, it could become a processing bottleneck. A reactive strategy also increases latency as the switch needs a forwarding rule from the controller for every new flow. This introduces a delay of one round-trip time for communication between the controller and the switch. Therefore, operators cannot rely on a purely reactive strategy and must consider using a combination of proactive and reactive strategies in order to achieve high performance.

Another aspect to consider is the centralization of the control-plane. If only one controller is deployed in the network, then the controller becomes a single point of failure in addition to becoming the performance bottleneck.

To achieve scalability and reliability, operators should deploy several physically distributed controllers. Onix [67] is an example of such a distributed control-plane platform. However, to maintain logically centralized control in a distributed architecture, operators and application designers must take in to account the CAP theorem [30] which states that it is impossible for a distributed architecture to be consistent, available and partition tolerant all at the same time. Furthermore, to improve scalability, DevoFlow [37] and DIFANE [114] propose relegating some of the control-plane tasks back to the switches themselves. Additionally, [42] proposes a mechanism for dynamically changing the controller to which a switch is connected based on the temporal and spatial distribution of the data plane traffic.

Lastly, operators need to determine how many controllers are required for their networks and where these controllers should be placed. The authors of [57] point out that the optimal placement of the controller is strongly dependent on the network topology. They also show that adding k controllers into the network reduces the network latency by a factor of k . However, there is always a trade-off between the worst-case latency and average latency depending on the location of the controller. Furthermore, [93] develops heuristics for the number of controllers required in a network. They demonstrate that having 10 controllers is often sufficient to achieve 99.999% reliability in most of the network topologies they analyzed.

5.2 Managing SDN networks after deployment

This thesis has focused on the bootstrapping phase of network management. However, centralized control and visibility over the full network topology creates opportunities for other management functions to benefit from SDN and justifies its deployment. Network monitoring, traffic engineering, and operations automation are likely to be the initial management tasks that utilize SDN [91]. We now discuss these management tasks and provide examples of how they have been realized using OpenFlow.

Network monitoring is a crucial component in flow-based programmable SDN networks as applications require timely and accurate information about, for example, link utilization in order to make optical routing decisions. Network monitoring can be performed actively or passively. In the active method, the controller periodically polls switches in order to obtain flow statistics whereas, in the passive method, the controller gains flow information by observing the control-plane communication. OpenNetMon [108] is an example of an active monitoring scheme where the controller periodically sends flow statistics requests to the flow end points (source and destination switches)

and collects information regarding the bandwidth and packet loss. However, this method introduces an overhead on the control-plane channel and the overhead increases with the frequency of the monitoring. On the other hand, FlowSense [113] proposes a passive scheme wherein the controller listens to packet-in and flow-removed messages and, based on them, determines the average utilization of link over a period of time. While this method eliminates any monitoring overhead, the information is gathered at longer and variable intervals. [35] combines these methods and proposes that the monitoring application maintains a timer value for each flow. If the flow ends before the timer expires, the controller will receive a flow removed message. However, if the timer expires, a flow statistics request is sent to the switch. This method balances the timeliness of the flow information and the control-plane overhead.

With sufficient network monitoring capabilities, applications can get real time information on the traffic status and link status and route traffic on paths that can satisfy *traffic engineering* requirements. These requirements include guaranteeing quality of service (QoS), load balancing traffic over network links and servers, and providing resilience and fast recovery through path protection schemes. FlowQoS [100] describes mechanisms for enforcing QoS per application on the home broadband access network. Such a mechanism could be extended to customer-premise equipment deployed by network operators. [53, 55] demonstrate schemes how load balancing web traffic using OpenFlow can reduce response time for web services. Lastly, [102] demonstrates how segment protection can be provisioned with OpenFlow leading to recovery times of about 60ms.

SDN can also enable the automatic configuration and deployment of virtual network functions. The recently proposed OF-CONFIG protocol [4] can be used to instantiate OpenFlow data planes and manage resources such as queues and ports on these switches. Additionally, SDN combined with Network Functions Virtualization (NFV) will enable the migration towards a completely software-based network platform. NFV is an initiative by the European Telecommunications Standards Institute that aims at decoupling network functions (e.g. network address translation and firewalls) from proprietary hardware and deploying them as software modules on commercial off-the-self hardware by employing standard IT virtualization schemes. The SDN controller can be used to control virtual network resources as well as to provision service chaining between the different virtual network functions [95, 115].

5.3 Reflections on the implementation

We would like to highlight three important lessons learned while developing our prototype system. First, Mininet is a great platform for network emulation. The Python API is feature rich and well documented. However, it is worth exploring platforms such as Mininet CE [17] and MaxiNet [112] that enable the emulation of much larger topologies. Secondly, we noticed a lack of open source implementations of certificate management protocols. The OpenSCEP server used in our implementation was developed over a decade ago. Research would greatly benefit from an open source implementation of the certificate management protocol [13]. Lastly, the software used for the thesis were the latest versions available when developing the prototype. Hence they were sometimes buggy and did not compile when built directly from the Ubuntu repositories. We were able to get a deeper knowledge of the code and functionality by building them by hand.

5.4 Future work

The work in this thesis can be extended by emulating a complete telecommunication network topology with a clear demarcation between network sections. Then, instead of using only OpenFlow for all network sections, each network section could use a specific southbound protocol. For example, the access network could use OpenFlow, the aggregation network PCEP, and the core network BGP. Additionally, the emulation could include link metrics such as bandwidth and delay and also impose restrictions on the CPU capacity of the controller nodes in order to better understand factors like flow set up latency. Furthermore, the performance can be evaluated by using traffic traces from real networks. Network applications can be developed for optimal routing and the network resource utilization can be compared to that of legacy networks. Also, feasibility of the proposed approach to topology discovery could be studied. Lastly, it would also be valuable to test the end-to-end process with real hardware equipment.

Chapter 6

Conclusion

This thesis has focused on the deployment of SDN in telecommunication networks and has provided a comprehensive survey of factors an operator must consider while deploying SDN. We have investigated strategies for the deployment of new programmable forwarding hardware, described the engineering process to bring up the devices in an automated manner, reviewed various southbound protocols, and also examined network topology discovery.

We believe that operators are most likely to opt for an incremental deployment of SDN using hybrid forwarding devices. Such an approach will enable operators to move to an SDN-ready network while they develop the required network applications. Subsequently, operators can gain confidence in SDN by carrying out field trials where low priority traffic is handled by SDN and all other traffic is handled using legacy protocols. Additionally, SDN devices are likely to be placed at the edges of the network. Operators can then harness SDN benefits such as centralized policy control at the point where traffic enters or exits the network.

Furthermore, operators may be required to deploy more networking hardware in order to cope with the rapidly growing Internet traffic. Automating the bringing up of network devices will reduce the operational costs and allow faster deployment of new hardware. We have devised an automated process that enables the newly installed device to obtain IP connectivity, perform certificate enrollment operations to obtain a domain certificate, and then connect securely to the SDN controller using this certificate. We have also proposed the use of logging servers to monitor the bootstrapping process. To evaluate the functional correctness of the automated process, we have emulated the NORDUnet network topology using Mininet and performed the automatic bootstrapping for one of the switches in the network.

Once the device is connected to the controller, it receives rules for packet forwarding over southbound protocols. OpenFlow has emerged as the pre-

dominant southbound protocol for this task. However, existing protocols such as PCEP and BGP can also be used for this task. Using these protocols has the advantage that operators will need only a software upgrade to support the modified protocols instead of upgrading their network hardware. It also allows operators to build on their knowledge of these protocols. We have reviewed various southbound protocols and highlighted the use cases for each of these protocols. We think that networks are likely to make use of several of these protocols simultaneously.

Lastly, we have reviewed how the physical network topology can be discovered in hybrid networks where legacy and SDN devices interoperate. Topology discovery is crucial in SDN networks as applications require a global network view to be able to make optimal routing decisions. On one hand, it is possible to perform topology discovery in legacy networks by using existing techniques such as SNMP crawling. On the other hand, topology discovery in SDN networks can be performed by using the existing LLDP protocol although the controller rather than the forwarding devices will create the LLDP packets. By combining both these techniques, the entire network topology can be discovered.

SDN presents a great opportunity for network operators to improve network efficiency through centralization of control-plane decisions. It also allows reduction in operational costs through lower cost hardware and by automating processes such as network management. However, for SDN to be successfully deployed in telecommunication networks, research must focus on developing a seamless migration path from the existing legacy network, first towards a hybrid SDN network, and eventually towards a network comprised of purely SDN devices.

Bibliography

- [1] Interface to the Routing System (i2rs) - Charter. <http://datatracker.ietf.org/wg/i2rs/charter/>. Accessed on 09.02.2015.
- [2] The Internet Topology Zoo. <http://www.topology-zoo.org/> Accessed on 23.06.2015.
- [3] NORDUnet. <https://www.nordu.net/> Accessed on 23.06.2015.
- [4] OpenFlow Management and Configuration Protocol. Open Networking Foundation (ONF). Technical Specification. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf>. Accessed on 09.02.2015.
- [5] OpenFlow Switch Specification (Version 1.3.0). Open Networking Foundation (ONF). Technical Specification. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf>. Accessed on 09.02.2015.
- [6] CMP and CMC Comparison. Discussion, 3GPP, May 2002. http://www.3gpp.org/ftp/tsg_sa/wg3_security/tsgs3_24_helsinki/docs/pdf/S3-020366.pdf.
- [7] An Overview of Digital Certificates and How They Are Used in VPN Authentication . Tech. rep., SANS Institute, 2003. <http://www.giac.org/paper/gsec/2711/overview-digital-certificates-vpn-authentication/104625> Accessed on 02.06.2015.
- [8] IEEE standard for local and metropolitan area networks: Media access control (mac) bridges. *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)* (June 2004), 1–277.

- [9] Telecom Network Planning for evolving Network Architectures. Reference manual draft version 5.1 part 1, International Telecommunication Union, Jan 2008. https://www.itu.int/ITU-D/tech/NGN/Manual/Version5/NPM_V05_January2008_PART1.pdf Accessed on 02.06.2015.
- [10] IEEE standard for local and metropolitan area networks– station and media access control connectivity discovery. *IEEE Std 802.1AB-2009 (Revision of IEEE Std 802.1AB-2005)* (Sept 2009), 1–204.
- [11] Extending MPLS Across the End-to-End Network: Cisco Unified MPLS. White paper, Cisco, 2011. http://www.cisco.com/c/en/us/products/collateral/optical-networking/carrier-packet-transport-cpt-system/white_paper_c11-656286.pdf Accessed on 02.06.2015.
- [12] March 2012 interoperability event white paper. White paper, Open Networking Foundation, April 2012. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-test/onf-testing-interop-march-2012-whitepaper-v1.0.pdf> Accessed on 09.02.2015.
- [13] ADAMS, C., FARRELL, S., KAUSE, T., AND MONONEN, T. Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP). RFC 4210, RFC Editor, September 2005. <http://www.rfc-editor.org/rfc/rfc4210.txt>.
- [14] ALEXANDER, S., AND DROMS, R. Dhcp options and bootp vendor extensions. RFC 2132, RFC Editor, March 1997. <http://www.rfc-editor.org/rfc/rfc2132.txt>.
- [15] ALSHAMSI, A., AND SAITO, T. A technical comparison of ipsec and ssl. In *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on* (March 2005), vol. 2, pp. 395–398 vol.2.
- [16] ANTIKAINEN, M., AURA, T., AND SÄRELÄ, M. Spook in Your Network: Attacking an SDN with a Compromised OpenFlow Switch. In *The 19th Nordic Conference on Secure IT Systems (NordSec 2014)* (October 2014).
- [17] ANTONENKO, V., AND SMELYANSKIY, R. Global network modelling based on mininet approach. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking* (New York, NY, USA, 2013), HotSDN '13, ACM, pp. 145–146.

- [18] ATLAS, A., HALPERN, J., HARES, S., WARD, D., AND NADEAU, T. An architecture for the interface to the routing system. Work in progress draft-ietf-i2rs-architecture-08, January 2015. <http://www.ietf.org/internet-drafts/draft-ietf-i2rs-architecture-08.txt>.
- [19] ATLAS, A., NADEAU, T., AND WARD, D. Interface to the routing system problem statement. Work in progress draft-ietf-i2rs-problem-statement-06, January 2015. <http://www.ietf.org/internet-drafts/draft-ietf-i2rs-problem-statement-06.txt>.
- [20] BAHADUR, N., FOLKES, R., KINI, S., AND MEDVED, J. Routing information base info model. Work in progress draft-ietf-i2rs-rib-info-model-05, January 2015. <http://www.ietf.org/internet-drafts/draft-ietf-i2rs-rib-info-model-05.txt>.
- [21] BATES, T., CHANDRA, R., KATZ, D., AND REKHTER, Y. Multi-protocol extensions for bgp-4. RFC 4760, RFC Editor, January 2007. <http://www.rfc-editor.org/rfc/rfc4760.txt>.
- [22] BEHRINGER, M., PRITIKIN, M., AND BJARNASON, S. Making the internet secure by default. Work in progress draft-behringer-default-secure-00, IETF Secretariat, January 2014. <http://www.ietf.org/internet-drafts/draft-behringer-default-secure-00.txt>.
- [23] BEJERANO, Y. Taking the skeletons out of the closets: A simple and efficient topology discovery scheme for large ethernet lans. In *INFO-COM 2006. 25th IEEE International Conference on Computer Communications. Proceedings* (April 2006), pp. 1–13.
- [24] BENANTAR, M. The internet public key infrastructure. *IBM Systems Journal* 40, 3 (2001), 648–665.
- [25] BENTON, K., CAMP, L. J., AND SMALL, C. OpenFlow Vulnerability Assessment. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking* (New York, NY, USA, 2013), HotSDN '13, ACM, pp. 151–152.
- [26] BIERMAN, A., AND JONES, K. Physical Topology MIB. RFC 2922, RFC Editor, September 2000. <http://www.rfc-editor.org/rfc/rfc2922.txt>.
- [27] BJORKLUND, M. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020, RFC Editor, October 2010. <http://www.rfc-editor.org/rfc/rfc6020.txt>.

- [28] BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C., MALER, E., AND YERGEAU, F. Extensible Markup Language (XML) 1.0 (Fifth Edition). Tech. rep., World Wide Web Consortium REC-xml-20081126, November 2008. <http://www.w3.org/TR/xml/>.
- [29] BREITBART, Y., GAROFALAKIS, M., JAI, B., MARTIN, C., RASTOGI, R., AND SILBERSCHATZ, A. Topology discovery in heterogeneous ip networks: the netinventory system. *Networking, IEEE/ACM Transactions on* 12, 3 (June 2004), 401–414.
- [30] BREWER, E. Pushing the cap: Strategies for consistency and availability. *Computer* 45, 2 (Feb. 2012), 23–29.
- [31] CASADO, M., KOPONEN, T., SHENKER, S., AND TOOTOONCHIAN, A. Fabric: A retrospective on evolving sdn. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks* (New York, NY, USA, 2012), HotSDN '12, ACM, pp. 85–90.
- [32] CASE, J., MUNDY, R., PARTAIN, D., AND STEWART, B. Introduction and Applicability Statements for Internet-Standard Management Framework. RFC 3410, RFC Editor, December 2002. <http://www.rfc-editor.org/rfc/rfc3410.txt>.
- [33] CASELLAS, R., MARTINEZ, R., MUNOZ, R., LIU, L., TSURITANI, T., AND MORITA, I. An integrated stateful PCE / OpenFlow controller for the control and management of flexi-grid optical networks. In *Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC), 2013* (March 2013), pp. 1–3.
- [34] CASELLAS, R., MARTINEZ, R., MUNOZ, R., VILALTA, R., LIU, L., TSURITANI, T., AND MORITA, I. Control and management of flexi-grid optical networks with an integrated stateful path computation element and OpenFlow controller [invited]. *Optical Communications and Networking, IEEE/OSA Journal of* 5, 10 (Oct 2013), A57–A65.
- [35] CHOWDHURY, S., BARI, M., AHMED, R., AND BOUTABA, R. Payless: A low cost network monitoring framework for software defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE* (May 2014), pp. 1–9.
- [36] COLTUN, R., FERGUSON, D., MOY, J., AND LINDEM, A. OSPF for IPv6. RFC 5340, RFC Editor, July 2008. <http://www.rfc-editor.org/rfc/rfc5340.txt>.

- [37] CURTIS, A. R., MOGUL, J. C., TOURRILHES, J., YALAGANDULA, P., SHARMA, P., AND BANERJEE, S. Devoflow: Scaling flow management for high-performance networks. *SIGCOMM Comput. Commun. Rev.* 41, 4 (Aug. 2011), 254–265.
- [38] DAS, S., PARULKAR, G., AND MCKEOWN, N. Unifying Packet and Circuit Switched Networks. In *GLOBECOM Workshops, 2009 IEEE* (Nov 2009), pp. 1–6.
- [39] DAS, S., PARULKAR, G., MCKEOWN, N., SINGH, P., GETACHEW, D., AND ONG, L. Packet and circuit network convergence with open-flow. In *Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC)* (March 2010), pp. 1–3.
- [40] DEVLIC, A., JOHN, W., AND SKOLDSTROM, P. A use-case based analysis of network management functions in the onf sdn model. In *Software Defined Networking (EWSDN), 2012 European Workshop on* (Oct 2012), pp. 85–90.
- [41] DIERKS, T., AND RESCORLA, E. The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor, August 2008. <http://www.rfc-editor.org/rfc/rfc5246.txt>.
- [42] DIXIT, A., HAO, F., MUKHERJEE, S., LAKSHMAN, T., AND KOMPPELLA, R. Towards an elastic distributed sdn controller. *SIGCOMM Comput. Commun. Rev.* 43, 4 (Aug. 2013), 7–12.
- [43] DODIG-CRANKOVIC, G. Constructive research and info-computational knowledge generation. *Model-Based Reasoning in Science and Technology* (2010).
- [44] DODIG-CRANKOVIC, GORDANA. Scientific Methods in Computer Science. In *Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden* (Apr 2002).
- [45] DROMS, R. Dynamic host configuration protocol. RFC 2131, RFC Editor, March 1997. <http://www.rfc-editor.org/rfc/rfc2131.txt>.
- [46] ENNS, R., BJORKLUND, M., SCHOENWAELDER, J., AND BIERMAN, A. Network Configuration Protocol (NETCONF). RFC 6241, RFC Editor, June 2011. <http://www.rfc-editor.org/rfc/rfc6241.txt>.

- [47] FEAMSTER, N., BALAKRISHNAN, H., REXFORD, J., SHAIKH, A., AND VAN DER MERWE, J. The case for separating routing from routers. In *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture* (New York, NY, USA, 2004), FDNA '04, ACM, pp. 5–12.
- [48] FEITELSON, D. G. Experimental Computer Science: The Need for a Cultural Change, Dec 2006. Internet version: <http://www.cs.huji.ac.il/~feit/papers/exp05.pdf>.
- [49] FLOODLIGHT, P. LinkDiscoveryManager (Dev). <http://docs.projectfloodlight.org/display/floodlightcontroller/LinkDiscoveryManager+%28Dev%29>. Accessed on 09.02.2015.
- [50] FRANKEL, S., AND KRISHNAN, S. Ip security (ipsec) and internet key exchange (ike) document roadmap. RFC 6071, RFC Editor, February 2011. <http://www.rfc-editor.org/rfc/rfc6071.txt>.
- [51] GÄMPEL, A., KOTRONIS, V., AND DIMITROPOULOS, X. Evaluating the effect of centralization on routing convergence on a hybrid bgp-sdn emulation framework. *SIGCOMM Comput. Commun. Rev.* 44, 4 (Aug. 2014), 369–370.
- [52] GREDLER, H., MEDVED, J., PREVIDI, S., FARREL, A., AND RAY, S. North-bound distribution of link-state and te information using bgp. Work in progress draft-ietf-idr-ls-distribution-10, January 2015. <http://www.ietf.org/internet-drafts/draft-ietf-idr-ls-distribution-10.txt>.
- [53] H, N., SEETHARAMAN, S., MCKEOWN, N., AND JOHARI, R. Plug-n-serve: Load-balancing web traffic using open-flow, 2009. <http://conferences.sigcomm.org/sigcomm/2009/demos/sigcomm-pd-2009-final26.pdf>, Accessed on 26.06.15.
- [54] HAAS, J., AND HARES, S. Definitions of managed objects for bgp-4. RFC 4273, RFC Editor, January 2006. <http://www.rfc-editor.org/rfc/rfc4273.txt>.
- [55] HANDIGOL, N., SEETHARAMAN, S., FLAJSLIK, M., GEMBER, A., MCKEOWN, N., PARULKAR, G., AKELLA, A., FEAMSTER, N., CLARK, R., KRISHNAMURTHY, A., BRAJKOVIC, V., AND ANDERSON, T. Aster*x: Load-Balancing Web Traffic over Wide-Area. <http://yuba.stanford.edu/~nikhilh/pubs/handigol-gec9.pdf>, Accessed on 26.06.15.

- [56] HARTMANIS, J. Response to the Essay: “On Computational Complexity and the Nature of Computer Science”. *ACM Comput. Surv.* 27, 1 (Mar. 1995), 59–61.
- [57] HELLER, B., SHERWOOD, R., AND MCKEOWN, N. The controller placement problem. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks* (New York, NY, USA, 2012), HotSDN '12, ACM, pp. 7–12.
- [58] HOUSLEY, R., AND SANTESSON, S. Update to directorystring processing in the internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 4630, RFC Editor, August 2006. <http://www.rfc-editor.org/rfc/rfc4630.txt>.
- [59] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. Intermediate system to Intermediate system intra-domain routing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode Network Service (ISO 8473). International Standard ISO/IEC 10589:2002, Second Edition, November 2002.
- [60] JAIN, S., KUMAR, A., MANDAL, S., ONG, J., POUTIEVSKI, L., SINGH, A., VENKATA, S., WANDERER, J., ZHOU, J., ZHU, M., ZOLLA, J., HÖLZLE, U., STUART, S., AND VAHDAT, A. B4: Experience with a globally-deployed software defined wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (New York, NY, USA, 2013), SIGCOMM '13, ACM, pp. 3–14.
- [61] JENS GUSTEDT, EMMANUEL JEANNOT, MARTIN QUINSON. Experimental Methodologies for Large-Scale Systems: a Survey. In *Parallel Processing Letters, World Scientific Publishing* (2009). <https://hal.inria.fr/inria-00364180v2/document>.
- [62] JOHN, W., KERN, A., KIND, M., SKOLDSTROM, P., STAESSENS, D., AND WOESNER, H. Splitarchitecture: Sdn for the carrier domain. *Communications Magazine, IEEE* 52, 10 (October 2014), 146–152.
- [63] JOYAL, D., GALECKI, P., GIACALONE, S., COLTUN, R., AND BAKER, F. Ospf version 2 management information base. RFC 4750, RFC Editor, December 2006. <http://www.rfc-editor.org/rfc/rfc4750.txt>.
- [64] KEMPF, J., WHYTE, S., ELLITHORPE, J., KAZEMIAN, P., HAITJEMA, M., BEHESHTI, N., STUART, S., AND GREEN, H. Openflow

- mpls and the open source label switched router. In *Teletraffic Congress (ITC), 2011 23rd International* (Sept 2011), pp. 8–14.
- [65] KIRAN, S., LAREAU, P., AND LLOYD, S. PKI Basics - A Technical Perspective. Tech. rep., PKI Forum, Nov 2002. http://www.oasis-pki.org/pdfs/PKI_Basics-A_technical_perspective.pdf.
- [66] KOBAYASHI, M., SEETHARAMAN, S., PARULKAR, G., APPENZELLER, G., LITTLE, J., VAN REIJENDAM, J., WEISSMANN, P., AND MCKEOWN, N. Maturing of openflow and software-defined networking through deployments. *Comput. Netw.* 61 (Mar. 2014), 151–175.
- [67] KOPONEN, T., CASADO, M., GUDE, N., STRIBLING, J., POUTIEVSKI, L., ZHU, M., RAMANATHAN, R., IWATA, Y., INOUE, H., HAMA, T., AND SHENKER, S. Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2010), OSDI'10, USENIX Association, pp. 1–6.
- [68] KOTRONIS, V., DIMITROPOULOS, X., AND AGER, B. Outsourcing the routing control logic: Better internet routing based on sdn principles. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2012), HotNets-XI, ACM, pp. 55–60.
- [69] KREUTZ, D., RAMOS, F., ESTEVES VERISSIMO, P., ESTEVE ROTHENBERG, C., AZODOLMOLKY, S., AND UHLIG, S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE* 103, 1 (Jan 2015), 14–76.
- [70] LANGE, C., KOSIANKOWSKI, D., WEIDMANN, R., AND GLADISCH, A. Energy consumption of telecommunication networks and related improvement options. *Selected Topics in Quantum Electronics, IEEE Journal of* 17, 2 (March 2011), 285–295.
- [71] LANTZ, B., HELLER, B., AND MCKEOWN, N. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (New York, NY, USA, 2010), Hotnets-IX, ACM, pp. 19:1–19:6.
- [72] LEVIN, D., CANINI, M., SCHMID, S., SCHAFFERT, F., AND FELDMANN, A. Panopticon: Reaping the benefits of incremental sdn deployment in enterprise networks. In *Proceedings of the 2014 USENIX*

- Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2014), USENIX ATC'14, USENIX Association, pp. 333–346.
- [73] LIN, P., HART, J., KRISHNASWAMY, U., MURAKAMI, T., KOBAYASHI, M., AL-SHABIBI, A., WANG, K.-C., AND BI, J. Seamless interworking of sdn and ip. *SIGCOMM Comput. Commun. Rev.* 43, 4 (Aug. 2013), 475–476.
- [74] LIU, L., CASELLAS, R., TSURITANI, T., MORITA, I., MARTINEZ, R., AND MUNOZ, R. Experimental demonstration of an OpenFlow/PCE integrated control plane for IP over translucent WSON with the assistance of a per-request-based dynamic topology server. In *Optical Communications (ECOC), 2012 38th European Conference and Exhibition on* (Sept 2012), pp. 1–3.
- [75] LIU, L., CHOI, H. Y., CASELLAS, R., TSURITANI, T., MORITA, I., MARTINEZ, R., AND MUNOZ, R. Demonstration of a dynamic transparent optical network employing flexible transmitters/receivers controlled by an openflow-stateless pce integrated control plane [invited]. *Optical Communications and Networking, IEEE/OSA Journal of* 5, 10 (Oct 2013), A66–A75.
- [76] LIU, L., TSURITANI, T., MORITA, I., GUO, H., AND WU, J. Openflow-based wavelength path control in transparent optical networks: A proof-of-concept demonstration. In *Optical Communication (ECOC), 2011 37th European Conference and Exhibition on* (Sept 2011), pp. 1–3.
- [77] LIU, L., ZHANG, D., TSURITANI, T., VILALTA, R., CASELLAS, R., HONG, L., MORITA, I., GUO, H., WU, J., MARTINEZ, R., AND MUNOZ, R. Field trial of an openflow-based unified control plane for multilayer multigranularity optical switching networks. *Lightwave Technology, Journal of* 31, 4 (Feb 2013), 506–514.
- [78] LOWEKAMP, B., O'HALLARON, D., AND GROSS, T. Topology discovery for large ethernet networks. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (New York, NY, USA, 2001), SIGCOMM '01, ACM, pp. 237–248.
- [79] MANZALINI, A., MINERVA, R., KAEMPFER, E., CALLEGARI, F., CAMPI, A., CERRONI, W., CRESPI, N., DEKEL, E., TOCK, Y.,

- TAVERNIER, W., CASIER, K., VERBRUGGE, S., COLLE, D., VILALTA, R., MUNOZ, R., CASELLAS, R., MARTINEZ, R., MAZZOCCA, N., AND MAINI, E. Manifesto of edge ict fabric. In *Intelligence in Next Generation Networks (ICIN), 2013 17th International Conference on* (Oct 2013), pp. 9–15.
- [80] MARQUES, P., SHETH, N., RASZUK, R., GREENE, B., MAUCH, J., AND MCPHERSON, D. Dissemination of flow specification rules. RFC 5575, RFC Editor, August 2009. <http://www.rfc-editor.org/rfc/rfc5575.txt>.
- [81] MAUGHAN, D., SCHNEIDER, M., AND SCHERTLER, M. Internet security association and key management protocol (isakmp). RFC 2408, RFC Editor, November 1998. <http://www.rfc-editor.org/rfc/rfc2408.txt>.
- [82] MAYORAL, A., VILALTA, R., MUNOZ, R., CASELLAS, R., AND MARTINEZ, R. Experimental validation of automatic lightpath establishment integrating OpenDayLight SDN controller and Active Stateful PCE within the ADRENALINE testbed. In *Transparent Optical Networks (ICTON), 2014 16th International Conference on* (July 2014), pp. 1–4.
- [83] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (Mar. 2008), 69–74.
- [84] MOHAPATRA, S. Integrated planning for next generation networks. In *Integrated Network Management-Workshops, 2009. IM '09. IFIP/IEEE International Symposium on* (June 2009), pp. 205–210.
- [85] MOY, J. OSPF Version 2. STD 54, April 1998. <http://www.rfc-editor.org/rfc/rfc2328.txt>.
- [86] MUNOZ, R., CASELLAS, R., MARTINEZ, R., AND VILALTA, R. Pce: What is it, how does it work and what are its limitations? *Lightwave Technology, Journal of* 32, 4 (Feb 2014), 528–543.
- [87] PANDEY, S., CHOI, M.-J., WON, Y. J., AND HONG, J. W.-K. SNMP-based enterprise IP network topology discovery. *International Journal of Network Management* (2011), 196–184.

- [88] PARKER, J. Management information base for intermediate system to intermediate system (is-is). RFC 4444, RFC Editor, April 2006. <http://www.rfc-editor.org/rfc/rfc4444.txt>.
- [89] PERLMAN, R. An overview of pki trust models. *Network, IEEE* 13, 6 (Nov 1999), 38–43.
- [90] PRITIKIN, M., NOURSE, A., AND VILHUBER, J. Simple Certificate Enrollment Protocol. Internet-Draft draft-nourse-scep-23, September 2011. <http://www.ietf.org/internet-drafts/draft-nourse-scep-23.txt>.
- [91] REINECKE, E. Mapping the future of software-defined networking, 2014. <http://goo.gl/fQCvRF> Accessed on 23.06.2015.
- [92] REKHTER, Y., LI, T., AND HARES, S. A border gateway protocol 4 (bgp-4). RFC 4271, RFC Editor, January 2006. <http://www.rfc-editor.org/rfc/rfc4271.txt>.
- [93] ROS, F. J., AND RUIZ, P. M. Five nines of southbound reliability in software-defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking* (New York, NY, USA, 2014), HotSDN '14, ACM, pp. 31–36.
- [94] ROTHENBERG, C. E., NASCIMENTO, M. R., SALVADOR, M. R., CORRÊA, C. N. A., CUNHA DE LUCENA, S., AND RASZUK, R. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks* (New York, NY, USA, 2012), HotSDN '12, ACM, pp. 13–18.
- [95] RUCKERT, J., BLENDIN, J., LEYMAN, N., SCHYGUDA, G., AND HAUSHEER, D. Demo: Software-defined network service chaining. In *Software Defined Networks (EWSDN), 2014 Third European Workshop on* (Sept 2014), pp. 139–140.
- [96] SCHAAD, J., AND MYERS, M. Certificate Management over CMS (CMC). RFC 5272, RFC Editor, June 2008. <http://www.rfc-editor.org/rfc/rfc5272.txt>.
- [97] SCHAFER, I., AND FELSER, M. Topology discovery in PROFINET. In *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on* (Sept 2007), pp. 704–707.

- [98] SCHENKER, S. The future of networking, the past of protocols, Oct 2011. [Online] <http://opennetsummit.org/archives/oct11/shenker-tue.pdf>.
- [99] SCHOENWAELDER, J. Overview of the 2002 IAB Network Management Workshop. RFC 3535, RFC Editor, May 2003. <http://www.rfc-editor.org/rfc/rfc3535.txt>.
- [100] SEDDIKI, M. S., SHAHBAZ, M., DONOVAN, S., GROVER, S., PARK, M., FEAMSTER, N., AND SONG, Y.-Q. Flowqos: Qos for the rest of us. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking* (New York, NY, USA, 2014), HotSDN '14, ACM, pp. 207–208.
- [101] SEZER, S., SCOTT-HAYWARD, S., CHOUHAN, P., FRASER, B., LAKE, D., FINNEGAN, J., VILJOEN, N., MILLER, M., AND RAO, N. Are we ready for sdn? implementation challenges for software-defined networks. *Communications Magazine, IEEE* 51, 7 (July 2013), 36–43.
- [102] SGAMBELLURI, A., GIORGETTI, A., CUGINI, F., PAOLUCCI, F., AND CASTOLDI, P. Openflow-based segment protection in ethernet networks. *Optical Communications and Networking, IEEE/OSA Journal of* 5, 9 (Sept 2013), 1066–1075.
- [103] SHARAFAT, A. R., DAS, S., PARULKAR, G., AND MCKEOWN, N. Mpls-te and mpls vpns with openflow. *SIGCOMM Comput. Commun. Rev.* 41, 4 (Aug. 2011), 452–453.
- [104] SHARMA, S., STAESSENS, D., COLLE, D., PICKAVET, M., AND DE-MEESTER, P. Automatic bootstrapping of openflow networks. In *Local Metropolitan Area Networks (LANMAN), 2013 19th IEEE Workshop on* (April 2013), pp. 1–6.
- [105] SIAMWALLA, R., SHARMA, R., AND KESHAV, S. Discovering Internetg Topology. In *Submitted to INFOCOM '99. 18th IEEE International Conference on Computer Communications* (1999).
- [106] TRCEK, D. *Managing Information Systems Security and Privacy*. Springer, 2006.
- [107] UNGERMAN, J. SDN for Service Providers. http://www.cisco.com/web/HR/ciscoconnect/2013/pdfs/software_defined_networks_sdn_for_service_providers.pdf, May 2013. Accessed on 09.02.2015.

- [108] VAN ADRICHEM, N., DOERR, C., AND KUIPERS, F. Opennetmon: Network monitoring in openflow software-defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE* (May 2014), pp. 1–8.
- [109] VASSEUR, J., AND ROUX, J. L. Path Computation Element (PCE) Communication Protocol (PCEP). RFC 5440, RFC Editor, March 2009. <http://www.rfc-editor.org/rfc/rfc5440.txt>.
- [110] VEREECKEN, W., VAN HEDDEGHEM, W., DERUYCK, M., PUYPE, B., LANNOO, B., JOSEPH, W., COLLE, D., MARTENS, L., AND DEMEESTER, P. Power consumption in telecommunication networks: overview and reduction strategies. *Communications Magazine, IEEE* 49, 6 (June 2011), 62–69.
- [111] VISSICCHIO, S., VANBEVER, L., AND BONAVENTURE, O. Opportunities and research challenges of hybrid software defined networks. *SIGCOMM Comput. Commun. Rev.* 44, 2 (Apr. 2014), 70–75.
- [112] WETTE, P., DRÄXLER, M., SCHWABE, A., WALLASCHEK, F., ZAHRAEE, M., AND KARL, H. Maxinet: Distributed emulation of software-defined networks. In *Networking Conference, 2014 IFIP* (June 2014), pp. 1–9.
- [113] YU, C., LUMEZANU, C., ZHANG, Y., SINGH, V., JIANG, G., AND MADHYASTHA, H. V. Flowsense: Monitoring network utilization with zero measurement cost. In *Proceedings of the 14th International Conference on Passive and Active Measurement* (Berlin, Heidelberg, 2013), PAM'13, Springer-Verlag, pp. 31–41.
- [114] YU, M., REXFORD, J., FREEDMAN, M. J., AND WANG, J. Scalable flow-based networking with difane. *SIGCOMM Comput. Commun. Rev.* 41, 4 (Aug. 2010), –.
- [115] ZHANG, Y., BEHESHTI, N., BELIVEAU, L., LEFEBVRE, G., MANGHIRMALANI, R., MISHRA, R., PATNEYT, R., SHIRAZIPOUR, M., SUBRAHMANYAM, R., TRUCHAN, C., AND TATIPAMULA, M. Steering: A software-defined networking for inline service chaining. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on* (Oct 2013), pp. 1–10.
- [116] ZHANG, Y., CHOWDHURY, P., TORNATORE, M., AND MUKHERJEE, B. Energy efficiency in telecom optical networks. *Communications Surveys Tutorials, IEEE* 12, 4 (Fourth 2010), 441–458.