RMIT UNIVERSITY

Automatic Provisioning in Multi-Domain Software Defined Networking

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

Franciscus Xaverius Ari Wibowo
M.Eng. in Telecommunication, Bandung Institute of Technology, Indonesia
B. Eng in Telecommunication, Telkom University, Indonesia

School of Engineering
College of Science, Engineering and Health
RMIT University
November 2018

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of this thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

Franciscus Xaverius Ari Wibowo
1/11/2018

Dedicated to my Family

Acknowledgements

I wish to acknowledge and express my thanks to those that have provided support, feedback and the opportunity to undertake my PhD research project.

I would like to express my sincere gratitude to my Senior Supervisor, Associate Professor Mark a. Gregory, for your belief, patience, motivation, visions and immense knowledge. You have inspired me not only to become a good researcher but also on how I can develop my personal qualities and skills.

I would like to express my appreciation to my Associate Supervisor, Dr Karina M. Gomez, who provided guidance and recommendations, from the beginning of my candidature to the last milestone.

I would like to acknowledge my employer, PT Telekomunikasi Indonesia, Tbk, who provided me with the opportunity and financial support to pursue a PhD.

I would like to acknowledge my family, my wife Shanty, my sons Deo and Leo, and my daughter Dita. Thank you for your love and support during our time together here in Melbourne.

I would like to appreciate the many anonymous reviewers of my peer-reviewed published journal and conference proceedings. Their insightful comments helped me to improve my publications.

Finally, I would like to thank the staff and students in the School of Engineering at RMIT University, and all of my friends here in Melbourne for their help and support.

Abstract

Multi-domain Software Defined Networking (SDN) is the extension of the SDN paradigm to multi-domain networking and the interconnection of different administrative domains. By utilising SDN in the core telecommunication networks, benefits are found including improved traffic flow control, fast route updates and the potential for routing centralisation across domains. The Border Gateway Protocol (BGP) was designed three decades ago, and efforts to redesign interdomain routing that would include a replacement or upgrade to the existing BGP have yet to be realised. For the near real-time flow control provided by SDN, the domain boundary presents a challenge that is difficult to overcome when utilising existing protocols. Replacing the existing gateway mechanism, that provides routing updates between the different administrative domains, with a multi-domain centralised SDN-based solution may not be supported by the network operators, so it is a challenge to identify an approach that works within this constraint.

In this research, BGP was studied and selected as the inter-domain SDN communication protocol, and it was used as the baseline protocol for a novel framework for automatic multi-domain SDN provisioning. The framework utilises the BGP UPDATE message with Communities and Extended Communities as the attributes for message exchange. A new application called Inter-Domain Provisioning of Routing Policy in ONOS (INDOPRONOS), for the framework implementation, was developed and tested. This application was built as an ONOS controller application, which collaborated with the existing ONOS SDN-IP application.

The framework implementation was tested to verify the information exchange mechanism between domains, and it successfully carried out the provisioning actions that are triggered by that exchanged information. The test results show that the framework was successfully verified. The information carried inside the two attributes can successfully

be transferred between domains, and it can be used to trigger INDOPRONOS to create and install new alternative intents to override the default intents of the ONOS controller. The intents installed by INDOPRONOS immediately change the route of the existing connection, which demonstrated that the correct request sent from the other domain, can carry out a modification in network settings inside a domain.

Finally, the framework was tested using a bandwidth on demand use case. In this use case, a customer network administrator can immediately change the network service bandwidth which was provided by the service provider, without any intervention from the service provider administrator, based on an agreed-predefined configuration setting. This ability will provide benefits for both customer and service provider, in terms of customer satisfaction and network operations efficiency.

Table of Contents

Declaration ii
Acknowledgements iv
Abstract v
Table of Contents vii
List of Figures x
List of Tables xii
List of Acronyms and Abbreviations xiii
Chapter 1 Introduction 1
1.1. Background and Motivation 2
1.2. Research Problems 4
1.3. Research Objectives 6
1.4. Research Contributions 6
1.5. Publications 8
1.6. Thesis Structure 9
Chapter 2 Literature Review 11
2.1. Overview 12
2.2. Software Defined Networking Overview 12
2.2.1. From Programmable Network to SDN 13
2.2.2. SDN Architecture and Interfaces 15
2.2.3. OpenFlow 19
2.3. SDN Controller Deployment 21
2.3.1. SDN Controller Design Challenges 22
2.3.2. Centralised and Distributed SDN Controllers 25
2.4. SDN in the Wide Area Network 30
2.4.1. Multi-domain SDN Architecture 30
2.4.2. Inter-Domain SDN Communication 33
2.5. Border Gateway Protocol 35
2.5.1. BGP Overview 35
2.5.2. BGP Operation 37
2.5.3. Path Attributes 40
2.6. Conclusion 41
Chapter 3 Inter-Domain SDN Communication Protocol 43
3.1. Overview 44

3.2. Requirements for Inter-Domain SDN Communication 44
3.2.1. Primary Ability of SDN Inter-Domain Communication 44
3.2.2. Message Exchange between SDN domains 45
3.3. Inter-Domain SDN Communication Utilising BGP 46
3.4. Implementation and Test Scenario 48
3.4.1. Test Scenario 49
3.5. Results 50
3.6. Conclusion 53
Chapter 4 Information Exchange Mechanism for Multi-domain SDN Provisioning . 54
4.1. Overview 55
4.2. BGP Message for Multi-domain SDN Provisioning 55
4.2.1. Selecting Path Attributes 56
4.2.2. Communities and Extended Communities Attributes for Multi-domain SDN Provisioning 58
4.3. Inter-Domain SDN Message Exchange Implementation 61
4.3.1. ONOS SDN-IP Structure 61
4.3.2. Modification of ONOS BGP Routing Modules 63
4.4. Verification of Message Exchange Mechanism 66
4.4.1. Verification Topology 66
4.4.1. ONOS Verification CLI 68
4.5. Results 68
4.6. Conclusion 71
Chapter 5 Multi-domain SDN Automatic Provisioning Framework 72
5.1. Overview 73
5.2. Multi-domain SDN Provisioning Challenges 73
5.2.1. Automated Provisioning 73
5.2.2. SDN Provisioning 74
5.3. Design of a Multi-Domain SDN Provisioning Framework 76
5.3.1. Proposed Provisioning Procedures 77
5.3.2. Proposed Framework 79
5.4. Framework Implementation in ONOS Controller 80
5.4.1. INDOPRONOS Software Modules 80
5.4.2. INDOPRONOS Algorithm 83
5.4.3. Implementation Test Bed 84

5.5. Test Scenarios 86
5.5.1. Scenario A: Framework Verification 87
5.5.2. Scenario B: Bandwidth on Demand Use Case 88
5.5.3. Scenario C: Non-Bandwidth on Demand Subscriber 91
5.6. Results 93
5.6.1. Scenario A Results 93
5.6.2. Scenario B Results 97
5.6.3. Scenario C Results 101
5.7. Conclusion 104
Chapter 6 CONCLUSION AND FUTURE WORK 106
6.1. Conclusions 107
6.2. Future Work 108
BIBLIOGRAPHY 110
Appendix 1 – BgpConstants Module 120
Appendix 2 – BgpUpdate Module 131
Appendix 3 – BgpRouteEntry Module 162
Appendix 4 – BgpRoutesListCommand Module 174
Appendix 5 – Indopronos Module 181
Appendix 6 – IndopronosConnectivityManager Module 183

List of Figures

Figure 1-1 Centralized and Federated Inter-Domain Communication..... 3
Figure 1-2 Multi-domain Provisioning Approach 7
Figure 2-1 SDN Architecture [28] 16
Figure 2-2 OpenFlow Components [30] 21
Figure 2-3 Implementation Approaches for Multi-domain SDN. 32
Figure 2-4 BGP Network Examples..... 36
Figure 2-5 BGP FSM and Messages. 38
Figure 3-1 BGP Sessions Establishment Process in Multi-Domain SDN. 47
Figure 3-2 Test Topology. 49
Figure 3-3 Flow Setup Latencies..... 51
Figure 3-4 End to End Delay Variation. 52
Figure 4-1 Data Formats of Selected Patch Attributes..... 59
Figure 4-2 Information Flow Diagram of Exchanged Message in Multi-domain SDN..... 61
Figure 4-3 SDN-IP Architecture 62
Figure 4-4 SDN-IP Software Modules..... 64
Figure 4-5 Software Modules Modifications..... 65
Figure 4-6 Verification Topology 67
Figure 4-7 BGP UPDATE Message Received by BGP Speaker..... 69
Figure 4-8 Routing Table Entry in BGP Speaker. 70
Figure 4-9 Displaying Community and Extended Community Attributes in ONOS..... 70
Figure 5-1 Example of Matching Process in SDN. 75
Figure 5-2 Multi-domain SDN Provisioning Procedure 78
Figure 5-3 Multi-domain SDN Provisioning Framework 80
Figure 5-4 INDOPRONOS Application Algorithm..... 82
Figure 5-5 Test Bed Implementation using ONOS..... 86
Figure 5-6 Topology for Scenario A. 88
Figure 5-7 Topology for Scenario B. 90
Figure 5-8 Topolgy for Scenario C. 92
Figure 5-9 Verification of the Communities Values Exchanged between Domains..... 94
Figure 5-10 Verification of New Installed Intents. 96
Figure 5-11 RTT Results from Network A to Network B..... 97
Figure 5-12 Throughput Bandwidth Measured from Network A to Network B. 99

Figure 5-13 Average Packet Loss 100
Figure 5-14 Troughput Bandwidth towards Network B 102
Figure 5-15 Jitter vs Packet Loss 103

List of Tables

Table 2-1 Summary of Controller Platforms [42] 25

Table 2-2 Summary of Multi-domain SDN Activities [42] 31

Table 2-3 Summary of Current SDN Inter-Domain Communication 34

Table 4-1 Selection of BGP Path Attributes for Multi-domain SDN Provisioning 57

Table 4-2 Communities Attribute Values 60

Table 4-3 Extended Communities Attribute Values 60

Table 4-4 BGP Software Modules Modifications 65

Table 4-5 Verification Devices 67

Table 5-1 SLA Example 84

Table 5-2 Test Bed Components 86

Table 5-3 Links Specifications 89

List of Acronyms and Abbreviations

API	Application Programming Interface
BGP	Border Gateway Protocol
CDN	Content Delivery Network
DISCO	Distributed Multi-domain SDN Controllers
DoS	Denial of Service
ForCES	Forwarding and Control Element Separation
IP	Internet Protocol
ISP	Internet Service Provider
ODL	OpenDaylight
ONF	Open Network Forum
ONOS	Open Network Operating System
PB	Petabytes
PCE	Path Computation Element
RCP	Routing Control Platform
REST	Representational State Transfer
SDN	Software Defined Networking
SNMP	Simple Network Management Protocol
SP	Service Provider
UDP	User Datagram Protocol
WAN	Wide Area Network
ZB	Zettabytes

Chapter 1 Introduction

1.1. Background and Motivation

The growth of digital networking has continued over the past decades and now telecommunications and networking are an essential part of the systems that impact on every aspect of our daily lives. The complexity of digital communications also continues unabated and today we're faced with massive growth in the various communication technologies used to support communications between people, systems and machines. Globally, the Internet Protocol (IP) traffic is predicted to increase almost threefold between 2016 until 2021, i.e. from 1.2 ZB (Zettabytes) in 2016, to 3.3 ZB in 2021 [1]. Content Delivery Networks (CDN) have become a vital part of the strategy to support real-time and content intensive applications, such as video streaming and enterprise video conferencing. Globally, 70 % of all internet traffic will cross CDNs by 2021, with the total traffic amounting to 165,651 PB (Petabytes) per month. This phenomenon has led to the need for high capacity, reliable and yet cost-effective networks that can carry increasing traffic volumes, with dynamic and distinct applications for each entity.

Telecommunication networks consist of the core, distribution and access networks and combinations of the network types, including the relationship between separate administrative domains are known as Wide Area Networks (WANs). As a widely deployed network, global Service Providers (SPs) connect the numerous Autonomous Systems (AS) or domains to form the WAN, which in turn is often termed the Internet with the inclusion of the services, systems and applications that operate and are provided over the WAN. Interconnections between domains rely on inter-domain communication protocols and currently the Border Gateway Protocols (BGP) are the mainstay of this interaction. BGP has been used for more than 30 years, it remains very similar to what it was when it was first implemented and today there is a need to improve its performance [2, 3].

In this context, Software Defined Networking (SDN) [4, 5] separates the network control and management plane from the data plane, thereby providing a programmatic, flexible and dynamic network management and operations environment. With SDN, network operators can configure the entire network from a vantage point by using a standardised Application Programming Interface (API) and protocols. SDN transforms the network devices into packet forwarding devices (data plane) [5]. The network devices can be controlled by an interface called the Southbound interface, using the OpenFlow protocol [6]. SDN offers benefits through its approach [5, 7], i.e. enhancing configuration, improving performance, and encouraging innovation, which can be offered to the multi-domain network environment.

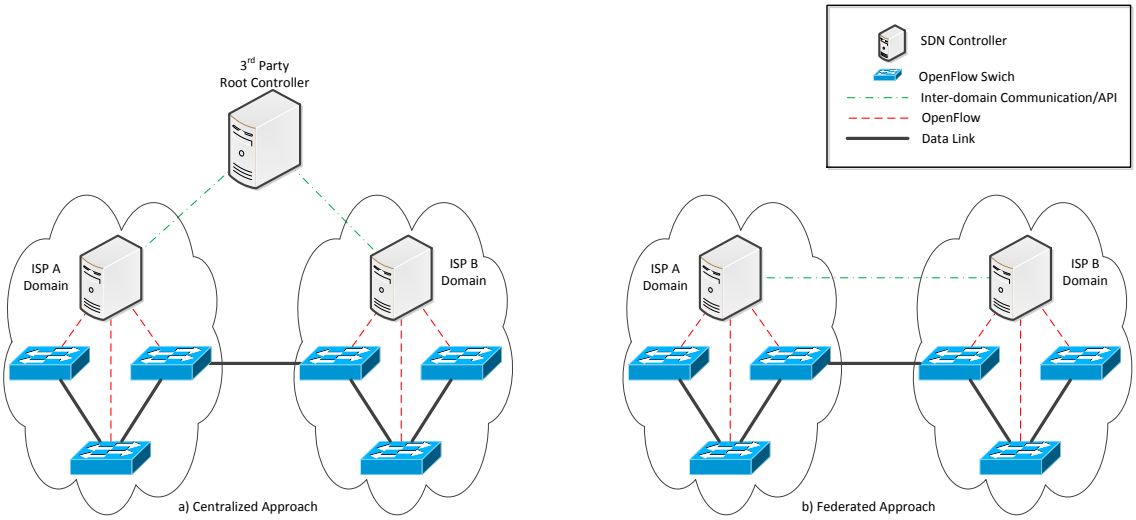


Figure 1-1 Centralized and Federated Inter-Domain Communication

The two general approaches for SDN based inter-domain communication, i.e. centralized and federated, are shown in Figure 1-1 [8]. The centralised framework introduced in [9], used a routing outsourcing concept [10] that relies on a single SDN domain overlaying the Service Provider (SP) networks. This concept requires the SP domains to disclose network information and configuration to the third party, which is not preferred in practice, due to typical administrative policies and security. The federated framework relies on communications between SDN domains using an inter-domain communication

protocol. This concept is more likely to be adopted and there are several research testbeds running globally [11-13]. However, this approach would require significant changes to the current inter-domain practices and implementations.

A SDN implementation in multi-domain networking provides an opportunity to introduce programmability into this key part of the network. The improved network flexibility could help SPs to meet their customer demands for dynamic traffic, which is currently solved using overprovisioning [14, 15]. Overprovisioning is an approach where the network is configured to operate at about 20-40% of capacity [15, 16], therefore it should be able to cope with the customer traffic surges, which is expensive and not efficient.

1.2. Research Problems

The federated multi-domain SDN approach relies on the communication protocols used between SDN domains. Unfortunately, although they are defined in the SDN architecture, the East-Westbound interface which facilitates controller-to-controller communication has not been standardised [17, 18]. Another challenge is how multi-domain SDN can improve the efficiency of WAN operations, especially to reduce the need for overprovisioning. Also, and probably most importantly, there is a likelihood that domain administrators will wish to continue to limit the visibility within their domain to other domain administrators, something that BGP does inherently now. How this would be possible within a federated multi-domain SDN approach is yet to be determined.

The vertical architecture that is currently the focus of SDN development promises to provide excellent design outcomes and performance improvements for single domain networks [9]. However, from an implementation perspective for a multi-domain environment, the concept faces difficulties, as each domain entity needs to communicate to a root controller and there would be indeterminate issues surrounding control and

administration of this root controller. There are at least two other problems, i.e. in the customer-provider network implementation models, each entity prefers not to disclose any network related information to its peer, outside the network prefixes (which it advertises to neighbouring peer). This model also requires a new business entity which acts as an outsourcing function with new protocols to be used between local domain controllers with the root controller. The addition of the new business entity could result in a lack of backwards compatibility with conventional IP networks. For a horizontal architecture, the resistance between domain entities would be less when compared to the vertical, where each domain will maintain their privileges and only exchange the information needed to route packets to other domains. However, this architecture could face the same problems related to backward compatibility if a new protocol is used.

In the operation of a WAN, the impact of overprovisioning could lead to inefficiency. And as described in [19], overprovisioning could help to maintain acceptable network service performance, but it limits network performance due to the reservation of unused resources and higher operational cost. The introduction of multi-domain SDN in the WAN should reduce the effect of this legacy approach to network resource demand. Currently, no multi-domain SDN framework has been proposed to facilitate flexible provisioning between different domains. This provides the motivation for this research.

Therefore, the following research problems were identified as research questions:

a. How is the performance of current inter-domain SDN communication techniques used in the multi-domain SDN architecture?

b. What are the requirements and the design drivers of flexible provisioning using multi-domain SDN architecture?

c. How can SDN controllers in different domains exchange information with peers?

d. What are the specifications and the step-by-step process that can be used for message exchange between controllers which support Automated Provisioning in Multi-Domain SDN?

1.3. Research Objectives

The research objectives for the research presented in this thesis are:

a. To develop an understanding of multi-domain SDN frameworks and current architectures to investigate their advantages and drawbacks.

b. To evaluate the inter-domain SDN communication techniques in the federated approach that are used to exchange messages between controllers in support of multi-domain SDN automatic provisioning.

c. To propose new techniques and a specification for the exchange of information between SDN domains and to design a framework for an automatic multi-domain SDN provisioning framework.

d. To develop a prototype to demonstrate the information exchange and the process of automated provisioning in multi-domain SDN.

1.4. Research Contributions

The research carried out successfully met the research aims and research objectives. The contributions of this research can be described as follows:

a. A comprehensive review of SDN with its challenges and controller designs has been presented with the classification controller design to cope with scalability challenges.

b. An extensive review on current border gateway protocols that could used for inter-domain SDN communication is presented, which provided the motivation to explore

the use of BGP as preferred baseline protocol for inter-domain SDN communication.

c. Procedures for inter-domain SDN communication utilising BGP were discussed and proposed. The comparison study to compare multi-domain SDN using BGP with legacy networking was discussed and as a result the utilisation of the existing BGP as an inter-domain SDN communication protocol was considered.

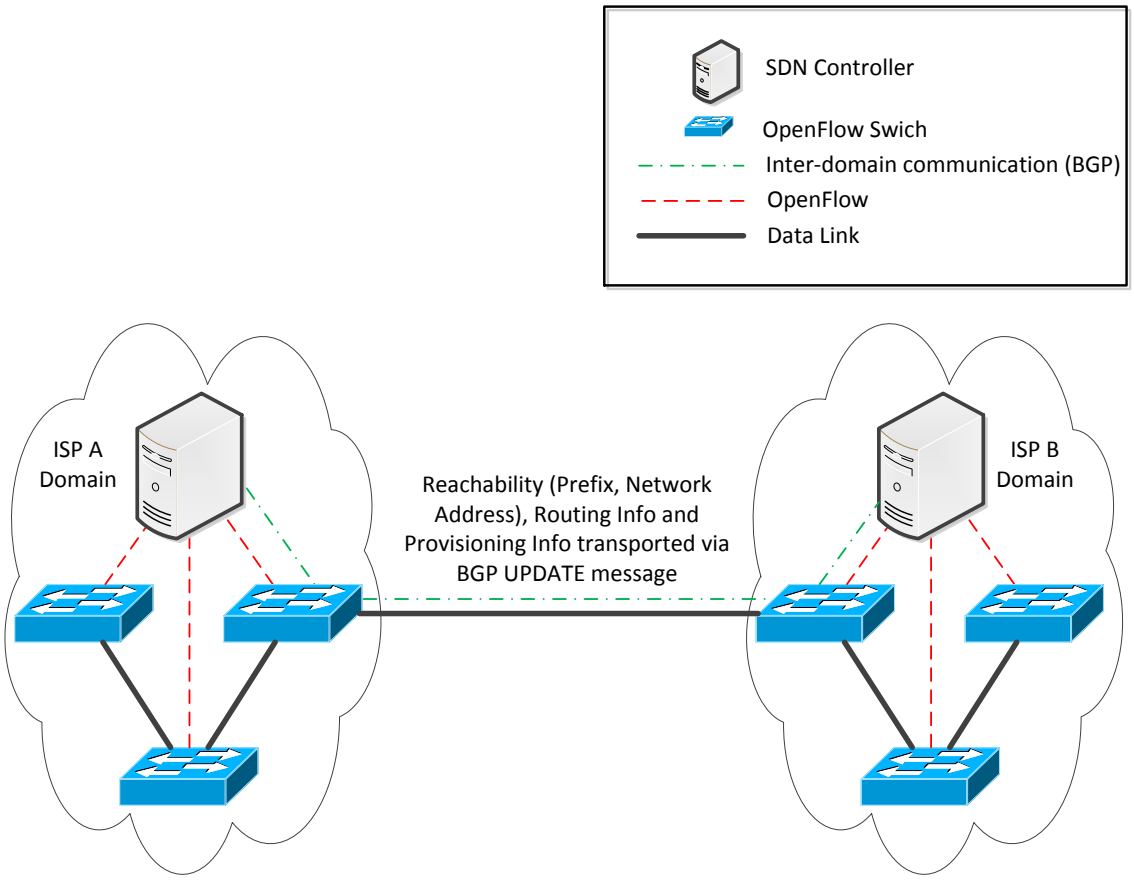


Figure 1-2 Multi-domain Provisioning Approach

d. This research identified a new and innovative approach to using the BGP UPDATE message as the information exchange mechanism in a multi-domain SDN environment. The proposal was implemented using an ONOS controller and successfully verified. This mechanism was shown to be successful when used in the automatic provisioning multi-domain SDN framework.

e. The framework was implemented using an ONOS controller with a SDN-IP

application and the new Inter-Domain Provisioning of Routing Policy in ONOS (INDOPRONOS) application. INDOPRONOS is a novel application developed in ONOS to conduct the provisioning process between SDN controllers.

1.5. Publications

Peer-reviewed publications either published or submitted for publication during the research program.

Journal

a. F. X. A. Wibowo, M. A. Gregory, K. Ahmed, and K. M. Gomez, "Multi-domain Software Defined Networking: Research status and challenges," Journal of Network and Computer Applications, vol. 87, pp. 32-45, 2017/06/01/ 2017. (status: published).

b. F. X. A. Wibowo and M. A. Gregory, "Bandwidth Update Impact in Multi-Domain Software Defined Networking," International Journal of Information, Communication Technology and Applications, vol. 3, pp. 1-14, 2017. (status: published).

c. F. X. A. Wibowo and M. A. Gregory, "Inter-Domain Software Defined Network Provisioning Framework," submitted to Journal of Network and Computer Applications, August 2018. (status: under review)

Conference Paper

d. F. X. A. Wibowo and M. A. Gregory, "Software Defined Networking properties in multi-domain networks," in 2016 26th International Telecommunication Networks and Applications Conference (ITNAC), 2016, 2016, pp. 95-100.

e. F. X. A. Wibowo and M. A. Gregory, "Updating guaranteed bandwidth in multi-

domain Software Defined Networks," in 2017 27th International Telecommunication Networks and Applications Conference (ITNAC), 2017, pp. 1-6.

f. F. X. A. Wibowo and M. A. Gregory, "Multi-domain Software Defined Networking," in 2018 28th International Telecommunication Networks and Applications Conference (ITNAC), 2018 (accepted for presentation)

1.6. Thesis Structure

The thesis chapters presenting the work carried out are summarised to provide a guide to the thesis composition.

a. Chapter 1 presents the introduction to the research, which consists of the background and includes research aims, objectives and publications. The introduction presents the motivating factors behind the research, why it is crucial and adds to the body of knowledge and briefly outlines the approach taken.

b. Chapter 2 presents a literature review describes the supporting knowledge for this research and provides the current research status within the research area, especially on SDN and multi-domain SDN.

c. Chapter 3 presents the study of BGP as an inter-domain SDN communication protocol. The protocol is tested using a test bed to evaluate its properties in the multi-domain SDN environment.

d. Chapter 4 presents the study and implementation of information exchange mechanism in multi-domain SDN. BGP UPDATE message is studied and tested inside the test bed to verify the proposed mechanism.

e. Chapter 5 presents the proposed framework of automatic provisioning in multi-domain SDN. Three scenarios are tested and evaluated inside a test bed to verify the

proposed framework.

f. Chapter 6 presents a summary of the research contributions and provides suggestions for future research.

Chapter 2 Literature Review

2.1. Overview

The literature review provided in this chapter identifies current research and provides knowledge of the fundamental technologies and systems encountered during the research project. In this chapter, a summary of SDN is provided to highlight the origin of SDN, its architecture and protocol. Scalability as one of the SDN challenges will be discussed, along with the comparison of how single and multiple controllers might be deployed and used to meet that challenge. Special attention is given to the multi-controller approach for WANs and multi-domain SDN research. Finally, selected inter-domain SDN communication proposals are described.

2.2. Software Defined Networking Overview

In the past, telecommunication networks were generally built using tightly coupled software and hardware, which made the networks inflexible and hard to facilitate new or updated service deployments in a timely manner. The use of vendor-specific network devices and systems often led to the organisation's systems becoming tailored to match the complexities of the vendor equipment and systems. The idea of programmable networking was introduced to overcome some of the legacy networking challenges and over time this approach became known as SDN. SDN has evolved over the past decade to provide a more flexible and dynamic networking architecture that incorporates improved support for management and network services. The SDN approach decouples the management and control of traffic flows from the underlying transmission infrastructure and systems that is used to forward traffic. To facilitate separation of the control and data planes a standardised and open protocol, known as OpenFlow, was developed to facilitate control traffic transfer between the management and control systems, known as controllers, and the network devices, e.g. a switch, that forward traffic.

2.2.1. From Programmable Network to SDN

The development of SDN originated from the early work on programmable networking and the separation of control logic from the data transfer mechanism. There were two concepts of programmable networking, i.e. active networks and open signalling. However, the idea to decouple the control logic from the data transfer mechanism emerged later as a new architecture, to reduce the complexity of the distributed computations.

The active network concept was introduced in the mid-1990s in an endeavour to control a network in real-time. Active networking introduced a method that permits packets flowing through the network to carry instructions to be executed at network nodes. The code carried within the packets alters the network operation either temporarily for an individual packet or a stream of packets. In this approach, the network devices become a dynamically programmable environment that can be altered using the code carried by the packets, which differs from the rigidity of traditional networking [20].

Implementations of active networks include SwitchWare [21] and conventional computer routing suites such as Click, XORP, Quagga, and BIRD [7]. With the active networking implementations, the operations and behaviour of the network can be modified dynamically. Although the active networking approach offered a new paradigm by providing a more dynamic environment, there was only minor development of the control plane. The active networking approach placed the intelligence at the endpoints (which can be inferred to be computers and servers acting as smart devices) while utilising enhanced switches and routers to execute and carry out limited tasks based on the instructions carried within the packets traversing the network. Therefore, in active networking, packets are entities that can determine or control how nodes manage packets and streams.

The second programmable network concept was open signalling (OPENSIG), which was proposed by the telecommunication network community [22]. This community suggested the idea of separating the communication hardware and control software by accessing the network hardware using an open and programmable network interface. That idea was very challenging due to the vertically integrated network device architecture, e.g. routers and switches. OPENSIG suggested that the well-defined programmable network interfaces could lead to a distributed programming environment which provides open access to switches and routers and facilitates the entry into the telecommunication software market of third-party software providers.

In parallel with the programmable networking effort, innovations emerged on the separation of the control and data planes, which aimed to develop open and standardised interfaces along with a logically centralised network control model. One of the innovations was the Forwarding and Control Element Separation (ForCES). FORCES tried to improve networking with the use of distributed packet processing network elements [23]. Other innovations including, Routing Control Platform (RCP), Path Computation Element (PCE), and Intelligent Route Service Control Protocol (IRSCP), were focused on improving network operation, however, the need for coexistence and backward compatibility prevented them from being deployed immediately [17].

In 2004, the 4D project [24] introduced a clean slate design, which stressed the separation of the routing decision logic and the protocols used to pass messages between network elements. The proposal used a global view of the network with a decision plane, serviced by a dissemination and discovery plane that would be used to control traffic forwarding. Later, these ideas inspired advanced research into NOX, the first SDN controller, which in the context of an OpenFlow-enabled network is also known as an operating system for networks.

The IETF Network Configuration Working Group proposed NETCONF in 2006, as a management protocol for network device configuration. The protocol carried messages to network devices through an exposed API that supported extensible configuration data. NETCONF had efficiency, effectiveness, and security advantages when compared to the Simple Network Management Protocol (SNMP), a popular network management protocol, especially when managing complex and diverse network environments [25]. Simple Network Management Protocol (SNMP) and NETCONF are both useful management tools that can be used in parallel on hybrid switches that support programmable networking.

Other protocols that preceded OpenFlow were the SANE [26] and Ethane [27] projects completed in 2006. They defined a new architecture for enterprise networks that utilised a centralised controller to manage policy and security. Identity-based access control was the notable feature. Relatively the same as SDN, Ethane consists of two components, i.e. a controller and Ethane switch. The controller decides whether a packet should be forwarded, while the switch connects to the controller via a secure channel and holds a flow table, which provided the foundation for SDN.

2.2.2. SDN Architecture and Interfaces

The Open Networking Forum (ONF) defined SDN as an emerging networking architecture where network control is decoupled from forwarding and is directly programmable [28]. The control function is migrated from formerly tightly bound devices in each network into manageable computing devices. This migration enables the underlying infrastructure to be abstracted for applications and network services. Later, the network could be treated as a logical or virtual entity.

There are two primary SDN characteristics, as depicted from its architecture shown in

Figure 2-1, including decoupling of the control and data planes, and control plane programmability. Both have previously been the focus of extensive research and recent improvements in the reliability, capacity, and capability of global networks that have enabled the control plane programmability concept to move forward. SDN encompasses the separation of control and data planes in the network's architectural design, which means that network control is to be carried out utilising separate channels between device control management ports that utilise different addresses to that used for the data plane. The network intelligence is taken out of the switching devices, thereby leaving the switching devices as general forwarding devices.

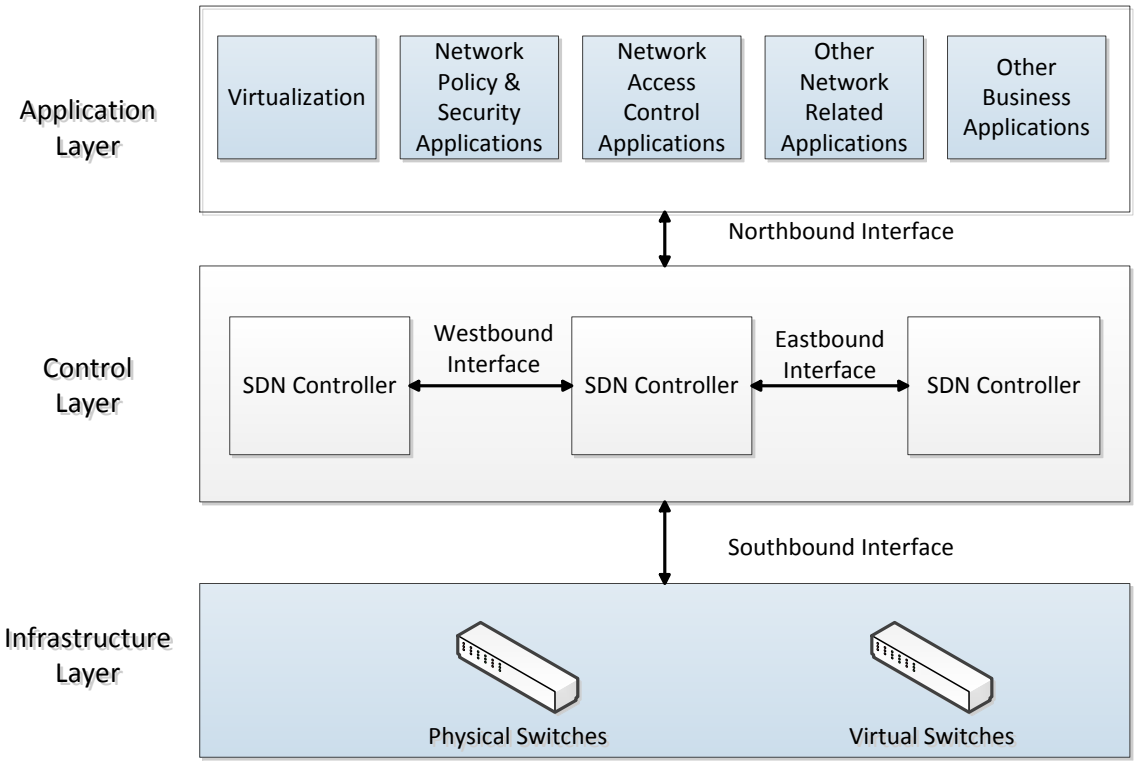


Figure 2-1 SDN Architecture [28]

There are three functional layers in the SDN architecture, i.e. Application Layer, Control Layer, and Infrastructure Layer. Each layer provides functionality and communicates with the other layers via a specific interface [17, 29]. The descriptions of SDN layers are presented below:

a. Application Layer. The Application Layer consists of network services and applications that can be abstracted using the dynamic modular structure of the Application Layer. Examples of network services and applications include management systems, monitoring, security and flow control related network services. The network abstraction utilises an API to provide consistency and standardisation of the interface. Through this API, SDN services and applications can access network status information reported from forwarding devices. SDN services and applications can also use this API to transfer flow rules to forwarding devices through the lower layers.

b. Control Layer. The Control Layer consists of a set of software-based SDN controllers that provide control functionality through open API-based interfaces that facilitate the control and management of traffic forwarding. The controllers incorporate three communication interfaces, i.e. southbound, northbound and eastbound/westbound. The control plane acts as an intermediary layer between the application and data planes. The controller provides a programmatic interface that can be accessed by network services and applications in the Application Layer and used to implement management and control tasks. This abstraction assumes that the control is centralised, and applications are written assuming that the network is a single interconnected system, which enables the SDN model to be implemented for a broad range of scenarios, such as centralised, hybrid or distributed and for heterogeneous network technologies (wireless or wired). The controller implementation design significantly affects the overall performance of the network. Several challenges must be overcome to achieve network performance that is at or above the network performance of the previous legacy network.

c. Infrastructure Layer. The Infrastructure Layer is the lowest layer in the SDN

architecture. Forwarding elements are the main components in this layer, which includes the physical and virtual routers and switches. These devices are accessible via an open interface and carry out packet routing, switching and forwarding. The control connections to the network devices utilise separate secure channels to that used for user data flows.

SDN architecture employs three specific interfaces. These interfaces enable the interactions between and within SDN layers. The SDN interfaces include:

a. Northbound Interface. The Northbound interface is used to connect network services and applications found in the Application Layer to the controllers in the Control Layer. The Northbound interface consists of one or more API providing a programmability capability that is used to manage network traffic flows dynamically. It is more of a software API than a protocol-based interface to take advantage of the innovative programmability paradigm. The ONF suggest a definition that the different levels of abstractions are latitudes and the various use cases are longitudes, but this characterisation is yet to be finalised. The ONF approach suggests that more than a single northbound interface standard can provide increased flexibility to serve different use cases and environments. Representational State Transfer (REST) is one of the proposed APIs as the programmable interface for business applications to the controllers.

b. Eastbound/Westbound Interface. The Eastbound/Westbound interface supports control traffic between controllers using a yet to be standardised communication protocol. It is identified as enabling communication between groups or federations of controllers to synchronise states for high reliability and resiliency. The Eastbound/Westbound interface concept aims to partially meet the SDN scalability

and reliability challenge. The Eastbound/Westbound interface protocol manages communications between multiple controllers. There are two possible use cases for this interface. The first use case is an interconnecting interface between conventional IP networks with SDN networks. As there are no standards defined for this interface, its implementation depends on the technology used by the underlying network. An example of this use case is to connect the SDN domain with a legacy domain using a legacy routing protocol to react to message requests, e.g., Path Computation Element (PCE) protocol and Multi-Protocol Label Switching (MPLS). The second use case is to use the interface as an information conduit for admission and authentication, between the SDN control planes of different SDN domains. The multi-domain connectivity challenge needs to be overcome to facilitate a global network view and influence the routing decisions of controllers on domain boundaries. A solution would allow a seamless setup of network flows across heterogeneous SDN domains. Conventional border protocols like BGP could be used or modified to support interconnection of remote SDN domains.

c. Southbound Interface. The Southbound interface enables the controller to communicate with the forwarding elements found in the Infrastructure Layer using a standardised protocol. OpenFlow, a protocol maintained by the ONF, is described as a fundamental element in the development of SDN solutions and is seen to be a promising implementation of a southbound interface protocol. A standardised protocol permits multi-vendor network device SDN implementations.

2.2.3. OpenFlow

OpenFlow [6] is an open source communications protocol that is used to transport messages from controllers to network devices via the Southbound interface. OpenFlow provides software-based access to the switch and router flow tables to enable dynamic

network traffic management. Manual or automated control systems can be used depending on the specific network management operations scenario. Also, the OpenFlow protocol provides a core set of management tools which can be utilised to manage several features, such as topology changes and packet filtering.

An OpenFlow-enabled switch contains flow and group tables that include several entries, depending on the network device [30]. The OpenFlow messages, exchanged between the switch and the controller over a secure channel, can manipulate the flow entries as desired. By maintaining a flow table, the switch can make forwarding decisions for incoming packets using a simple look-up on the entries of its flow table [31]. The switch carries out an exact match check on selected fields of the incoming packets. The switch examines the flow table entries to find a matching entry for incoming packets. The flow tables are numbered sequentially, starting at 0. The packet processing pipeline starts at the first flow table and if a match is not found it moves on to the next flow table and so on until a match is found, or the end of the last flow table is reached. If the specific packet fields match a flow entry, the corresponding instruction set is executed. Instructions related to each flow entry describe packet forwarding, packet modification, group table processing, and pipeline processing.

Pipeline-processing instructions enable the packet fields to be matched to a flow entry in one table and based on the flow entry instructions be sent to associated tables for additional matching and processing, which can result in an aggregation of actions that are to occur to the packet before transmission. The aggregated information (metadata) can be communicated between flow tables. Flow entries may also forward to a physical or virtual port.

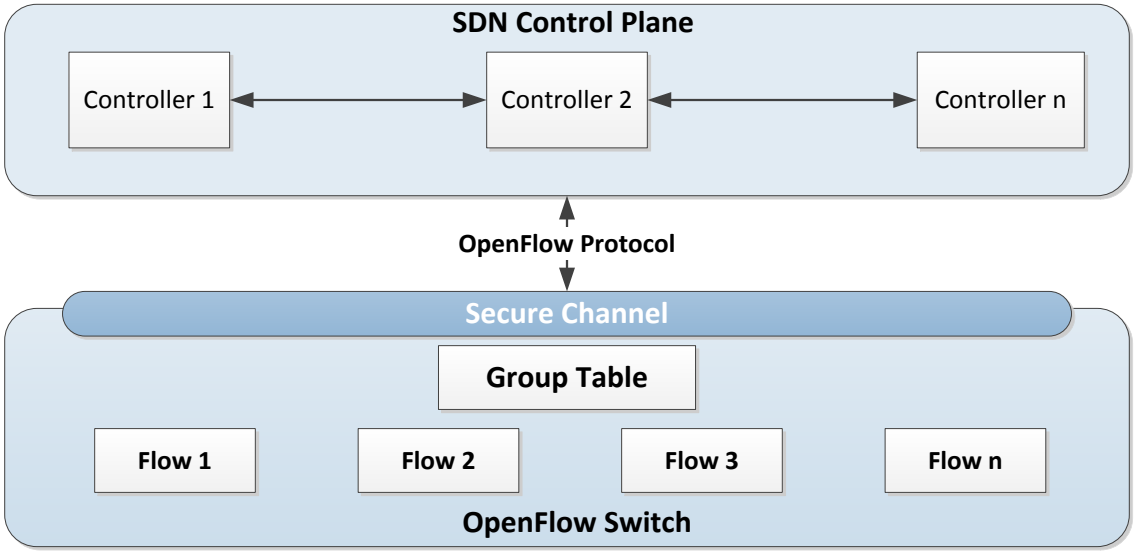


Figure 2-2 OpenFlow Components [30]

Flow entries may link to a group table entry, which specifies additional processing. A group table, which consists of group entries, offers additional forwarding methods (multicast, broadcast, fast reroute, link aggregation). A group entry contains a group identifier, a group type, counters, and a list of action buckets, which contain a set of actions to be executed and associated parameters. Groups also support multiple flows to be forwarded to a single identifier, e.g., IP forwarding to a common next hop [31]. Sometimes, a packet might not match a flow entry in any of the flow tables; this is called a 'table miss'. The action taken in the case of a miss depends on the table configuration. By default, selected packet fields are sent to the controller over the secure channel. Another option is to drop the packet.

2.3. SDN Controller Deployment

SDN offers flexibility when managing network traffic flows. The global digital network is increasing in its size and complexity and individual network domains are connected to become a massive and vast network. With its advantages, SDN could provide a more efficient way of managing the traffic flows and global networks. The SDN controller deployment that has been selected to cope with scalability challenges will be described,

and as SDN supports both centralised and distributed controller models, each model will also be discussed.

2.3.1. SDN Controller Design Challenges

Controller design requires substantial effort if the controller is to provide flexible interfaces for network services and applications and a verified OpenFlow interface. It is more than just a matter of designing and implementing the interfaces, matching the interfaces with the network and applications, programming languages, and software architecture in the controller; the design also relates to the performance of SDN-enabled networks. Selected challenges are described below:

a. Scalability. An initial concern that arises when offloading control from the switching hardware is the scalability and performance of the network controller(s). SDN's centralised control methodology naturally faces scalability issues. The controller is the most important element in the SDN architecture, and a single controller elucidation may result in a single point of failure and performance bottleneck problems in a wide area SDN [8, 20]. The entire network will break down if the controller fails. On the other hand, no matter where we place the controller, it will be farther away from some of the switches under its control. These switches will experience higher flow setup latency [32]. A single controller solution is not suitable for wide-area SDN and some enterprise networks. Other concerns on the scalability of SDN in vast networks are the aggregating and disseminating of a massive amount of information, both from and throughout the network [33]. Those processes need to be carried out in real-time, which make things worse.

b. Placement and Reliability. In the wide area SDN implementation, controller(s) location may impact on the overall network performance. Whether the SDN consists

of single or multiple controllers, the placement of the controller(s) will have an impact on the performance and the cost of the network. Research is ongoing into architecture, location and the number of controllers concerning the average and worst-case latencies of control plane [34] and other metrics, e.g. latency in case of failure and inter-controller latency [35]. The research shows that latency drives the overall behaviour of the network, and bandwidth for the control traffic affects the number of flows that the controller can process. Network modelling is used to identify controller locations that enhance reliability and limit control message latency [36]. Alternative approaches try to solve the controller(s) placement issue, optimise the reliability of the control network and identify several placement algorithms and strategies along with metrics to characterise the reliability of SDN control networks [35, 37].

c. Availability. SDN in the vast global network could suffer from various failures. The potential failures include controller/switch failure and link failure. SDN controllers can be overloaded due to an enormous number of requests from linked network devices. An SDN-enabled switch could be confronted with a failure if any of its sub-components does not function correctly. Controller failure could cause the traffic routing/forwarding functions of linked network devices to become limited. Network link failure can occur due to the link itself failing or the terminating devices failing. The cause of this failure could be from network connectivity and hardware problems, or software problems in any network device that generates link down notifications falsely [38]. In SDN networks, the overall design, including placement and selection of network devices such as the controllers and switches, should be robust, and this should be tested for a range of anticipated scenarios. An approach that improves SDN robustness is to use a runtime system that automates failure recovery by spawning a new controller instance and replaying inputs observed by the old controller. The

controller can install static rules on the switches to verify topology connectivity and locate link failures based on those rules. Another approach is to try to improve recovery time by the frequent issuing and receipt of monitoring messages, but this may place a significant load on the control plane [20] [7]. In multi-controller SDN, a load balancing mechanism based on a load informing strategy is proposed to balance the load among controllers dynamically [39].

d. Security. SDN controllers may suffer from a range of security problems, which can reduce network performance. The attacks might affect performance due to the lack of controller scalability in the event of a denial of service (DoS) attack. The impact could become severe in the extensive network for single controller or multiple controller scenarios. The attacks can target the forwarding layer, control layer, and application layer, and their types can be discussed as follows [40, 41]. On the application layer, the attacks could occur through unauthenticated applications and policy enforcement. DoS is an attack that might target the forwarding and control layers. DoS could be caused by massive traffic flows that flood the switches which subsequently flood controllers with traffic route requests. The result is a slowing down of the network and possibly device collapse. Another type of attack is the compromised controller attack, which happens when the attacker gains access to the controller and utilises this access to alter or deny traffic routing. Data leakage and flow rule modification are the impacts of attacks on forwarding layer input buffers, which can be disastrous. Controller hijacking and fraudulent rule insertion attacks are types of malicious attacks. DoS is related to availability related attacks. Types of DoS attack include the Controller-Switch Communication Flood that aims to overload the affected switches and controllers. There are possible countermeasures for each of the likely attacks. Attacks targeting the forwarding plane could be avoided by proactive rule caching,

rule aggregation, increasing switch buffering capacity, decreasing switch-controller communication delay, and packet type classification based on traffic analysis. Other attacks that target the control and application plane could be mitigated by controller replication, dynamic master controller assignment, efficient controller placement, controller replication with diversity, and resourceful controller assignments [40].

2.3.2. Centralised and Distributed SDN Controllers

The proposed control plane design solutions that cope with scalability issues can be classified into two categories, i.e. centralised controller and distributed controller deployment. Currently many SDN controllers are available, both as open source and commercial. The controllers have their specific features and capabilities, especially in their support of solutions for the SDN scalability challenge. A summary of controller platform deployments is presented in Table 2-1.

Table 2-1 Summary of Controller Platforms [42]

Controller Name	Programming Language	Open Source	Architecture	Description
Beacon [43]	Java	Yes	Centralised Multi-threaded	Cross-platform, modular, Java-based OpenFlow Controller that supports event-based and threaded operation.
Maestro [44]	Java	Yes	Centralised Multi-threaded	A network operating system based on Java which provides interfaces for implementing modular network control applications.
Floodlight [45]	Java	Yes	Centralised Multi-threaded	Java-based OpenFlow Controller, based on the Beacon implementation, works with physical and virtual OpenFlow switches.
RISE [46]	C & Ruby	Yes	Centralised	OpenFlow Controller based on Trema, which is an OpenFlow stack based on Ruby and C.
ONOS [47]	Java	Yes	Distributed	Open source SDN controller platform explicitly designed for scalability and high-availability. With this design, ONOS projects itself as a network operating system, with separation of control and data planes for WAN and service provider networks.

Controller Name	Programming Language	Open Source	Architecture	Description
RYU [48]	Phyton	Yes	Centralised Multi-threaded	SDN operating system that aims for logically centralised control and APIs, to create new network management and control applications.
NOX/POX [49]	Phyton	Yes	Centralised	The first OpenFlow controller.
OpenContrail [50]	Phyton	Yes	Distributed	An open source version of Juniper's Contrail controller
OpenDaylight [51]	Java	Yes	Distributed	An open source project based on Java. It supports OSGi Framework for local controller programmability and bidirectional REST for remote programmability as Northbound APIs.
OpenMUL [52]	C	Yes	Centralised Multi-threaded	C-based multi-threaded OpenFlow SDN controller that supports a multi-level northbound interface for attaching applications.
Ericsson SDN Controller [53]	Java	No	Distributed	Ericsson used the OpenDaylight as the base of its controller and bind a Policy Control to drive end-user service personalisation in network connectivity.
HP VAN SDN Controller [54]	Java	No	Distributed	OpenDaylight based controller with HP contributions in AAA, device drivers, OpenFlow and Hybrid Mode, clustering for High Availability, multi-application support including the Network Intent Composition (NIC) API, Persistence, Service Function Chaining, OpenStack integration and federation of controllers.
Programmable Network Controller [55]	Java	No	Distributed	OpenDaylight based controller offered by IBM as part of its Data Center Solution.
Contrail [56]	Phyton	No	Distributed	Contrail Controller is Juniper's software controller that is designed to operate on a virtual machine (VM). Contrail exposes a set of REST APIs for northbound interaction with cloud orchestration systems, such as OpenStack, as well as other applications.
Open SDN Controller [57]	Java	No	Distributed	OpenDaylight based SDN Controller with additional Cisco's embedded applications, robust application development environment and additional OpenFlow protocol support for Cisco Multiprotocol Label Switching (MPLS) extensions.
Huawei IP SDN Controller [58]	Java	No	Distributed	OpenDaylight based SDN controller with the addition of Huawei's Open Programmability System (OPS), which implements multi-layer capability openness including network control and management

A description of the centralised and distributed approaches follows:

a. Centralised Controller Approach. SDN may have either a centralised or distributed control plane [18, 29], although the protocols such as OpenFlow specify that a controller controls a switch, and this seems to imply centralisation. OpenFlow is not defined for controller-to-controller communication, but it is apparent that something similar is needed for distribution or redundancy in the control plane. The centralised controller approach is based on a single centralised controller that manages and supervises the entire network or domain. In this model, network intelligence and states are logically centralised inside a single decision point. This centralised controller uses the Southbound interface protocol (e.g. OpenFlow) to conduct global management and control operations. The centralised controller must have a global view of the entire network, including the load on each switch along the routing path. It also must monitor link bottlenecks between the remote SDN nodes. Additionally, statistical information, errors and faults from each network device can be collected by the controller from the attached switches and this information is passed on to another entity, which is often a database and analytic system that identifies switch and network loads and predicts future loads. As mentioned before, the single controller approach exploits two methods, i.e. utilising the hardware and reducing the overhead to minimise controller loads. The first method provides performance scalability at times of high load or when a controller failure occurs. Conventional software optimisation techniques can be used to improve the controller's performance. Multi-core hardware that supports multi-threading can be used to support parallel process optimisation, load balancing, and replication. High-performance controllers, such as McNettle [59], targeted powerful multi-core servers and are being designed to scale

up to handle large data centre workloads (around 20 million flow requests per second and up to 5000 switches). Despite the performance improvements presented in this approach, there are limitations and challenges. The hardware setup cannot be easily modified due to its rigid nature. Another limitation is that this approach retains a Single Point of Failure due to its single-controller architecture. The second method tries to reduce the controller load, as highlighted by proposals including DIFANE [54] and DevoFlow [55], by extending the switch data plane mechanism. DIFANE partly uses intermediate switches called authority switches to make the forwarding decisions instead of relying on the centralised controller to reduce the load and the latencies of rule installation. A similar approach is also proposed in DevoFlow with the selection of flows to be directed to the controller, while switches handle the other flows.

b. Distributed Controller Approach. The distributed controller approach uses multiple controllers to manage, control and supervise the network. The controllers are distributed around the network and can be called distributed controllers. There are two classes of distributed SDN controllers, i.e. logically centralised but physically distributed controller, and the exclusively distributed controller. Distributed controllers have several key challenges that should be addressed to improve the scalability and robustness of networks. First, the distributed approaches requires that a consistent network-wide view be maintained in all of the controllers. Also, the mapping between control planes and forwarding planes must be automated instead of the current static configuration which can result in an uneven load distribution among the controllers. Second, it is difficult to obtain an optimal global view of the entire network. Further, it is difficult to identify an optimal number of distributed controllers that provides a linear scale-up of the SDN network when their number increases. Finally, most of these approaches use local algorithms to develop coordination

protocols in which each controller needs to respond only to events that take place in its local neighbourhood. Thus, there remains the need to synchronise the overall local and distributed events to provide a global picture of the network. The first class of distributed controllers is the logically centralised but physically distributed controller. The controllers share information to build a consistent view of the entire network. They are using either a distributed file system, a distributed hash table, or a pre-computation of all possible routing combinations to centralise their logic. This approach imposes a strong requirement: a strong consistent network-wide view that is maintained in each of the controllers. The network-wide view is maintained via controller-to-controller synchronisation. When the local view of a controller changes, the controller will synchronise the updated state information with the other controllers. The information exchange or state synchronisation among controllers consumes network resources; therefore, it is critical to reducing the resulting network load, while keeping the information consistent for the logically-centralised control plane. The examples of this implementation are Onix [56], Hyperflow [57] and Devolved controllers [59]. The second type of distributed controller model is the exclusively distributed controller. It introduces a physically distributed control plane state and logic; therefore, there is no synchronisation of network states between controllers to maintain a global view. Synchronisation could lead to a network overload due to the frequent network changes, and it suffers from control state inconsistency which will degrade the performance of applications running on top of SDN. The examples of this implementation are Kandoo [60], Distributed Multi-domain SDN Controllers (DISCO) [61], Pratyaastha [62] and Open Network Operating System (ONOS) [47].

2.4. SDN in the Wide Area Network

The introduction of the SDN paradigm into the WAN has commenced [63, 64] and promises to improve management and control efficiency and performance. One of the critical factors for SDN implementation in the WAN is scalability. Issues such as single point of failure and performance bottleneck [8, 20] can appear due to the centralised nature of SDN. Decisions as to controller placement can cause latency in the flow setup [32, 34], and restrict the capability for controller to switch control message flow [33].

There are two approaches for SDN deployment in the WAN, i.e., using a single controller or multiple controllers. Due to its limitation of becoming a single point of failure and lower performance, the single controller approach is not likely to be suitable for SDN implementation in the WAN [64]. In the multiple controller approach, two architectures are proposed, i.e. the centralised architecture and fully-distributed or multi-domain architecture [42, 65]. The centralised architecture can be considered to be similar to the initial design for SDN, while the distributed or multi-domain architecture has a different design, in which it creates distributed control planes [66]. Ongoing research on multi-domain SDN has suggested two architectures, i.e., horizontal and vertical.

2.4.1. Multi-domain SDN Architecture

A multi-domain SDN architecture refers to a network architecture that connects multiple SDN domains. SDN domain refers to the administrative SDN domain, which might be a sub-network in a data centre network, or a carrier or an enterprise network, or an Autonomous System (AS). Most of the distributed control plane architectures with a logically centralised approach such as Onix, Hyperflow, and Devolved controllers currently cannot manage inter-domain flows between SDN domains. According to [67], the fully distributed SDN controller architectures could be utilised for multi-domain SDN communication. A summary of contributions in multi-domain SDN is presented in Table

2-2.

Table 2-2 Summary of Multi-domain SDN Activities [42]

Contributions	Descriptions
Elasticon	Proposed proposes a controller pool which dynamically grows or shrinks corresponding to traffic conditions, and the workload is dynamically allocated among the controllers [68].
Pratyaastha	Proposes a novel method for assigning SDN switches and partitions of SDN application state to distributed controller instances [62].
DISCO	Proposed an open and extensible distributed SDN control plane able to deal with the distributed and heterogeneous properties of modern overlay networks and WAN [61].
Kandoo	Proposed a hierarchical model of distributed controllers, i.e. root controller and local controllers [60].
ONOS	Propose the distributed architecture for high availability and scale-out SDN [47].
ODL SDNi	OpenDaylight split the network logically and physically into different slices or tenants. It developed a multi-controller instance by using an inter-SDN controller communication application called ODL-SDNi (Software Defined Network interface)[51].
IETF SDNi	Proposed an IETF draft to connect SDN domains using an automated system [69]
East-West Bridge	Proposes an information exchange platform for inter-domain SDN peering [70].
Novel SDN Multi-Domain Architecture	Proposed multi-domain SDN architecture and defined an interconnection protocol [71].

There are two implementation approaches for multi-domain SDN: vertical and horizontal [64, 72], as shown in Figure 2-1. In the vertical approach, the controllers are connected into a hierarchical control plane where the controller functionality is organised vertically. In this deployment model, control tasks are distributed to different controllers depending on selected criteria such as network view and locality requirements. The communication between SDN controllers are performed via RESTful APIs. Local events are handled by

the controller that is lower in the hierarchy, and global events are handled at the higher level, which is called the master controller. In the horizontal approach, multiple controllers were organised in a flat control plane where each one governs a subset of the network switches, and the SDN controllers can communicate with each other using a standard inter-domain SDN protocol.

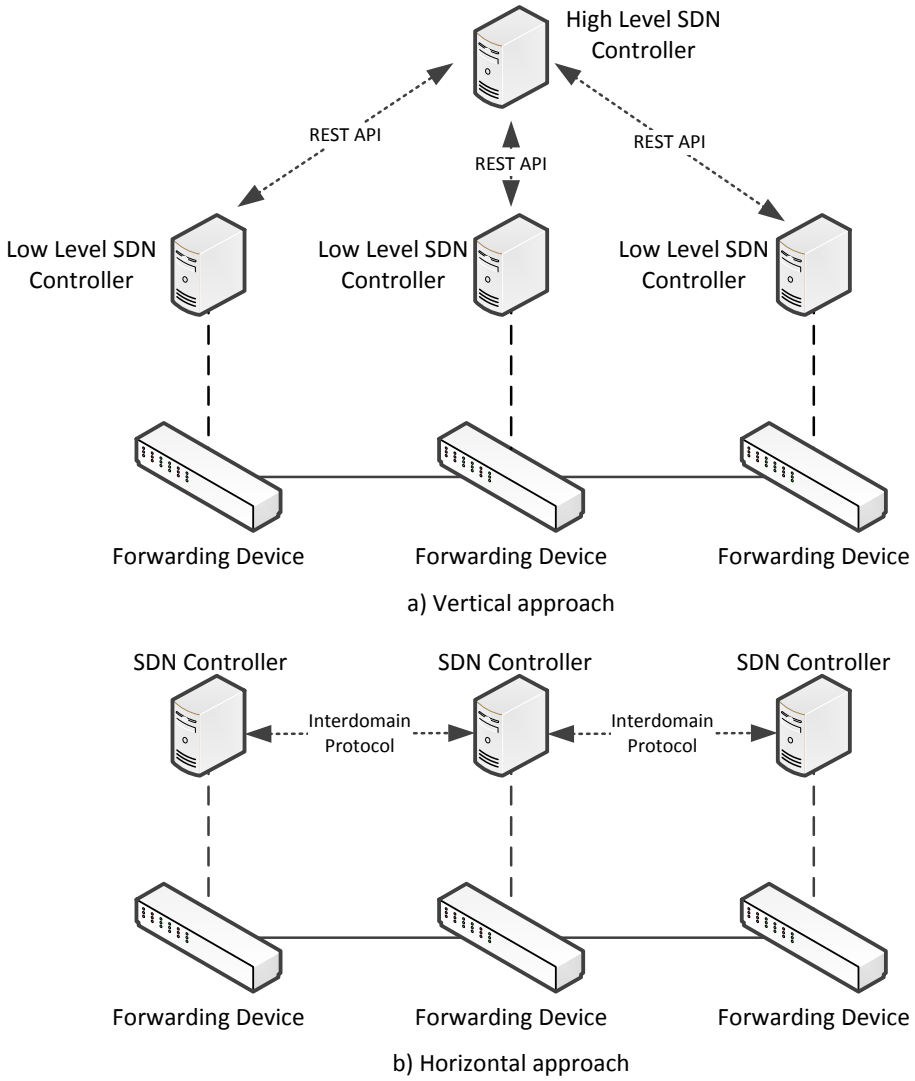


Figure 2-3 Implementation Approaches for Multi-domain SDN.

As discussed in [72], though the vertical approach can work for data centres, problems may arise if this approach is to be scaled up in an enterprise that spreads across different locations. The horizontal approach appears more realistic for scaling across multiple locations, as controllers in this approach can communicate with each other using a

standard inter-domain SDN protocol. This approach will maintain a federation between domain and keeps the SDN controller in each domain independent to manage the policies and path setup within its control.

2.4.2. Inter-Domain SDN Communication

The SDN inter-domain communication protocol can be implemented using the SDN controller East-Westbound interface. Its main functions are to set up a connection between controllers in different domains and exchange control, service and application information. As mentioned in the Section 2.2.2, the East-Westbound interface has not been standardised, which could lead to an interoperability challenge in the deployment of multi-domain SDN. Currently, many SDN controllers have been developed by open-source communities and private companies, complying with the Southbound interface standard, i.e. OpenFlow, but there are no interoperable protocols for the East-Westbound interface.

An IETF draft, titled SDN interconnection (SDNi), was proposed in 2012, as the initial work in defining the SDN inter-domain communication protocol [69]. This draft proposed a definition of a protocol to connect SDN domains using an automated system. However, the draft that was placed in the IETF data tracker in mid-2012 expired in December 2012, without any work progressing. SDNi proposed the extension of BGP and the Session Initiation Protocol (SIP) as implementation protocols.

Google® published a paper on its Software-Defined Wide Area Network (SDWAN), called B4 [15]. It presented the design and evaluation of its global data centre to data centre connectivity exploiting SDN. This excellent design showed the benefit of SDN in managing significant data traffic volumes between data centres, but still, the domains are inside the same administration entity. B4 used BGP as the communication protocol

between controllers in different data centres.

Another SDN inter-domain communication framework was introduced by DISCO [61]. DISCO was developed on top of the Floodlight controller. It used the Advanced Messaging Queuing Protocol (AMQP) as a base for implementing the inter-domain messenger. The use of this protocol created a limitation in the interoperability of DISCO with legacy IP networks. An additional BGP agent was proposed that would be added into the DISCO architecture to solve this limitation.

The interoperability between SDN with legacy IP networks was introduced by the work done in Software Defined Internet Exchange (SDX), which maintains BGP as the protocol to interconnect SDN domains with the legacy IP network domain [73]. Another SDN and IP network interworking was introduced in [74], which proposed a Route Information Base (RIB) synchronisation protocol to send RIB updates towards the SDN controller from BGP Speakers.

Table 2-3 Summary of Current SDN Inter-Domain Communication

Inter-Domain SDN Communications	Proposed Base Protocol(s)	Description
IETF SDNi	BGP or SIP	Using the extensions of BGP or SIP
DISCO	AMQP	Limit the interoperability with legacy IP network, proposed an additional BGP agent
SDX	BGP	Promote the interoperability between SDN domain with legacy IP domain
SDN-IP interworking	BGP	Based on RIB synchronisation
EW Bridge	BGP	Modify BGP update into JSON form
ODL SDNi	BGP	Exploit BGP applications of the controller
ONOS SDN-IP	BGP	Exploit BGP applications of the controller

East-West Bridge (EW Bridge) was introduced as an advanced development of interworking between SDN and legacy IP networks; EW Bridge is a platform to exchange elementary network information between different domains [70]. This platform also enables third-party innovations to be realised in the multi-domain environment. The EW Bridge design kept the principles used in BGP with the modification of the BGP UPDATE message to utilise JSON (Javascript Object Notation).

From the SDN controllers side, an OpenDaylight (ODL) controller application was developed to support the multi-controller functionality [75]. The ODL-SDNi (SDN interface) exploits the BGP application inside the ODL controller to facilitate the exchange of information between ODL controllers.

Almost at the same time, the ONOS controller was expanded to include BGP interfacing applications, i.e. the SDN-IP application [76], which permits interconnect between SDN domains with legacy IP network domains and with other SDN domains.

2.5. Border Gateway Protocol

The BGP is a core building block of the Internet that enables the exchange of information about networks, between Autonomous Systems (ASes) [77]. Therefore, one part of the Internet knows how to reach another part. BGP was known as the Inter-AS routing protocol [78].

2.5.1. BGP Overview

The BGP protocol is commonly used to exchange routing information from an AS to another AS. It is done by setting up a communication session between bordering AS, using a Transmission Control Protocol (TCP) link with port 179, for reliable delivery. This TCP link should always be connected and runs over a physical link that connects AS border routers.

The routing information update in BGP follows a path-vector routing protocol approach [78], which is based on a distance vector-type approach where looping is avoided through path tagging and where reliable sessions, for information exchange, are used. Network reachability information in BGP contains the network number, path-specific attributes, and the list of AS numbers that a route must transit to reach a destination network. This list is contained in the AS-path attribute. The BGP path-vector routing algorithm is a combination of the distance-vector routing algorithm and the AS-path loop detection.

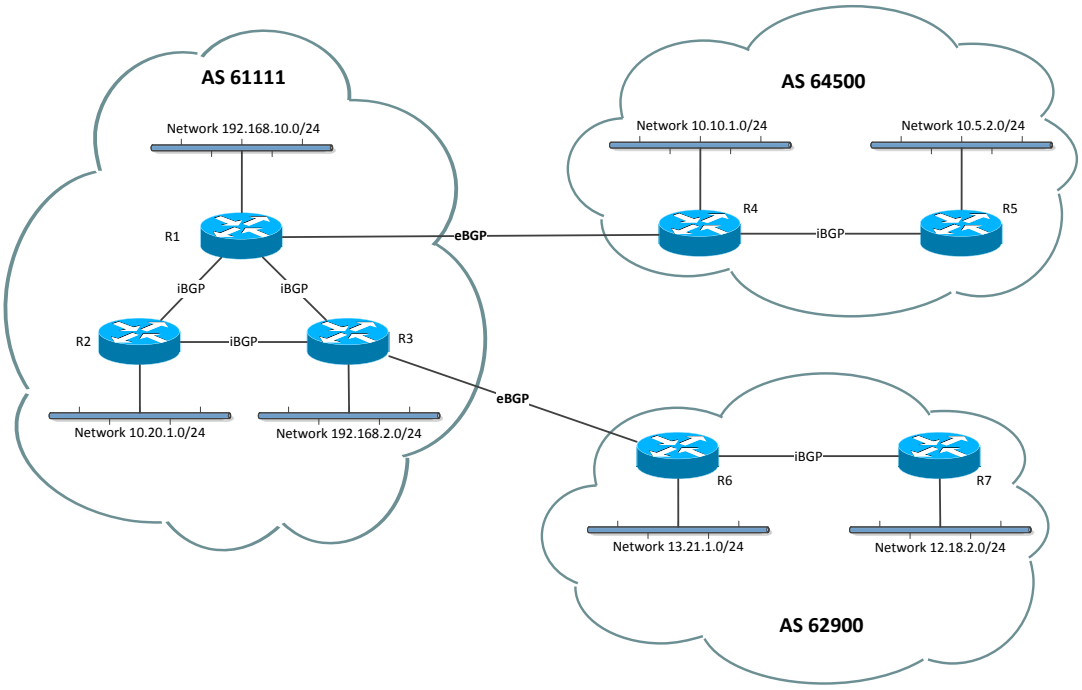


Figure 2-4 BGP Network Examples

BGP selects a single path as the best path to a destination host or network by default. The algorithm for best path selection analyses path attributes to determine which route should be installed as the best path in the BGP routing table. Each path carries well-known mandatory, well-known discretionary, and optional transitive attributes that are used in BGP best path analysis.

There are two types of BGP sessions, based on its implementation scenario. The first type is the external BGP (eBGP) session, which is created when a BGP speaker wants to

connect to another BGP speaker in a different AS. The second type is the internal BGP (iBGP) session, which is created to set up a peer connection between two BGP speakers within an AS. Rules regulate the use of eBGP and iBGP, i.e.:

a. A BGP speaker can advertise IP prefixes it has learned from an eBGP speaker to a neighbouring iBGP speaker; similarly, a BGP speaker can advertise IP prefixes it has learned from an iBGP speaker to an eBGP speaker.

b. An iBGP speaker cannot advertise IP prefixes it has learned from an iBGP speaker to another neighbouring iBGP speaker.

The underlying mechanism for iBGP and eBGP is the same, when the two rules discussed above are correctly addressed. The illustration of BGP with an eBGP and iBGP example is shown in Figure 2-4.

2.5.2. BGP Operation

The BGP operation follows a Finite State Machine (FSM) described in [77]. The BGP session may report the following states: Idle, Connect, Active, OpenSent, OpenConfirm and Established. Aside from those states, the specification also defines the messages sent between BGP peers, i.e., OPEN, UPDATE, KEEPALIVE, and NOTIFICATION.

A description of the BGP's FSM include:

a. Idle. The first stage of the BGP FSM. In this state, BGP detects a start event, and upon the receiving of a start event, it will initiate a TCP connection to the BGP peer, listens for a new connection from a peer router, and change its state to Connect.

b. Connect. In this stage, BGP initiates the TCP connection to the BGP peer. The Connect state indicates the router waits for the completion of a TCP connection between itself and another BGP speaking peer. If the 3-way TCP handshake

succeeded, the OPEN message is sent to the neighbour and change the state to OpenSent.

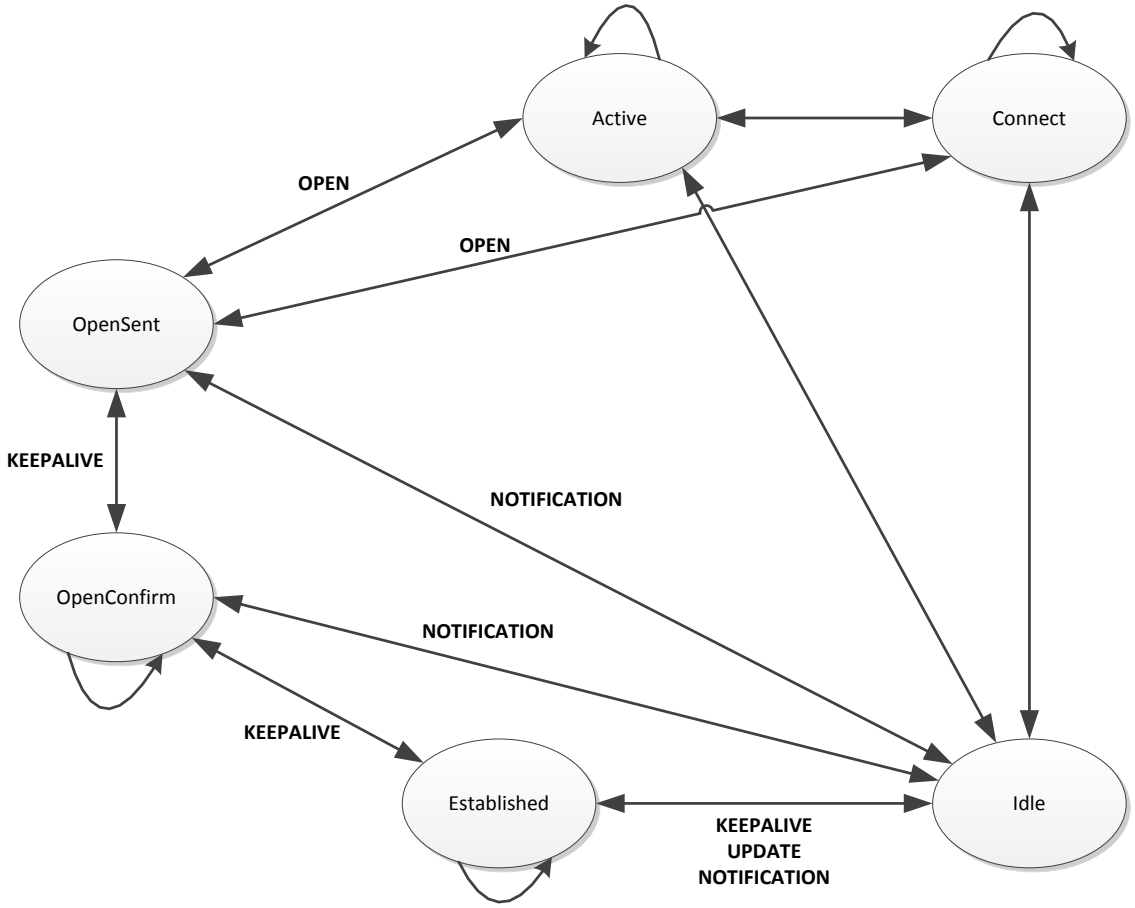


Figure 2-5 BGP FSM and Messages.

c. Active. In this state, BGP starts a new TCP handshake, as the initial connect failed. If a connection is established, an OPEN message is sent, Hold timer is set, and the state moves to OpenSent. If this attempt for TCP connection fails, the state moves back to the Connect state.

d. OpenSent. In this state, an OPEN message has been sent from the originating router and is awaiting an OPEN message from the other router. After the originating router receives the OPEN message from the other router, both OPEN messages are checked for errors. If no error found, KEEPALIVE message is sent, and the state is moved to OpenConfirm. If an error is found, a NOTIFICATION message is sent, and the state

is moved back to Idle.

e. OpenConfirm. In this state, BGP waits for a KEEPALIVE or NOTIFICATION message. Upon receipt of a neighbour's Keepalive, the state is moved to Established. If the hold timer expires, a stop event occurs, or a Notification message is received, and the state is moved to Idle.

f. Established. In this state, the BGP session is established. BGP neighbours exchange routes via UPDATE messages. As UPDATE and KEEPALIVE messages are received, the Hold Timer is reset. If the Hold Timer expires (not reset), an error is detected, and BGP moves the neighbour back to the Idle state.

Also, the description of each BGP's messages are discussed as follows [77]:

a. OPEN. This is the first message sent to establish a BGP session after the TCP connection has been established. Routers use this message to identify itself and to specify its BGP operational parameters. The information carried inside this message is BGP version, AS number of the sender, Hold Timer, BGP identifier, Optional Parameters Length and the Optional Parameters.

b. UPDATE. This message is used to transfer routing information between BGP peers. The information in the UPDATE message can be used to construct a graph that describes the relationships of the various AS. UPDATE message carries routing information such as Unfeasible Routes Length, Withdrawn Routes, Total Path Attribute Length, Path Attributes (describes the characteristics of the advertised path) and Network Layer Reachability Information (NLRI).

c. KEEPALIVE. This message is used by BGP for the keep-alive mechanism, to determine if peers are reachable.

d. NOTIFICATION. This message is sent whenever something wrong has happened, e.g. an error is detected and causes the BGP connection to close.

2.5.3. Path Attributes

Path Attributes are carried by the BGP UPDATE message and describe the characteristics of the advertised path. As described in [77, 79, 80], Path Attributes consist of a triple <attribute type, attribute length, attribute value> of variable length. There are several categories of Path Attributes, i.e.:

a. Well known, Mandatory. This attribute must appear in every UPDATE message. It must be supported by all BGP software implementations. If a well-known, mandatory attribute is missing from an UPDATE message, a NOTIFICATION message must be sent to the peer. Examples:

   - AS_path
   - Origin
   - Next_Hop

b. Well known, Discretionary. This attribute may or may not appear in an UPDATE message, but it must be supported by any BGP software implementation. Examples:

   - Local_Pref
   - Atomic_aggregate

c. Optional, Transitive. These attributes may or may not be supported in all BGP implementations. If it is sent in an UPDATE message, but not recognised by the receiver, it should be passed on to the next AS. Examples:

   - Aggregator

- Community
- Extended Community

d. Optional, Non-Transitive. These attributes may or may not be supported, but if received, it is not required that the router pass it on. It may safely and quietly ignore the optional attribute. Examples:
    - Multi Exit Discriminator (MED)
    - Originator_ID
    - Cluster List

2.6. Conclusion

The literature review had established the knowledge to understand the SDN concepts and the current status of inter-domain communications including a description of BGP. It reviewed past programmable network efforts and the journey to SDN. This chapter discussed the SDN architecture, its interfaces and the OpenFlow protocol as the key SDN enabler. This review has discussed the challenges faced by SDN and provide a comprehensive review of the controller designs to cope with one of the challenges, i.e. scalability.

In this chapter, the design configurations of a SDN implementation in the WAN was presented and discussed the concept of vertical and horizontal implementation approaches. The horizontal approach was perceived as more realistic for enterprise network scaling across multiple locations, while it also can maintain federation between controllers. This approach was adopted during the research project. The literature review has also provided a discussion on the state of the art of inter-domain SDN communication research. From the review, most of the inter-domain SDN communication research

indicated using BGP or emulating the BGP functions in an inter-domain SDN protocol. Finally, the current inter-domain network protocol, BGP, is described, which covers its operations, type of messages and the path attributes, which will be used as a primary reference in the design of an inter-domain SDN communication protocol.

Chapter 3 Inter-Domain SDN Communication Protocol

3.1. Overview

This chapter provides the analysis of BGP as the inter-domain SDN communication protocol. It describes the reasons for using BGP as the inter-domain SDN communication protocol and the process of connecting different SDN domains using BGP. A multi-domain SDN test bed is presented and the comparison of network performance test results between legacy IP network, SDN networks and multi-domain SDN is also presented.

3.2. Requirements for Inter-Domain SDN Communication

Inter-domain SDN communication is needed to connect different SDN domains. Currently, no standardised protocol is defined as the inter-domain SDN protocol. The primary ability of inter-domain SDN communication protocols will be presented, including the requirements for information to be exchanged between SDN domains.

3.2.1. Primary Ability of SDN Inter-Domain Communication

As described in Section 2.4.1, the multi-domain SDN architecture requires an inter-domain communication protocol that supports the exchange of information between domains while maintaining the federation of each domain. The inter-domain SDN communication protocol should have two primary capabilities, as recommended in [69]. Those capabilities are:

a. Coordination of flow setup originated by applications, which will allow the controllers in a different domain to coordinate the flow setup from applications across multiple SDN domains. This flow setup can contain information such as path requirement, Service Level Agreement (SLA), and Quality of Service (QoS).

b. Reachability information exchange to facilitate SDN-based inter-domain routing, which will allow the single flow to traverse along multiple SDN domains, and each controller will select the most suitable path when multiple paths are available.

3.2.2. Message Exchange between SDN domains

An inter-domain SDN protocol will enable information exchange between SDN domains. According to the SDN architecture discussed in Section 2.2.2, there is an interface defined for that purpose, i.e. east-westbound interface. Unfortunately, no standard has been released on this interface, which motivates researchers to utilise existing inter-domain protocols, as described in Section 2.4.2. Apart from the interface, the information exchanged between domains is still being researched. Proposals were described in [69, 72], which listed the type of information that should be exchanged, i.e.:

a. Reachability update, which will update the reachability information to provide a dynamic route for a single flow to traverse multiple SDN domains.

b. Flow setup, tear-down, and update requests: to Coordinate flow setup requests, which contain information such as path requirements, QoS, and so on, across multiple SDN domains.

c. Capability Update: Controllers exchange information on network-related capabilities such as bandwidth, QoS and so on, as well as system and software capabilities available inside the domain.

The East-West Bridge described in Section 2.4.2 and [70], introduced a more structured form of the information exchanged across multiple domains. This information was called the Network View and contains two aspects:

a. network static state information, which includes reachability information (e.g. IP address prefixes), topology (e.g. node, links), network service capability (e.g. Service Level Agreement), and QoS parameters (e.g. latency, packet loss rate)

b. network dynamic state information, which includes flow/table entries information in

each switch, real-time bandwidth utilisation in the topology, and the all the flow paths in the network.

3.3. Inter-Domain SDN Communication Utilising BGP

Research has occurred into SDN inter-domain communications which use BGP as the baseline protocol. BGP is a standardised gateway protocol which enables the exchange of information (routing and reachability) between AS [77]. BGP has the potential as the baseline protocol for SDN inter-domain communication due to:

a. BGP could support information exchange regarding capability and reachability, as part of the existing message format.

b. BGP is a standard and the most feasible protocol for any peer data to be exchanged.

c. BGP would enable interoperability for interconnection between SDN with legacy IP networks.

Therefore, based on these reasons and along with the research found in the literature into SDN inter-domain communication developments, BGP will be used as the baseline inter-domain SDN communication protocol throughout this thesis.

In the current global networks, i.e. the Internet, the interconnection between AS occurs using BGP. BGP is currently the network-wide deployed inter-AS communication protocol. From the previous discussion, BGP also being used to facilitate the communication between different SDN domains.

BGP sessions are used to established communication between SDN domains. The BGP sessions are established over TCP links and follow a state machine approach. The process used for connection establishment is similar to the regular BGP session establishment; only it involves SDN entities such as Controller and OpenFlow Switch along with BGP

Speakers. The step-by-step process was proposed in [72, 81] and shown in Figure 3-1. The process between two SDN domains can be explained as follows:

a. BGP speakers are deployed in each domain on top of SDN controllers. Each BGP speaker has state-machine logic to conduct the message exchange process. BGP speakers will coordinate with the SDN controller to start its process to establish a session between two domains. An event generated from a SDN controller can be used as a trigger for the BGP speakers.

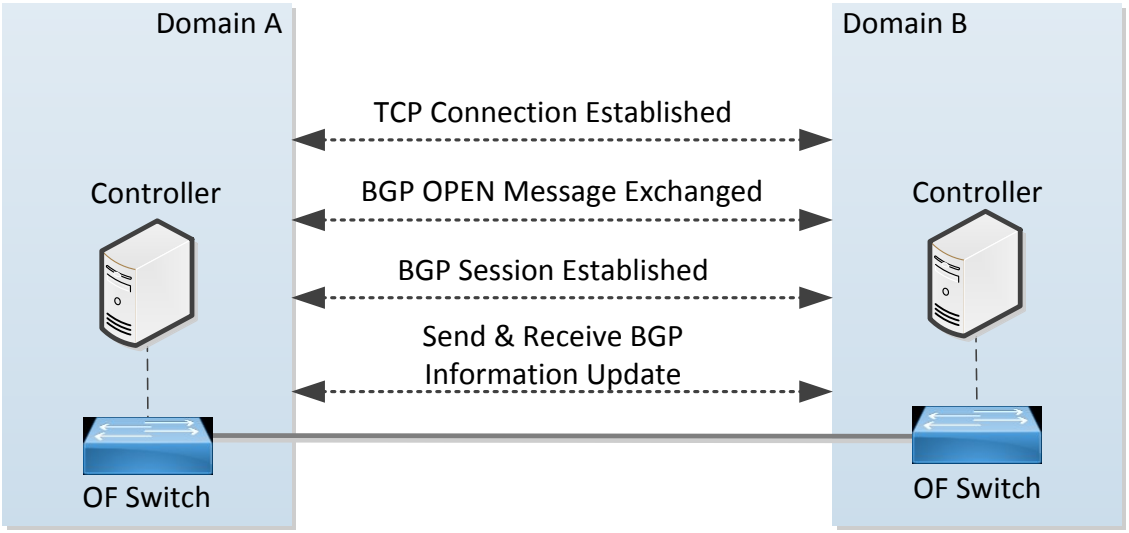


Figure 3-1 BGP Sessions Establishment Process in Multi-Domain SDN.

b. It is assumed that initially, the network administrators in each domain will configure the BGP information manually. The controller will establish a TCP connection with its neighbour. If the connection is not established due to reasons such as TCP timeout or BGP message timer expiry, the BGP speaker must establish a connection with another neighbour, which must be configured by the administrator.

c. BGP will use TCP and once the TCP connection is established, the OPEN message is sent by both the BGP speakers to each other, and after that, they move to the OPEN state.

d. During the OPEN state, BGP speakers can negotiate capabilities of the session through OPEN messages (as per RFC 5492). In each domain, information related to the network topology can be retrieved by the SDN controller(s) using the OpenFlow protocol.

e. Once the BGP peers are in a session, the controllers move to the ESTABLISHED state. The BGP UPDATE messages are exchanged in this state. At this state, the domains can exchange messages which can contain information such as reachability data, bandwidth information and other information.

f. As the two controllers are BGP speakers, the reachability data is maintained in each controller. This information is converted into flows in OpenFlow enabled switches.

g. Route selection is made when more than one path is available based on the BGP process decision. Once the path is established, packets can traverse successfully between SDN domains and later it can support the information exchange.

The communication between multiple SDN domains can be established in the same manner.

3.4. Implementation and Test Scenario

The motivation behind the study is to evaluate the impact of the programmability properties of SDN in a multi-domain environment and to provide a baseline to develop advanced algorithms and schemes for multi-domain SDN environments.

For this study, virtual machines were set up using the VirtualBox virtualisation platform. The virtual machines included the Mininet [82] network emulator and the Ryu controller with Quagga applications [83] inside Vandervecken VM [84] to enable BGP in each of the controllers. Three AS were used as illustrated in Figure 3-2. Each AS will have its

own AS Number (ASN).

The experiments included ICMP messages being sent from Domain A to Domain C, passing through Domain B as an intermediary domain. The latency was measured of the first ping message to determine the flow setup latency and results were compared between the three scenarios.

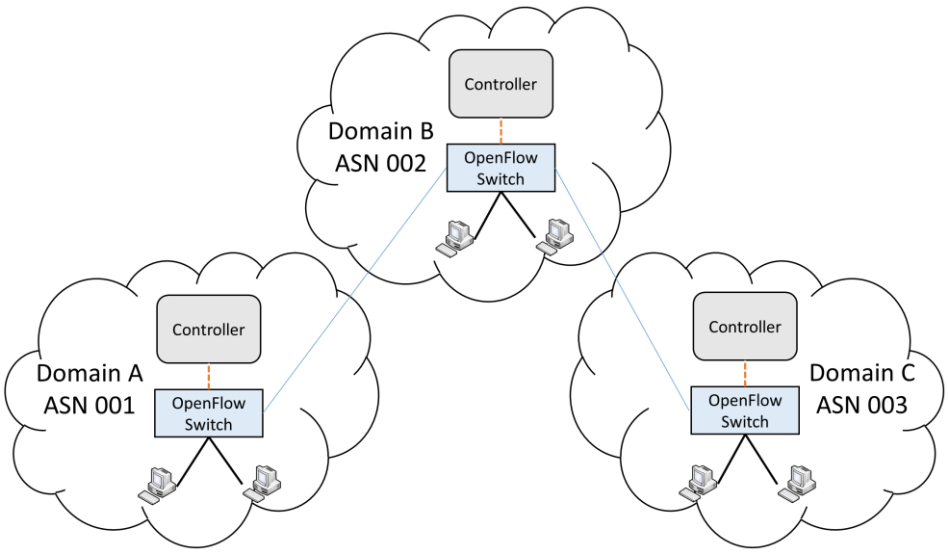


Figure 3-2 Test Topology.

3.4.1. Test Scenario

The scenarios were based on three use cases:

a. Legacy network, in this use case the domains represent legacy networks with switches and manual routing tables inserted into the switches before the operation. In this scenario, no SDN framework is used.

b. SDN without BGP, in this use case, the network uses SDN with default capabilities and no pre-defined routing tables, were implemented in the controllers and switches. In this scenario, the SDN controller did not have any information about the other AS network.

c. SDN with BGP, in this use case the networks use SDN with BGP capable

controllers. In this scenario, each of the AS retains knowledge of their neighbour network and advertises the local network to its neighbour. The experiment process was started when the network initialisation commenced.

The experiments were run multiple times, and the results were the average value of the measurements. For the two SDN scenarios, the network setup always reset to initial conditions (empty Flow Table) during each of the simulations.

3.5. Results

The results presented in Figure 3-3, show that the performance of legacy networks supporting multi-domain communications in terms of latency is still superior to that of the SDN approaches. Almost 90% of the legacy network traffic has a latency below 2.5 ms, while the SDN based approach has a minimum latency of 6.5 ms and 8 ms respectively.

The performance of SDN with BGP outperformed the default SDN without BGP setup for a multi-domain network. Most traffic was found to have a latency between 7.5 ms and 15 ms. The use of BGP reduced the latency significantly.

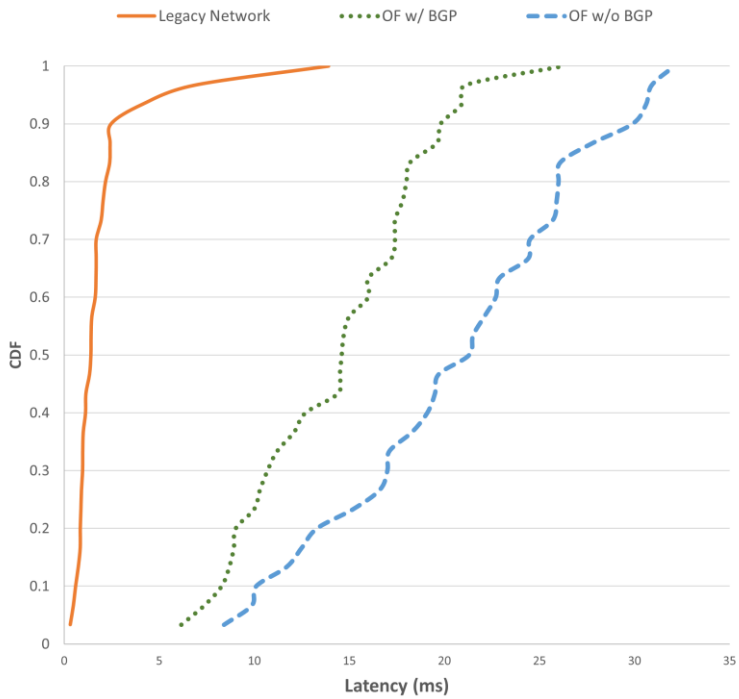


Figure 3-3 Flow Setup Latencies.

The results demonstrate that legacy networking still outperforms SDN based approaches and highlights the need for more work to optimise SDN systems and protocols. No flow setup was needed in the legacy network, due to the pre-operation manual setup and this provided an early performance benefit, but there is an associated cost of the manual intervention. Apart from performance, the legacy network suffers from inflexibility, high operational cost and is tightly dependent on the network devices remaining in sync.

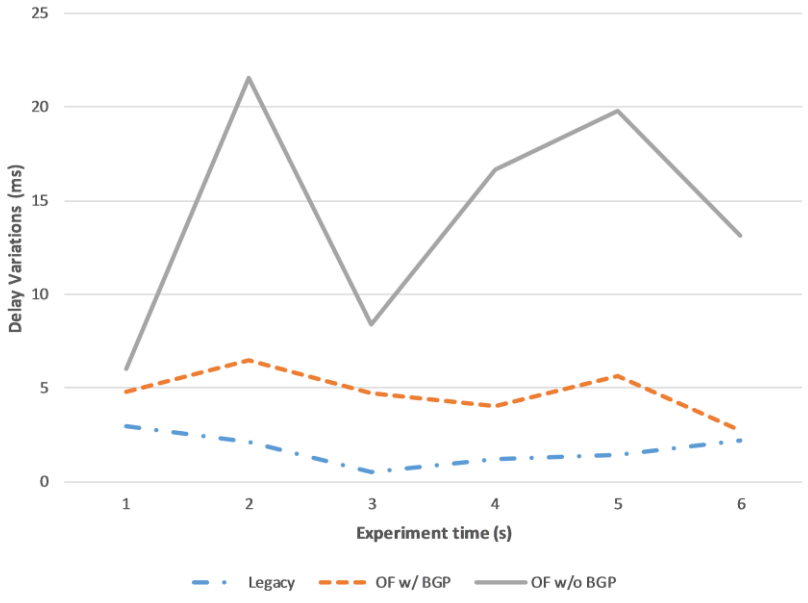


Figure 3-4 End to End Delay Variation.

The results presented in Figure 3-4 show the point-to-point or one-way delay variations from the experiments. As shown from the results, the legacy network scenario showed its delay variation varies from 0.5 ms to 2.95 ms, while the average value of its delay variation is 1.73 ms. The OpenFlow network without BGP scenario showed its delay variation varies from 6.02 ms to 21.54 ms, while the average value of its delay variation is 14.26 ms. Finally, the OpenFlow network BGP scenario showed its delay variation varies from 2.76 ms to 6.47 ms, while the average value of its delay variation is 4.75 ms.

The performance of legacy networks supporting multi-domain communications in terms of delay variation shows its superiority compared to the SDN scenarios. Although as shown in the results, the delay variation of SDN with BGP scenario show a similar stability with the legacy network, while the SDN without BGP scenario shows a more unstable delay variation. With this result, the performance of SDN with BGP scenario could be potentially be accepted by the user with the reference of a legacy network. Some applications which required real-time traffic will depend on the network performance in terms of delay variation.

The performance of SDN in the multi-domain scenarios shows that there is a need to improve the operational outcomes further but highlights the flexibility and automation potential of the programmability provided.

3.6. Conclusion

In this chapter, a comparative study on the inter-domain SDN communication protocols has been presented. The primary ability to send messages for inter-domain communication, the requirement for inter-domain SDN protocol and the type of information that should be exchanged in an inter-domain protocol have been discussed and presented. The discussion was focused on the use of BGP as a prospective protocol, because of its characteristics and capabilities.

The results of a comparative study showed that despite its lack of operational performance compared to a legacy multi-domain network, SDN offers the potential for improved flexibility and reduced network management interaction. There is a need for the SDN programmability properties to be coupled with an improved eastbound/westbound interface and a simplified protocol that facilitates improved performance. The outcomes of the study provide a baseline as a more advanced multi-domain management approach is developed including an improved eastbound/westbound interface protocol.

Chapter 4 Information Exchange Mechanism for Multi-domain SDN Provisioning

4.1. Overview

This chapter describes the mechanism to exchange information between SDN domains to support multi-domain provisioning. As discussed in Chapter 3, BGP will be used as the inter-domain SDN protocol in this research. Therefore this mechanism will be utilising current BGP operations and messages. The design will be implemented inside an ONOS controller by modifying the ONOS SDN-IP application. Finally, verification of the implementation will be presented.

4.2. BGP Message for Multi-domain SDN Provisioning

From the explanation in Section 2.5.2 and Section 3.3, SDN domains can be connected and reasons identified during the literature review support the adoption of BGP as the inter-domain SDN communication protocol, as excerpted from [72, 85], i.e.:

a. In general, BGP has several potential benefits such as a Finite State Machine (FSM) approach which make it easier to implement, it can be used independently as an application layer protocol, and simplicity of its message format is by OpenFlow Message.

b. BGP messages can support the transport of information required for inter-domain SDN communication. That information is capability, reachability and flow setup information. BGP OPEN message has the capability field that can be used, while BGP UPDATE message can be used to carry reachability and flow setup information.

c. The Internet currently is built from the interconnected domains which implement BGP. These domains could be managed by SDN in the future Internet architecture, which will still be reached by legacy IP network because of the backward compatibility introduced from using BGP as an inter-domain SDN communication protocol.

Provisioning in multi-domain SDN should be related to the flow setup activities. Therefore the BGP UPDATE message is the suitable BGP message to support multi-domain SDN provisioning.

4.2.1. Selecting Path Attributes

In the BGP operation described in Section 2.5.2, the Established state starts when two domains are connected and communicate to each other. Within this state, BGP UPDATE messages are exchanged. As described in Section 2.5.3, Path Attributes defines the character of an advertised BGP Route. A set of selection criteria should be determined to help with the decision of which attributes will be used for multi-domain SDN provisioning.

Firstly, multi-domain provisioning should be implemented in a production network which could also have a connection to legacy IP networks and still adopt the implementation of a router gateway. The criteria for the attributes for provisioning purposes are:

a. The attribute used should be accepted by the non-SDN network to maintain compatibility. As SDN domains might be connected to non-SDN domains, the attributes should not affect the connection to non-SDN domains, and both SDN and non-SDN peering should be supported.

b. The received attribute must be forwarded to the BGP Speaker inside the SDN network. This criterion will accommodate the use of ra outer gateway in the implementation of the production network. The router is commonly called the Edge router, and it can be implemented both as the Customer Edge (CE) router and Provider Edge (PE) router.

The matched BGP Path Attributes category is the Optional Transitive, and it will be used

as the first selection criteria. The second selection criteria should be motivated by the published scenarios where path attributes were used to exchange or collect data. Based on this publication, it is assumed that the path attributes could be adopted for carrying provisioning data.

According to the list of Path Attributes published by IANA [86] and based on the first and second criteria which were described above, Table 4-1 shows the selected BGP Path Attributes for multi-domain SDN provisioning, i.e. Community and Extended Communities.

Table 4-1 Selection of BGP Path Attributes for Multi-domain SDN Provisioning

No.	Path Attributes	Criteria 1: Optional-Transitive	Criteria 2: Published Scenarios	Remarks
1	ORIGIN	-	-	Not selected
2	AS_PATH	-	-	Not selected
3	NEXT_HOP	-	-	Not selected
4	MULTI_EXIT_DISC	-	-	Not selected
5	LOCAL_PREF	-	-	Not selected
6	ATOMIC_AGGREGATE	-	-	Not selected
7	AGGREGATOR	-	-	Not selected
8	COMMUNITY	✓	✓	Selected. This path attribute was used for data collection [87].
9	ORIGINATOR_ID	-	-	Not selected
10	CLUSTER_LIST	-	-	Not selected
11	MP_REACH_NLRI	-	-	Not selected
12	MP_UNREACH_NLRI	-	-	Not selected
13	EXTENDED	✓	✓	Selected.

No.	Path Attributes	Criteria 1: Optional-Transitive	Criteria 2: Published Scenarios	Remarks
	COMMUNITIES			This path attribute was used for QoS marking [88] and Link Bandwidth exchange [89].
14	AS4_PATH	✓	-	Not selected
15	AS4_AGGREGATOR	✓	-	Not selected
16	PMSI_TUNNEL	✓	-	Not selected
17	Tunnel Encapsulation Attribute	✓	-	Not selected
18	Traffic Engineering	-	-	Not selected
19	IPv6 Address Specific Extended Community	✓	-	Not selected
20	AIGP	-	-	Not selected
21	PE Distinguisher Labels	✓	-	Not selected
22	BGP-LS Attribute		-	Not selected
23	LARGE_COMMUNITY	✓	-	Not selected
24	BGPsec_Path	-	-	Not selected
25	BGP Community Container Attribute	✓	-	Not selected
26	Internal Only to Customer	-	-	Not selected
27	Unassigned	-	-	Not selected
28	BGP Prefix-SID	✓	-	Not selected
29	Unassigned		-	Not selected
30	ATTR_SET	✓	-	Not selected

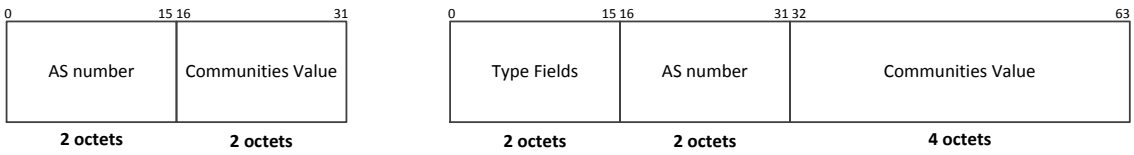
4.2.2. Communities and Extended Communities Attributes for Multi-domain SDN Provisioning

The BGP Communities Attribute is a BGP Path Attribute that may be used to pass additional information to both neighbouring and remote BGP peers [79]. A community is a group of destinations which share common properties. Each AS administrator may define to which communities a destination belongs. By default, all destinations belong to

the general Internet community.

The BGP Extended Communities Attribute is a BGP Path Attribute that provides a mechanism for labelling information carried in BGP-4 [80]. The labels can be used to control the distribution of this information, or for other applications. This attribute provides the enhancement of the Communities Attribute in terms of extended range and additional Type Field.

Each attribute has a data format. The Communities Attribute data format length is four octets or 32 bits, with the two first octets used for the AS number while the last two octets are used for the community values. The Extended Communities Attribute data format length is eight octets or 64 bits, with the first two octets used for Type Fields, the next two octets are used for the AS number, while the last four octets are used for the community values. The data formats are shown in Figure 4-1.



a) Communities Attribute's Data Format

b) Extended Communities Attribute's Data Format

Figure 4-1 Data Formats of Selected Patch Attributes.

The attributes can be adopted so that values for each field can be examined for provisioning purposes. The IETF documentation specifies the values for each field. As the community value is defined privately between peers, any value could be used, except the values that already specified in the IETF document. Those values are presented in Table 4-2 and Table 4-3.

Table 4-2 Communities Attribute Values

Fields	Current Value Rules	Proposed Value for Multi-domain SDN Provisioning
AS number	16 bits AS number 0x0000 and 0xFFFF are reserved	No changes
Community Value	16 bits numerical value, with 0x0000 until 0xFFFF are reserved.	No changes

Table 4-3 Extended Communities Attribute Values

Fields	Current Value Rules	Proposed Value for Multidomain SDN Provisioning
Type Field	0x00 or 0x40 for 2 octets AS number 0x02 set to indicate Route Target Community 0x03 to indicate Route Origin Community	0x03
AS number	16 bits AS number 0x0000 and 0xFFFF are reserved	No changes
Community Value	32 bits numerical values	No changes

From [80], a route may carry both the BGP Communities attribute, as defined in [RFC1997], and the Extended BGP Communities attribute. Based on the BGP FSM operations, an information flow of the message exchanged for provisioning is shown in Figure 4-2. In that diagram, Domain A is the SDN domain that wants to send a provisioning message. It has established peering with Domain B, another SDN domain, using eBGP. Domain A should be able to send to Domain B a BGP UPDATE message which contains Communities and Extended Communities Attributes. Domain B will receive the BGP UPDATE and store the received Communities, and Extended Communities Attributes inside its RIB. A message exchange application in the SDN controller will read the received attributes and extract their values to be used later in the provisioning application.

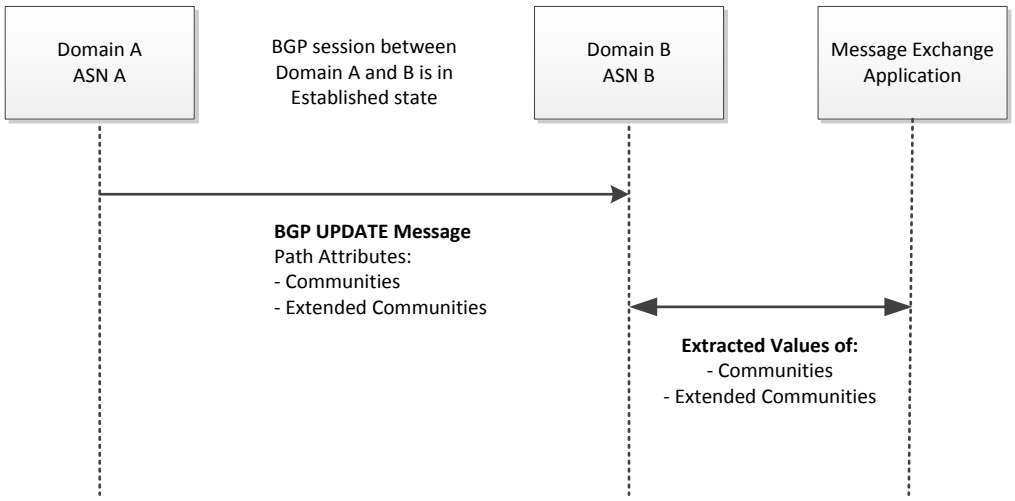


Figure 4-2 Information Flow Diagram of Exchanged Message in Multi-domain SDN.

4.3. Inter-Domain SDN Message Exchange Implementation

The messaging design in Section 4.2.2 was implemented to be verified. The implementation used ONOS SDN controller. ONOS provides an open platform that simplifies the creation of innovative and beneficial network applications and services that work across a wide range of hardware [90]. The ONOS Java APIs are the primary APIs of the ONOS system [91]. ONOS has an application that supports the connection between the SDN domain with another SDN domain or a legacy IP network, called SDN-IP. SDN-IP is an ONOS application that allows SDN domain to connect to external networks on the Internet using the standard BGP [76]. From the BGP perspective, the SDN network is seen as a single AS that behaves as a traditional AS. Within the AS, the SDN-IP application provides the integration mechanism between BGP and ONOS. At the protocol level, SDN-IP behaves like a regular BGP speaker. From ONOS perspective, it is just an application that uses its services to install and update the appropriate forwarding state in the SDN data plane [76].

4.3.1. ONOS SDN-IP Structure

ONOS is a SDN controller that was designed and developed to provide scalability, high performance, resiliency, legacy and next-generation device support [92]. It has been used

as the base for the development of next-generation solutions in the service provider network, such as the Central Office Re-architecture as Datacenter (CORD) [93] and Open Disaggregated Transport Network (ODTN) [94].

SDN-IP as an ONOS application supports the interconnection between SDN domains and the lagacy IP network using BGP. The SDN-IP network architecture shown in Figure 4-3, is composed of OpenFlow enabled switches controlled by an ONOS controller running the SDN-IP application and one or more internal BGP speakers. The internal BGP speakers use eBGP to exchange BGP routing information with the border routers of the adjacent external networks, and iBGP to propagate that information amongst themselves and to SDN-IP application instances [95].

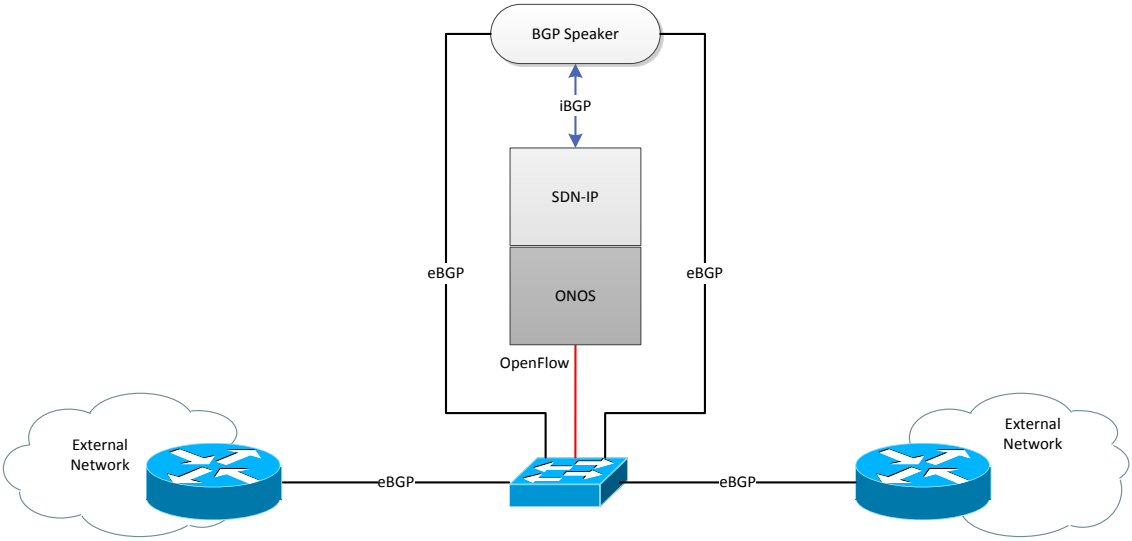


Figure 4-3 SDN-IP Architecture

As described in [95], the BGP speakers within SDN-IP receive the routes advertised by the external border routers belonging to external networks. The routes are processed according to the normal BGP processing and routing policies, and eventually re-advertised to the other external networks. The routes are also advertised to the SDN-IP application instances which act as iBGP peers. The SDN-IP application will select the best route for each destination according to the iBGP rules and translated into an ONOS

Application Intent Request. The Application Intent Request will be translated by ONOS into forwarding rules in the data plane, and those rules are used to forward the transit traffic between the interconnected IP networks. ONOS provides a built-in Intent-Framework [96] for installing low-level flow rules into the network devices using abstract high-level intents. There are two types of Application Intents used by SDN-IP, i.e. Single-point to Single-point intents and Multi-point to single point intents. The BGP speakers and the SDN-IP application instances are interconnected in a typical BGP deployment: a full iBGP mesh, route reflectors, and other deployments. The only difference is that the SDN-IP instances do not need iBGP peering among themselves; i.e., they only need to interconnect with the BGP speakers.

4.3.2. Modification of ONOS BGP Routing Modules

As discussed in Section 4.2.2, to implement the message exchange mechanism, the SDN-IP network should support the implementation of Communities and Extended Communities attributes. The border routers and internal BGP speakers are implemented by an existing BGP router or appropriate BGP software. As for the SDN-IP application, according to its developer's guide [97], the application software modules rely on the ONOS BgpSessionManager module to get the BGP session information from the BGP speaker, as shown in Figure 4-4.

The SDN-IP developer guide [97], describes the BgpSessionManager responsibilities as being responsible for interfacing with the BGP speakers, listening for incoming iBGP connections and creating a BgpSession instance per-session. The BgpSession instance is responsible for the following:

a. Receiving and decoding the BGP protocol messages.

b. Generating and transmitting the necessary BGP control messages (BGP Open and the

periodic BGP Keepalive messages) that are needed to establish and maintain the BGP session.

c. Generating BGP notifications if a BGP error is detected (as defined by the BGP protocol specification).

d. Translating the BGP Update control messages into BGP routing updates that are submitted to the BgpSessionManager.

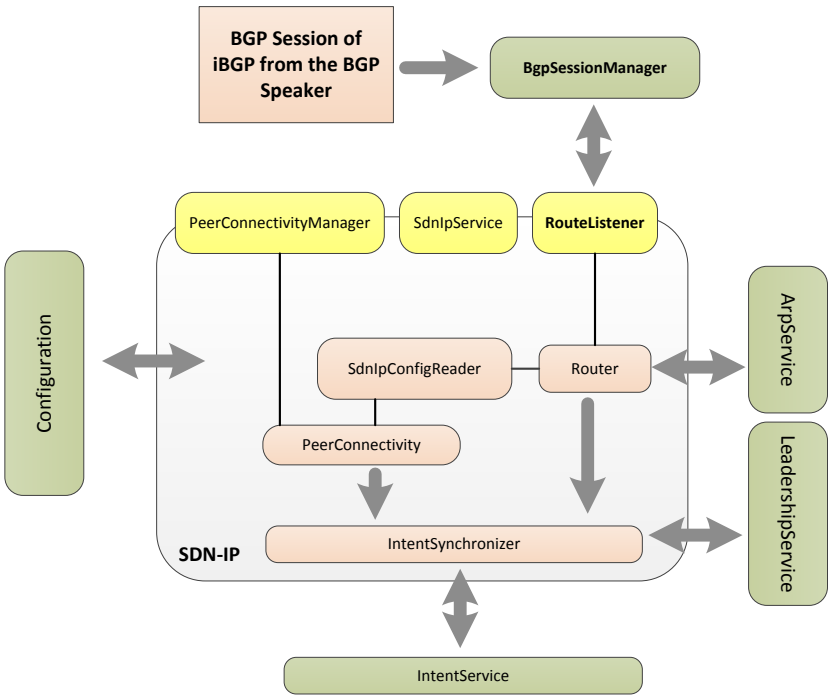


Figure 4-4 SDN-IP Software Modules

An observation of the BgpSessionManager related javascript in the ONOS online repository [98] shows that the current ONOS release has not implemented Communities and Extended Communities attributes. Thus, modifications of the software modules must be carried out to implement those attributes in the ONOS BGP routing modules. Three software modules must be modified, i.e. BgpConstants, BgpUpdate, and BgpRouteEntry.

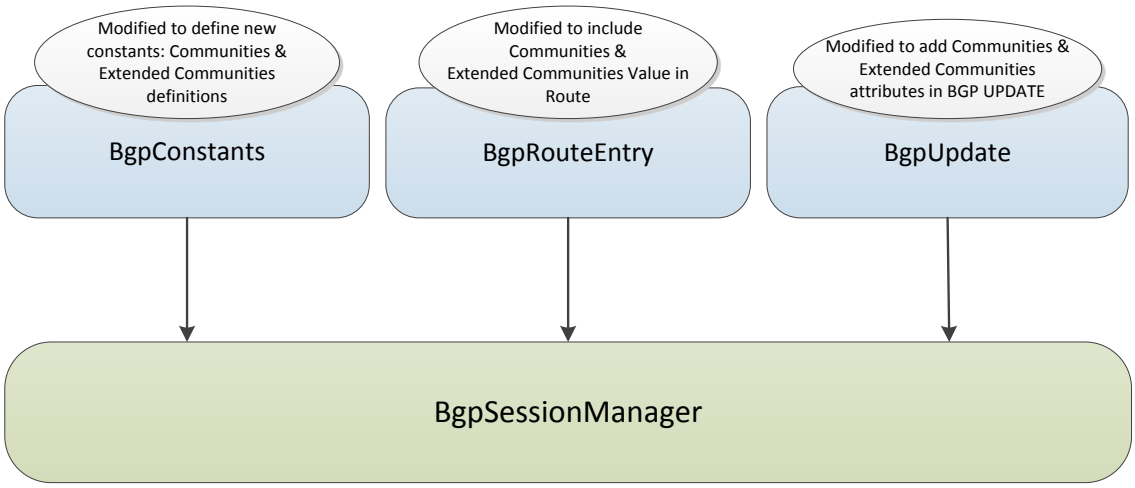


Figure 4-5 Software Modules Modifications

BGP session modules from the ONOS SDN-IP application BGP speaker component were modified to enable the Communities and Extended Communities Attributes. The software modules and their modifications are described in Table 4-4.

Table 4-4 BGP Software Modules Modifications

Software Modules	Software Module Functions	Modifications
BgpConstants.java	Define the BGP Session related constants	Add new constants definitions of Community as eight octets *int* data and Extended Community as 16 octets *int* data. Any BGP session software modules will use these new constants.
BgpUpdate.java	Define a class to handle BGP UPDATE message	Define the "Long" data type wrapper for Community and Extended Community type of data and format the stored value as "Pair<Long, Long>" for Community and Triple<Long,Long,Long>" for Extended Community. Add new parsing process of Community and Extended Community Attributes within BGP UPDATE message and add them in the added Route in BGP. Define the Community and Extended Community format to be read by BgpRouteEntry process, therefore it can understand the format of <AS number>:<Community Values> for Community and <TypeFields>:<AS number>:<Community Values> for the Extended community.

Software Modules	Software Module Functions	Modifications
BgpRouteEntry.java	Represent a Route in BGP	Include the Communities and Extended Communities in the BGP Route advertised in BGP session towards BgpSessionManager. Define a class to represent Communities and Extended Communities Attributes. Adds the new attributes to the existing Hash Code computation process

The complete code for those modifications can be viewed in the appendixes.

4.4. Verification of Message Exchange Mechanism

The implementation of a message exchange mechanism in ONOS was verified by comparing the Communities, and Extended Communities values sent from an external network with the values received by the ONOS controller SDN-IP application. The verification utilises a SDN-IP network topology implemented with Mininet [82] and Quagga Software [83]. Wireshark and an additional ONOS command line interface (CLI) command were used to verify this implementation.

4.4.1. Verification Topology

The verification topology was built using a Mininet SDN emulator. The BGP router and internal BGP speaker were implemented using Quagga [83]. The devices used are presented in Table 4-5, and the topology is shown in Figure 4-6

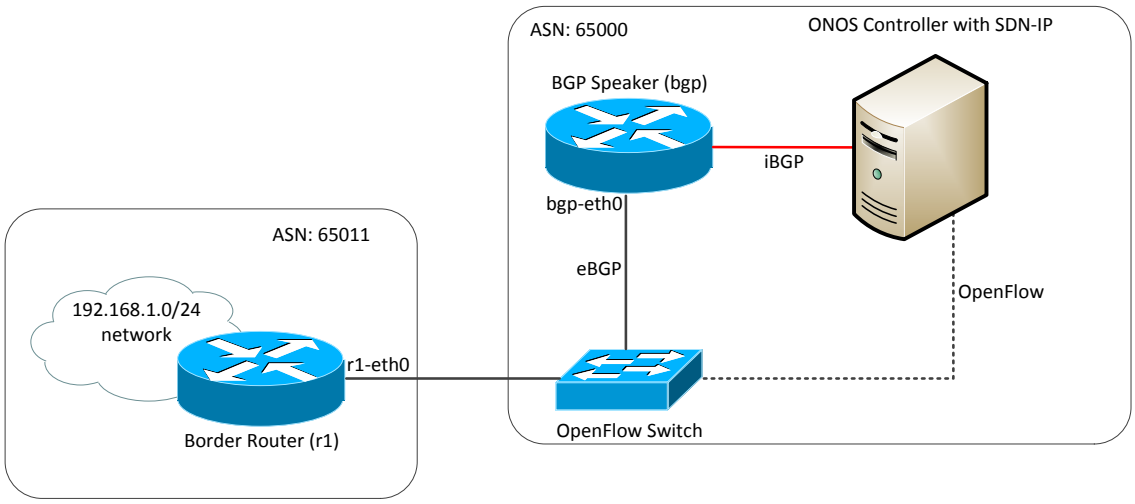


Figure 4-6 Verification Topology

The Border Router (r1) was configured to send the attributes in its routing information, as follows:

a. 65011:200 as Community Attribute

b. 65011:200 as route origin Extended Community Attribute

The ONOS controller with a BGP UPDATE extension, and the activated SDN-IP application were installed, running and connected to the BGP Speaker via a link using TCP port 2000. Wireshark software was setup to capture traffic in the bgp-eth0 interface and using an ONOS console the CLI command was executed.

Table 4-5 Verification Devices

Devices	Interfaces	IP Address
OpenFlow Switch	s1-eth0 and s1-eth2 (Data Plane) s1-lo for (Control Plane)	Control plane uses loopback IP address 127.0.0.1
BGP speaker (bgp)	bgp-eth0 and bgp-eth1	bgp-eth0: 10.0.1.101 bgp-eth1: 10.10.10.1
Border router (r1)	r1-eth0 and r1-eth1	r1-eth0: 10.0.1.1 r1-eth1: 192.168.1.1

4.4.1. ONOS Verification CLI

An additional CLI was initiated and linked to the ONOS BgpRoutesListCommand module. The command is “bgp-routes -c” was executed inside the ONOS console to display the Communities and Extended Communities values which are retrieved from the ONOS controller BGP speaker. The script is shown in the appendix.

4.5. Results

The verification was carried out by starting the SDN-IP network. The Border Router (r1) and BGP speaker (bgp) commence the BGP session and begin sending BGP messages. The Wireshark application captured the packets that pass through the bgp-eth0 interface, which is the BGP speaker interface peered with the Border Router (r1). As presented in Figure 4-7, the Wireshark screen capture shows the content of BGP UPDATE messages received by the BGP Speaker (BGP). The messages include both Community and Extended Community attributes, i.e. 65011:200 as Community and 65011:200 as route origin Extended Community Attribute. The sub-type field value is 0x03 because of the route origin type of Extended Community.

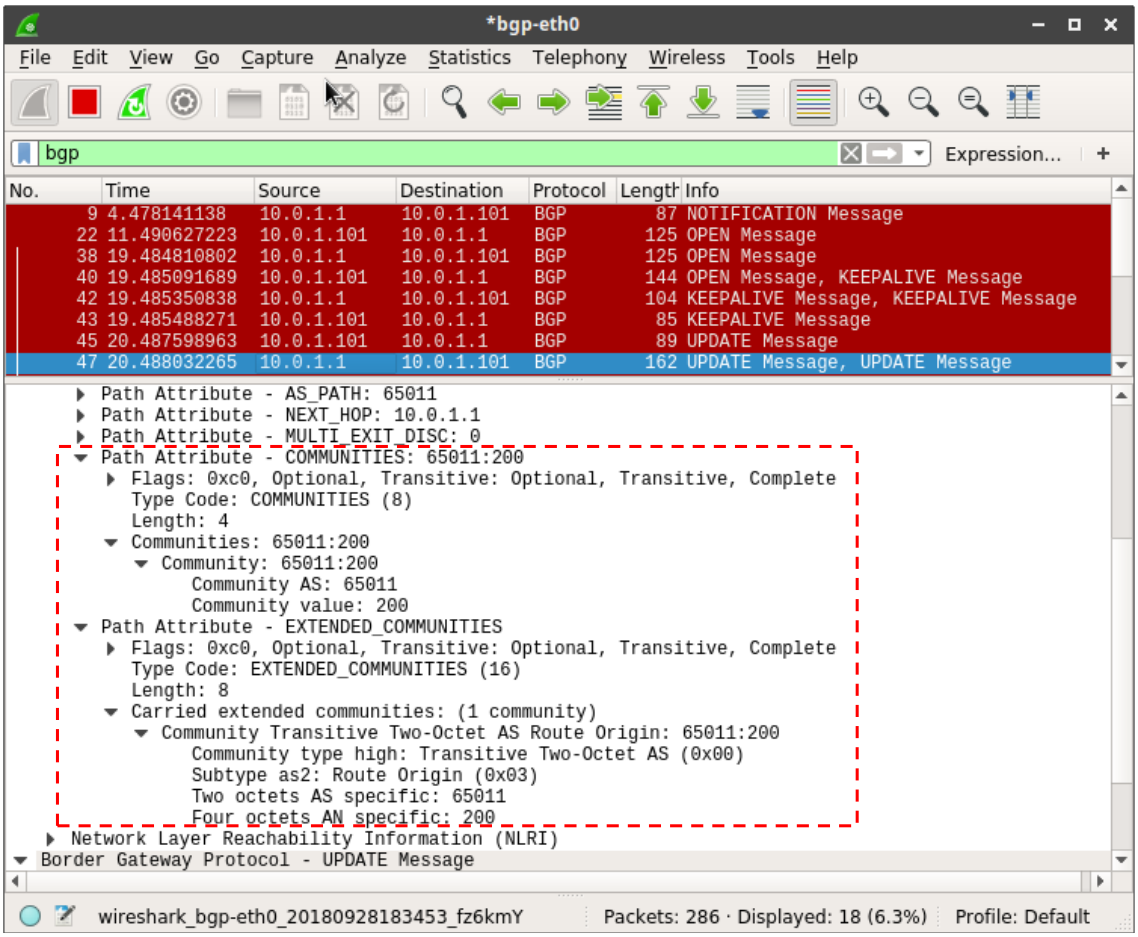


Figure 4-7 BGP UPDATE Message Received by BGP Speaker.

The attributes received by BGP speaker will be stored in its RIB and should be advertised to all peers, including to the ONOS controller SDN-IP application. Using the "show ip bgp" command inside the BGP Speaker console, the attributes information is presented, as shown in Figure 4-8. The result verified the values of the Community and Extended Community attributes are the same value that was sent by the Border Router and the attributes were correctly advertised to the ONOS controller SDN-IP application.

```
bgp> sh ip bgp 192.168.1.0
BGP routing table entry for 192.168.1.0/24
Paths: (1 available, best #1, table Default-IP-Routing-Table)
  Advertised to non peer-group peers:
  10.10.10.2
  65011
    10.0.1.1 from 10.0.1.1 (10.0.1.1)
      Origin IGP, metric 0, localpref 100, valid, external, best
      Community: 65011:200
      Extended Community: SoO:65011:200
      Local update: Fri Sep 28 18:35:16 2018

bgp>
```

Figure 4-8 Routing Table Entry in BGP Speaker.

The SDN-IP application peers with the BGP Speaker using iBGP via a TCP link using port 2000. The verification in the ONOS controller should also test the modifications made to the ONOS software modules. The modifications should enable the ONOS controller BgpSessionManager module to receive the path attributes from the BGP UPDATE message from Border Router (r1) advertised by BGP Speaker (bgp). Using the "bgp-routes -c" command inside the ONOS console, the Community and Extended Community attributes are shown, as presented in Figure 4-9. The result verified that the modification successfully enabled the ONOS controller to retrieve Community and Extended Community attributes from the BGP Speaker advertisement and the values received were the same as the value sent.

```
onos> bgp-routes -c
 IPv4 Network       BGP Community           BGP Extended Community
192.168.1.0/24          65011:200                          3:65011:200


 IPv6 Network       BGP Community           BGP Extended Community
onos>
```

Figure 4-9 Displaying Community and Extended Community Attributes in ONOS.

The results show that both Community and Extended Community attributes can be sent from a legacy IP or SDN domain into a SDN domain. The sender domain can be modified to be an SDN domain according to the test in Section 3.3. Therefore, the results presented verify the successful implementation of a multi-domain SDN message exchange

mechanism.

4.6. Conclusion

In this chapter, the BGP Path Attributes were selected to support provisioning in multi-domain SDN. The selection process considered several criteria, which include published scenarios outside multi-domain SDN provisioning. The selected path attributes were Communities and Extended Communities attributes. Consideration of their data format and flow diagrams in the mechanism were also defined.

The implementation of the proposed information exchange for multi-domain SDN provisioning was developed using an ONOS controller, with its SDN-IP application. The SDN-IP application allows the SDN domain to connect to external networks (SDN or non-SDN) on the Internet using the standard BGP. This implementation required modifications to be made to several of the ONOS controller software modules. The modifications were intended to enable the retrieval of Community and Extended Community attributes.

The verifications were carried out using a SDN-IP network topology and additional ONOS CLI command shells. The results of the test scenario, in the form of Wireshark capture results from the BGP Speaker, successfully displayed both attributes in the ONOS console. The results verified the implementation of the information exchange mechanism for multi-domain SDN provisioning, which forms the basis of the multi-domain SDN provisioning framework provided in the next chapter.

Chapter 5 Multi-domain SDN Automatic Provisioning Framework

5.1. Overview

In Chapter 4, the information exchange mechanism for multi-domain SDN provisioning has been studied. The Communities and Extended Communities attributes were selected as the information transport media for the exchange between domains. The mechanism was implemented in the ONOS Controller SDN-IP application by modifying several software modules.

The exchange mechanism can be used to send specific information that can be used to trigger a provisioning action in the remote domain. The provisioning action would occur automatically without any intervention from the remote domain administrator. In this chapter, the proposed framework for the automatic provisioning in multi-domain SDN will be discussed and implemented using a test bed implementation.

5.2. Multi-domain SDN Provisioning Challenges

Network provisioning is a subset of the network operation processes that are defined by the TeleManagement Forum (TMForum) as the Enhanced Telecommunications Operation Map (eTOM) – The Business Process Frameworx [99]. In this standard, the TMForum defined network provisioning as a resource provisioning process in the resource domain. Resource provisioning is defined as encompassing allocation, installation, configuration, activation and testing of specific resources to meet the service requirements, or in response to requests from other processes to alleviate specific resource capacity shortfalls, availability concerns or failure conditions.

5.2.1. Automated Provisioning

The provisioning of a service in a network involves network operator roles and processes. Network administrators and engineers are the actors who stand in the first line to satisfy customer demand. They need to create and modify the network configuration to balance

the customer requirements with efficient network operation.

The network provisioning automation was developed in part by the motivation to reduce operational complexity and capital expenditure by diminishing human involvement in network operational tasks, as well as optimisation of network capacity, coverage, and service quality [100]. Automation was introduced through the integration of network planning, configuration and optimisation, into automated processes that require a minimum of human intervention. The TMForum has released a definition of their zero-touch provisioning for network management and operation as part of their Zero-time Orchestration, Operations and Management (ZOOM) model [101]. This zero-touch operation was defined as a self-service operation which can respond with the speed and agility to outpace competitors, and as guidance, it should require minimal intervention from expert resources and enable customer configuration.

One of the simplest examples of zero-touch provisioning is the IP address assignment using the Dynamic Host Control Protocol (DHCP) server as described in [100]. In this example, the IP address allocation should be configured using the DHCP server. A computer or any device that is connected in the same network will send a DHCP query and the computer is configured with an IP address allocated by the DHCP server without any end-user intervention.

The use of SDN in automatic provisioning has been mentioned in the TMForum [101]. The programmability of SDN creates an opportunity to develop software that carries out automatic provisioning.

5.2.2. SDN Provisioning

The addition of two network devices usually identifies SDN networks - the controllers and OpenFlow enabled switches. The two network elements work together to facilitate

the control and management of network traffic and to transmit network traffic from one location to another. The controller manages the traffic in the network by manipulating flow entries inside the flow tables found within SDN enabled switches. Flow tables also contain the instructions to be applied to the traffic. When a packet arrives at a switch, the switch will match the header fields with flow entries found in the flow table. If any entry matches, the indicated actions are performed, and switch counters are updated. If packet header fields do not match an entry in the switch flow table, the switch will ask the controller for instructions on what to do with the packet by sending a message to the controller with the packet header. The matching process can be observed in Figure 5-1.

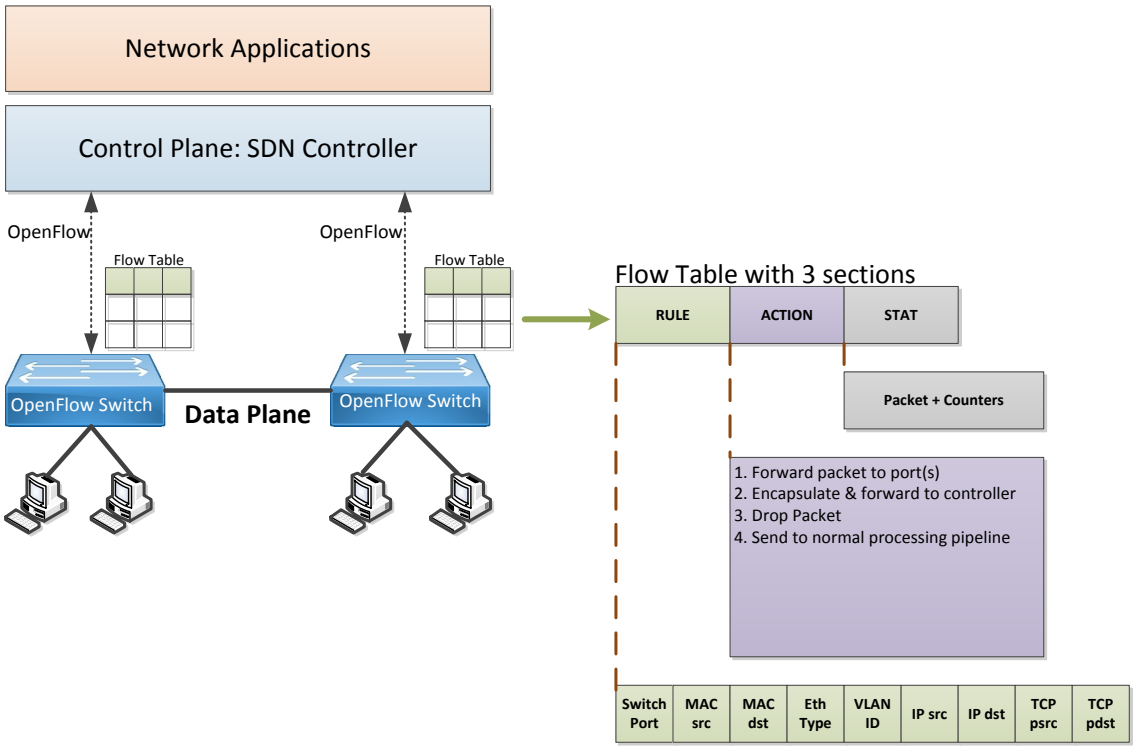


Figure 5-1 Example of Matching Process in SDN.

Figure 5-1 provides examples of the fields of flow entry rules in a flow table with possible actions. This example shows the fields of a packet header. As explained in the OpenFlow specification [30], the match fields identify a unique flow entry in a specific flow table. The specification also explains that a flow entry instruction may contain actions to be performed on the packet at some point of the pipeline. Therefore, every flow table will

have its own match fields along with its actions, in accordance with the OpenFlow specification.

In a single domain SDN architecture, OpenFlow is the protocol that is used to pass messages between the OpenFlow enabled switch and the controller's southbound interface. Before the controller and OpenFlow enabled switch can talk to each other, the administrator will pre-configure the OpenFlow enabled switch to be paired with the controller. Therefore the OpenFlow enabled switch will always accept flow entries from this controller. The OpenFlow specification [30] defines the OpenFlow messages that are used to add, modify or remove flow entries of flow table. This will change how the network behaves and the basic provisioning process in SDN.

In the multi-domain SDN architecture, the first challenge to do provisioning between SDN domains is to identify how information be exchanged between the domains. The second challenge is how the information can be understood as a network provisioning request. Both challenges are basic requirements that need to be fulfilled to implement provisioning in multi-domain SDN.

5.3. Design of a Multi-Domain SDN Provisioning Framework

In the previous section, it was mentioned that SDN has the potential to implement automated provisioning which will reduce human intervention when configuring network devices. It was also mentioned that two fundamental challenges exist when provisioning, i.e. an information exchange mechanism between SDN domains and the format of that information related to the provisioning of the SDN domain. The information exchange was discussed in the previous chapter; therefore, the same mechanism will be applied in this chapter.

5.3.1. Proposed Provisioning Procedures

In multi-domain SDN, the provisioning procedures start when one domain requests a provisioning action to be conducted in the other domain. As a prerequisite, both domain administrators should have agreements about the provisioning actions and parameters that can be carried out using the multi-domain SDN provisioning framework. The agreed actions and parameters are very specific to each party, based on their needs and could be updated when required. Both parties are likely to have this detail included in their SLA documentation.

To describe the common procedures, the SDN domain which requests a provisioning action is defined as the *Sender*. The SDN domain that accepts the request, and conducts the provisioning actions, is identified as the *Receiver*. It is assumed that all of the pre-configuration parameters and provisioning actions have been put in place and both *Sender* and *Receiver* are already connected using a BGP session. Each network domain had an AS number assigned. The procedures can be explained as follows:

1) *Initiation of the provisioning request.* In this first step, the *Sender* will initiate a provisioning action that will be sent to the *Receiver*. The provisioning action is matched with an agreed list of provisioning actions that is included in the current SLA. The SLA identified list will provide values that are to be sent from the *Sender* to the *Receiver*.

2) *Creating the provisioning request*. In the second step, the *Sender's* controller will create an update in its BGP module to add Communities and Extended Communities attributes with the values determined from the previous step. The BGP configuration in the *Sender* will be updated.

3) *Sending the provisioning request*. In this step, the *Sender* will invoke and send the

BGP UPDATE message to the *Receiver*. The BGP UPDATE message will carry the Communities and Extended Communities attributes values to the *Receiver*.

4) *Accepting the provisioning request*. In the next step, the *Receiver* receives the BGP UPDATE message. In the *Receiver* domain, the BGP Speaker accepts the BGP UPDATE message and extracts the Communities and Extended Communities attributes values which are then matched to the agreed SLA list of provisioning actions. If the values extracted from the Communities and Extended Communities attributes matches one or more specific actions in the list, the provisioning actions are then advertised to the SDN controller, which then determines the actions that it needs to take.

5) *Conduct the provisioning action*. If provisioning actions are to occur, the *Receiver's* controller will generate OpenFlow messages with parameters and actions based on the information stored in the SLA agreed provisioning list. The OpenFlow messages are then sent to the linked OpenFlow enabled switch, which installs the flow updates in its flow table.

The steps are illustrated in Figure 5-2.

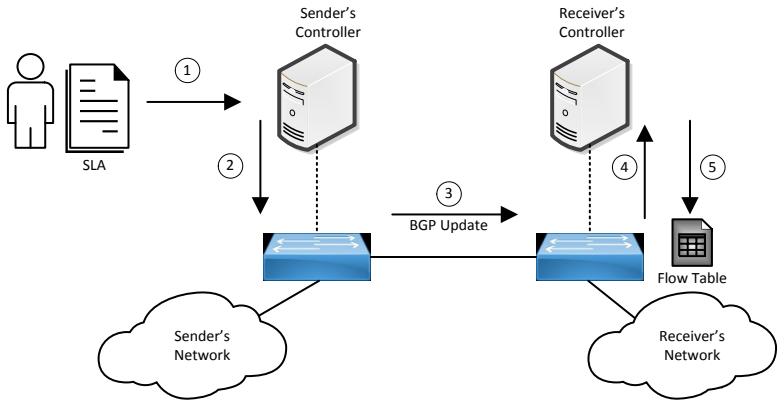


Figure 5-2 Multi-domain SDN Provisioning Procedure

5.3.2. Proposed Framework

The framework shown in Figure 5-3, utilises a BGP session established between SDN domains and uses the BGP UPDATE message as the information exchange media. The framework consists of four main components, i.e. BGP speaker, application modules, SDN Controller, and OpenFlow enabled switch. The framework can be described as follows:

a. The BGP speaker acts as a BGP route server that uses the BGP routing protocol and advertises routes to BGP peers. In this framework, the BGP speaker receives and advertises a prefix to the BGP speaker in the other SDN based domains via eBGP. The BGP speaker also advertises the prefix to the application modules via iBGP. This advertised prefix will contain information to be used by the application module to establish and modify the network service. The additional information for modifying the network service is carried inside the BGP UPDATE message, which is the message that actively exchanges information between BGP peers once the BGP session is established.

b. The application modules receive prefix information from the BGP speaker. Initially, the application modules will make the route selection and push a request to the controller to create forwarding rules in the data plane. The application modules also will check the BGP UPDATE message to extract the Communities and Extended Communities attributes that will be used to create a request to the controller to create new forwarding rules in the data plane. The process will be repeated if another BGP UPDATE message is received with those attributes.

c. The controller in this framework acts to create forwarding rules based on requests from application modules. The controller sends OpenFlow messages to install flow

entries into the linked OpenFlow enabled switch.

d. The OpenFlow enabled switch acts as a forwarding device that forwards traffic according to the flows installed in its flow table. Any changes in the flow table entries will change the path of the traffic. In this framework, the information sent from different SDN domains can be used to modify the flow table entries.

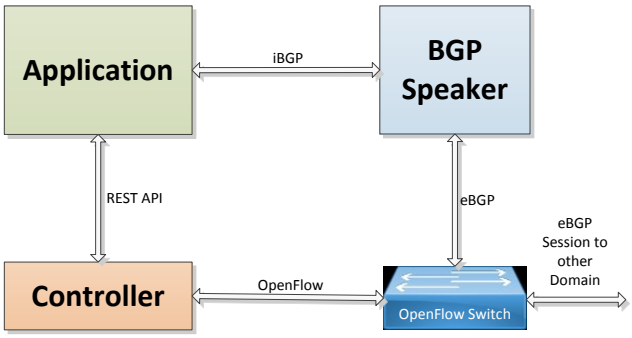


Figure 5-3 Multi-domain SDN Provisioning Framework

5.4. Framework Implementation in ONOS Controller

The automated provisioning framework proposed in Section 5.3.2 was verified and tested. As the information exchange mechanism presented in Section 4.3 was implemented on top of ONOS controller, the provisioning framework was implemented using the same ONOS controller for simplification.

5.4.1. INDOPRONOS Software Modules

The basis of this framework implementation is the ONOS controller that was modified and described in Section 4.3, with the inter-domain SDN information exchange mechanism applied. The existing SDN-IP application will be used in the framework implementation, and new software modules are developed to complete the implementation.

INDOPRONOS is the name given to the new software modules developed to complete the implementation of automated multi-domain SDN provisioning framework.

INDOPRONOS works together with the SDN-IP application module inside the ONOS controller to conduct the provisioning process. From the BGP perspective, the SDN domain will be considered as a single AS domain with the same properties as a traditional AS domain. At this level, SDN-IP behaves like a regular domain interconnection application based on BGP, while INDOPRONOS behaves as the provisioning application.

The SDN-IP application will select the best routes according to the iBGP rules and translates those routes into an ONOS Application Intents Request. ONOS translates the Application Intents Request into forwarding rules in the data plane to forward the transit traffic between the interconnected IP networks. SDN-IP creates the ONOS Intents that are used by the external BGP routers to peer with the BGP Speakers using the interconnected data plane.

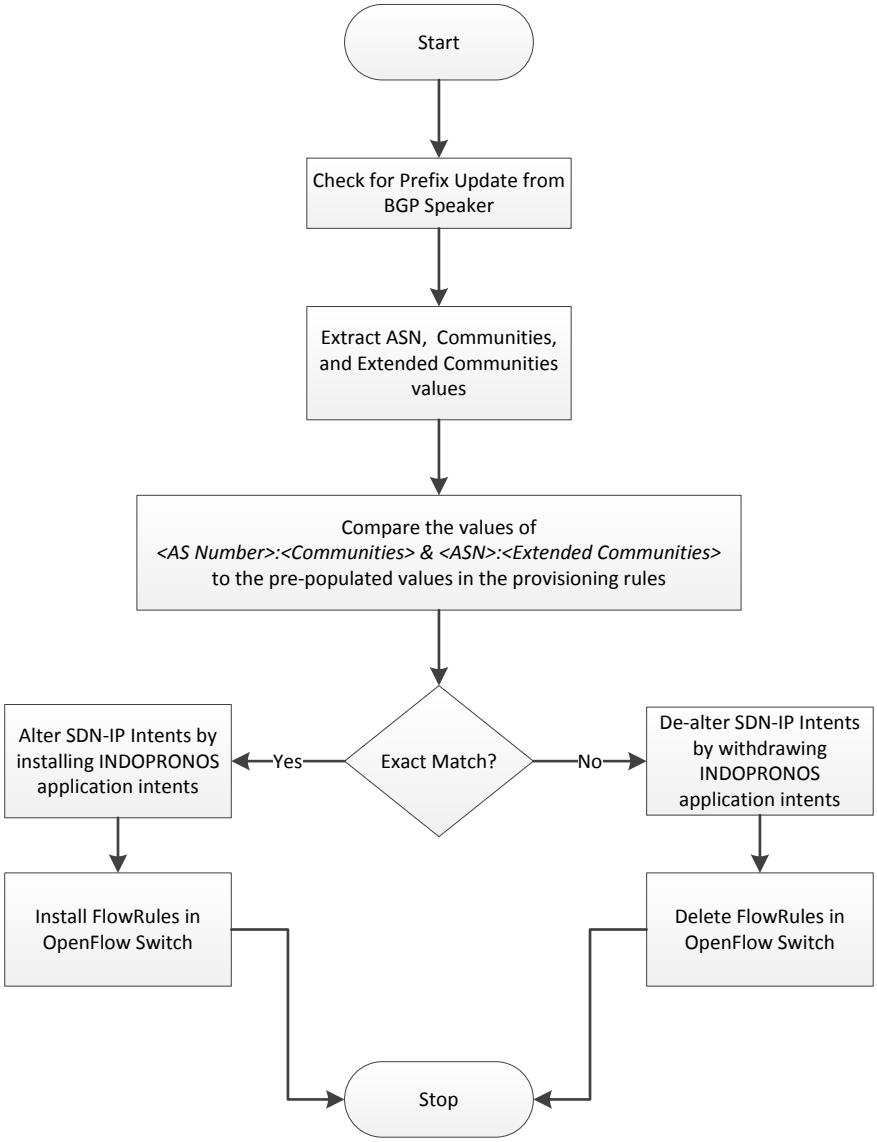


Figure 5-4 INDOPRONOS Application Algorithm

There are two types of Application Intents used by the SDN-IP application [95], i.e. single-point to single-point intents and multi-point to single-point intents. The single-point to single-point intents are unidirectional intents, which are used to establish the BGP peering session between external BGP routers with BGP speakers. The intents connect two single attachment points in the SDN network. The multi-point to single-point intent is a unidirectional intent used to connect the hosts of external networks. Intents are associated with an IP Prefix (IP destination) that connects the ingress attachment points of SDN networks with the single egress attachment point.

The egress attachment point is the connection to the best next-hop router toward the destination IP prefix. With the intent, an IP packet is matched using the IP destination address while entering the ingress edge of the SDN network. The packet will be forwarded toward the corresponding egress attachment point based on the selected forwarding entry that is the best match. Also, right before the packet is forwarded, the "change destination MAC address" action is applied such that the data packet will contain the MAC address of the egress IP router toward the destination.

5.4.2. INDOPRONOS Algorithm

INDOPRONOS was developed using several Java classes that were based on the existing SDN-IP Java classes (IntentSyncrhonizer), ONOS interface (RouteListener), and ONOS services (BgpSessionManager and IntentService). The INDOPRONOS application acts to alter the normal SDN-IP operation. The basic algorithm of the proposed application is shown in Figure 5-4.

INDOPRONOS detects the BGP Communities attribute inside a BGP UPDATE message. The first check is performed to match the AS number in the first two octets to an authorised AS number (pre-provisioned based on SLA). If no match occurs, no action is taken. If a match is found, then the second check occurs using the attribute VALUE octets. The VALUE is matched with the action list entries, as shown in Table 5-1. The application performs a different logic process as shown in Figure 5-4. If no match occurs, no action is taken. If a match is found, INDOPRONOS will install new intents to alter the SDN-IP Intents, but if no match is found, INDOPRONOS will widthdraw its intents from ONOS controller.

When INDOPRONOS install its new intents, it will carry out the provisioning action by installing new flow rules in the OpenFlow switch. While, when INDOPRONOS

widthdraw its new intents, it also will carry out the provisioning action by deleting the previous flow rules in the OpenFlow Switch. The full code scripts of IndopronosConnectivityManager.java and Indopronos.java are presented in the Appendix.

5.4.3. Implementation Test Bed

A test bed environment incorporating the implementation of the automatic multi-domain SDN provisioning framework was built to do testing. The prototype host was setup within a Ubuntu Virtual Machine (VM) with Mininet and the Quagga Software Routing Suite. The basic configuration was developed from the topology described in Section 4.4.1. A description of how the prototype was used to test the framework in an existing BGP network is provided in this section.

Table 5-1 SLA Example

AS Number	VALUE	Route Definition	SLA Actions
65011	100	Alternate Route (port 6)	Change to Premium Link
65011	Other	Default Route (port 5)	Change to Common Link
Other	100	Default Route (port 5)	Change to Common Link
Other	Other	Default Route (port 5)	Change to Common Link

The scenarios used for testing are focused on how the INDOPRONOS application is used to implement the multi-domain SDN provisioning framework. Therefore, it will be observed from the *Receiver* side, while on the *Sender* side, a regular BGP router is used to send the BGP UPDATE message which contains Communities and Extended Communities attributes during testing.

As mentioned in Section 5.2.1, specific connectivity arrangements between domain administrators are typically agreed upon and listed in an SLA to enable the zero-touch or automated process. An example of a connectivity arrangement that could be included in

the SLA is presented in Table 5-1, and it was used during the testing. The *Sender* domain passes the values in Table 5-1 to the *Receiver* domain inside the BGP Communities and Extended Communities attributes.

The ONOS controller was installed in the Ubuntu VM, and the Quagga [40] software was used as the BGP Speaker. An OpenFlow Virtual Switch (OVS) [41] was installed in the Ubuntu VM and a connection to the switch was made using the loopback interface. The Linux containers were used to create other domains which were represented by Quagga based BGP routers, as shown in Figure 5-5 Test Bed Implementation using ONOSFigure 5-5.

Based on the SLA example in Table 5-1, the framework utilises the SDN-IP application inside the *Receiver* domain to connect the *Sender* domain to the target network via the default route, i.e. BGP Router 2 via port number 5 of the OpenFlow switch. This situation occurred if the Communities value is not 100. If the *Sender* domain sends Communities value of 100 inside both Communities and Extended Communities attribute, the framework will utilize the INDOPRONOS application inside the *Receiver* domain to create two point-to-point intents, which will create an alternate route in the *Receiver* domain, i.e. forward the traffic flows to BGP Router 3 via port number 6 of the OpenFlow switch. The framework was tested using both scenarios.

The framework includes a BGP UPDATE message monitoring function for messages sent from the *Sender* to the *Receiver* that examines the Communities and Extended Communities attribute value. If the values do not match an action entry in the list, the process sets the default connectivity. If an action entry match is found in the list, the process will alter the default connectivity to that specified by the action entry.

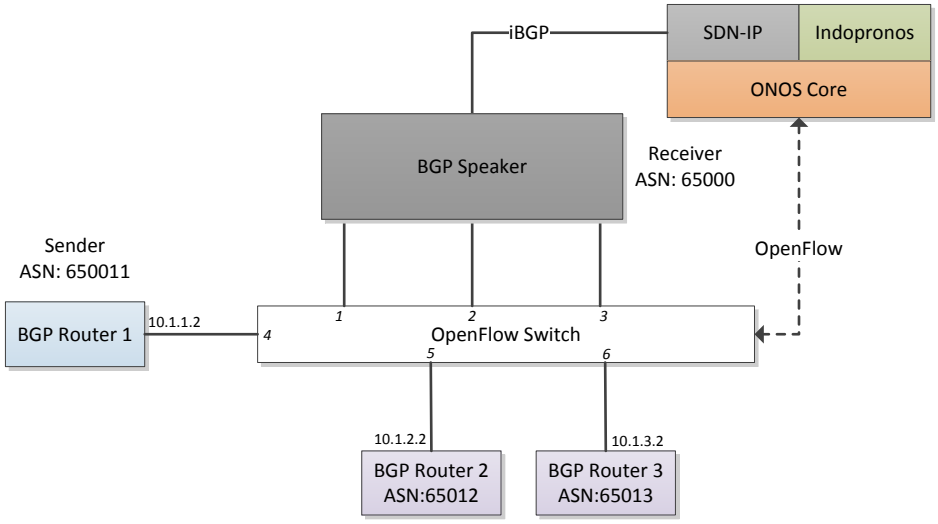


Figure 5-5 Test Bed Implementation using ONOS

5.5. Test Scenarios

There were two scenarios were developed to test the framework. Those scenarios were used to verify the behaviour of the framework implementation and test the framework using a simple Bandwidth on Demand use case. The list of virtual devices used in both scenarios is presented in Table 5-2.

Table 5-2 Test Bed Components

Virtual Devices	Networking	Remarks
ONOS Controller	Loopback (lo) interface	Running inside the Ubuntu VM with SDN- IP and INDOPRONOS application activated
BGP Speakers	loopback (lo) interface 10.1.1.0/24 to BGP Router 1 10.1.2.0/24 to BGP Router 2 10.1.3.0/24 to BGP Router 3	Using Quagga and it is connected to ONOS Controller with iBGP. Together with the ONOS controller and OpenFlow switch, they act as *Receiver* domain.
BGP Router 1	Peered with BGP Speakers with 10.1.1.0/24 network. It is the gateway to the 192.168.1.0/24 network.	Act as the *Sender* domain established the eBGP session with *Receiver* domain.
BGP Router 2	Peered with BGP Speakers with 10.1.2.0/24 network. The default gateway of the 192.168.2.0/24 network.	Act as the default gateway to test the provisioning. It connects the 192.168.2.0/24 network with a common link.
BGP Router 3	Peered with BGP Speakers with 10.1.3.0/24 network. Alternate gateway of 192.168.2.0/24 network	Act as the other gateway to test the network. It connects to BGP Router 4.

Virtual Devices	Networking	Remarks
BGP Router 4	Private BGP Router of 192.168.2.0/24 network.	BGP Router 4 connects the 192.168.2.0/24 network with a premium link.

5.5.1. Scenario A: Framework Verification

Scenario A is the test scenario used to verify the framework with the ONOS controller implementation. The framework was used for an end-to-end test using the test bed. As shown in Figure 5-6, the topology used in this scenario consists of three routers, one BGP Speaker, an OpenFlow switch, and an ONOS controller. The first BGP router is the *Sender* BGP router, which will send BGP Communities and Extended Communities Attributes to request the provisioning action. The BGP Speaker, OpenFlow switch, and ONOS controller play a role as the *Receiver*. Inside the ONOS controller, the INDOPRONOS and SDN-IP applications were installed and activated. The Scenario A test procedure is explained as follow:

a. BGP Router 1 establishes a connection with the BGP Speaker and sends the Communities and Extended Communities Attributes, within a BGP UPDATE message, each with value 65011:200. Where 65011 is the AS Number of the *Sender* domain, while 200 is an agreed value, between *Sender* and *Receiver*, for a specific action. In the following scenarios, this value will be used to indicate that the network should use a common link to forward traffic from *Sender*.

b. The BGP Speaker receives the value and advertises the value to the ONOS controller. The INDOPRONOS application within the ONOS controller detects the value of the attribute. The SDN-IP application installs the multi-point to single-point intents into the OpenFlow switch. The intents will facilitate the default routing to port five where the BGP Router 2 is connected.

c. BGP Router 1 is set to send another BGP UPDATE message with Communities value of 100.

d. BGP Speaker received the update and advertised the new value to the ONOS controller. The INDOPRONOS application within the ONOS controller detects the value and installs the new intents to override the intents previously installed by the SDN-IP application.

e. Wireshark was used to check the message contents and to check the intents installed in the ONOS controller. The monitoring process occurs when BGP Router 1 sends a BGP UPDATE message to the BGP Speaker.

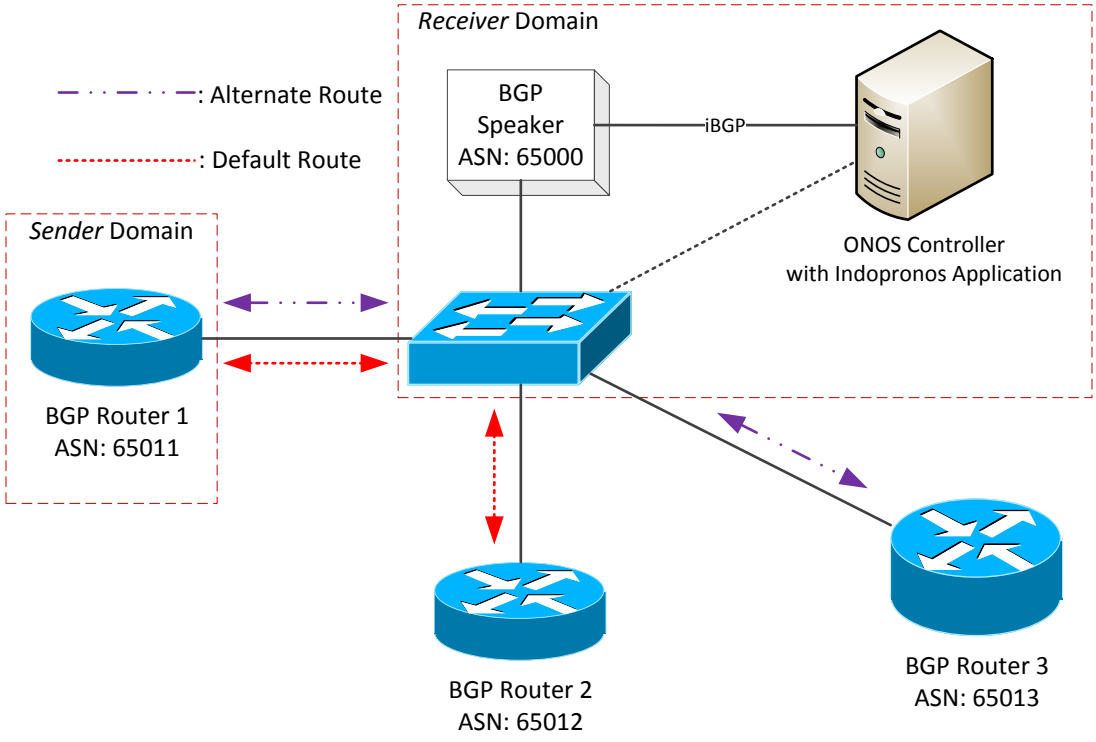


Figure 5-6 Topology for Scenario A.

5.5.2. Scenario B: Bandwidth on Demand Use Case

The second scenario was developed based on a Bandwidth on Demand use case. This topology is used to observe the impact of the proposed framework for a simple use case. In [102], a use case from an operator was observed, which uses the traffic captured from

two enterprise customers who subscribed to an IP Transit service. The study explored the captured traffic and discovered an over-provisioning phenomenon which will lead to a reduction of the network operation efficiency.

Scenario B will represent a Bandwidth on Demand use case of a typical network service. In this use case, a customer is connected to the SP network and traffic is routed to a remote network. There are two links that can be used by this customer, as it also subscribed to an On Demand (OD) service. The OD service enables the customer to automatically switch between the two links and is eventually charged based on traffic utilisation over each link. Therefore, the customer does not have to subscribe to a premium link all of the time and can subscribe to the premium link when it is needed.

Table 5-3 Links Specifications

Links	Maximum Bandwidth	Link Latency
Common Link	10 Mbit/second	15 milliseconds
Premium Link	20 Mbit/second	5 milliseconds

In this scenario, the use case is implemented in the network topology shown in Figure 5-7. The topology consists of two domains, the Customer domain and SP domain. Network A can send and receive data to Network B, which can be reached via a common link or premium link. The link specifications are presented in Table 5-3. The customer can switch between the two links without SP administrator intervention as it is subscribed to the OD service.

The default route for the common link is via BGP Route 2. This route is the default route used by Network A to reach Network B. The alternate route for the premium link is via BGP Route 3. This route will not be chosen by the transit network under default conditions as it has an additional hop to reach Network B, i.e. BGP Router 4.

A network operator (Network A) has subscribed to a network service with the SP network

using the SLA Example list shown in Table 5-1. Therefore, the customer domain administrator can switch the link between the premium link and common link without having to wait for the SP domain administrator to manually implement the provisioning request.

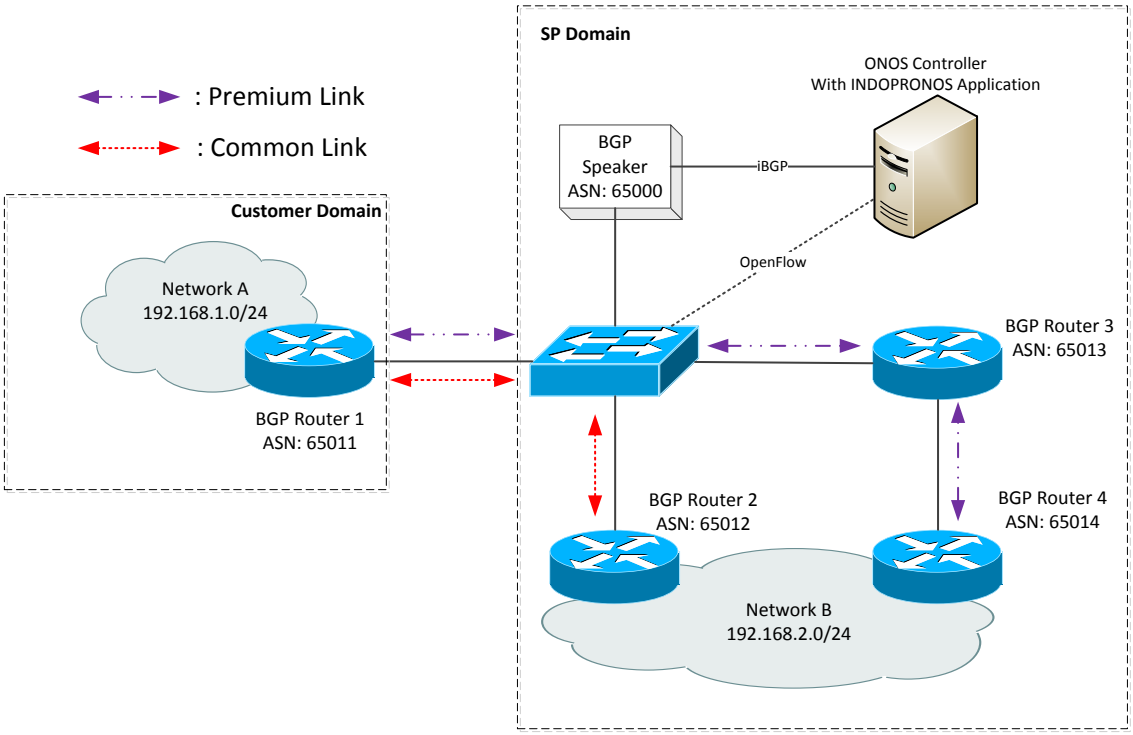


Figure 5-7 Topology for Scenario B.

The Scenario B can be described as follows:

a. Initially, the same as Scenario A, BGP Router 1 establishes a connection with BGP Speaker in the SP domain using eBGP. During this step, a BGP UPDATE message was used to send Communities value of 200 in both Communities and Extended Communities Attributes.

b. Inside the transit network, the BGP Speaker will advertise the BGP Communities Attribute value to the local ONOS controller. The INDOPRONOS application within the ONOS controller will detect the attributes value, and the SDN-IP application installs the multi-point to single-point intents into the OpenFlow enabled switch. The

intents provision the default routing to Network B via BGP Router 2.

c. When the Customer Domain administrator identifies the need to use the premium link, BGP Router 1 was used to send a BGP UPDATE message to the transit network with a Communities value of 100 in both Communities and Extended Communities Attributes.

d. Inside the SP Domain, the BGP Speaker will advertise the attribute value to the local ONOS controller. The new value is detected by the INDOPRONOS application within the ONOS controller, and it creates the new intents that will be used to override the intents installed in the OpenFlow enabled switch. The new intents will facilitate the alternate route towards Network B via BGP Router 3 and 4, which is the premium link.

e. Two network applications, i.e. fping and iperf were used to measure the Round-Trip Time (RTT) and throughput bandwidth with packet loss from Network A to Network B to check network performance before and after the automated provisioning.

5.5.3. Scenario C: Non-Bandwidth on Demand Subscriber

The third scenario was developed on top of the Bandwidth on Demand use case, with the addition of another domain. The scenario was focused how the provisioning framework can only work on the specific network, i.e. the subscribed network. This scenario is taken from the latest conference paper presented in Section 1.5. Point f.

The test scenario topology, shown in Figure 5-8, included two enterprise customer networks, Network A and Network C, an ISP network running INDOPRONOS application and the destination network, i.e. Network B. There are two routes provisioned to reach Network B, which has the same link specification of previous scenario, as shown

in Table 5-3.

It is assumed that the Network A customer is subscribed to a Bandwidth on Demand service with an SLA in place. An example of what might be contained in the SLA is shown in Table 5-1. Network C is not subscribed to this service. Network A can request the link be switched to the premium path to gain additional link capacity on demand.

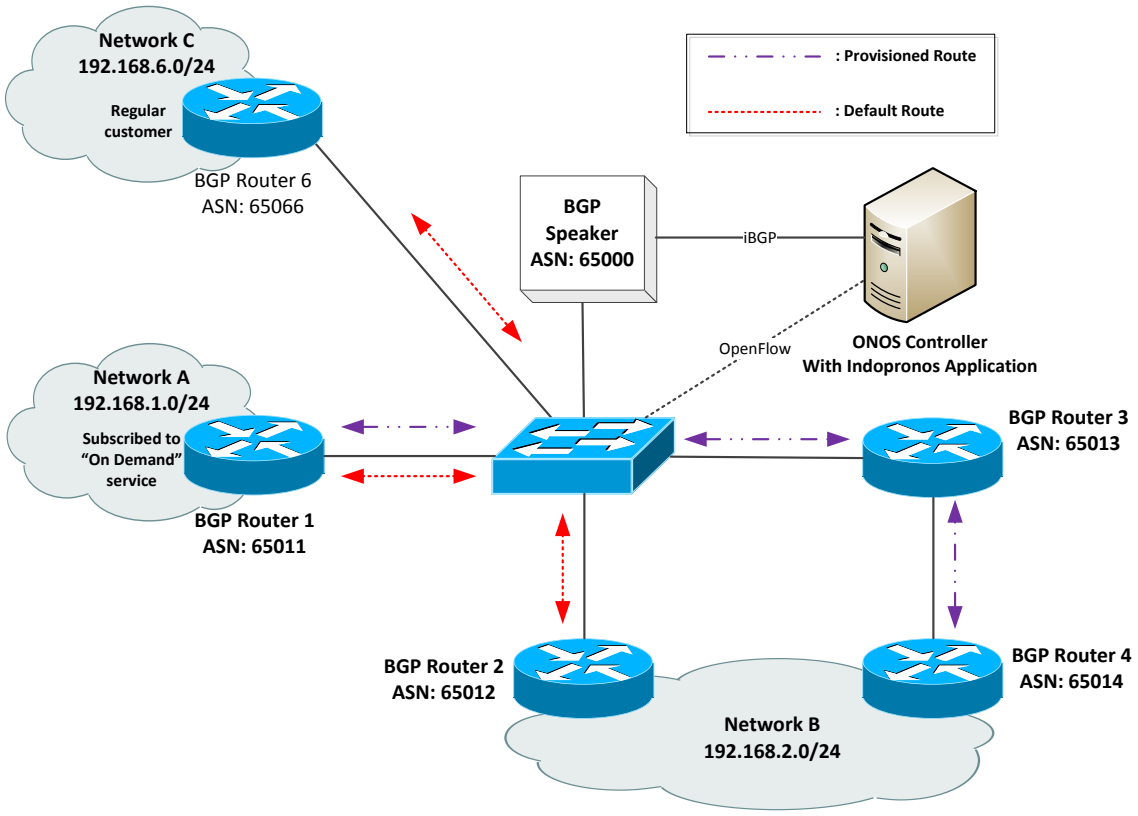


Figure 5-8 Topolgy for Scenario C.

The scenario C can be described as follows:

a. Initially, all BGP Routers establish connections with the ISP BGP Speaker using eBGP. During this step, a BGP UPDATE message was used to send the BGP Communities and Extended communities attributes value of 65011:200 from the Network A and Network C BGP Routers to the ISP network.

b. Inside the ISP network, the BGP Speaker advertises the BGP Communities Attribute

value to the local ONOS controller. The INDOPRONOS application within the ONOS controller detects the attributes value and the SDN-IP application installs the multi-point to single-point intents into the OpenFlow enabled switch. The intents provision the common routing to Network B via BGP Router 2.

c. For the next step, Network A and Network C's BGP Routers send BGP UPDATE messages to the ISP network with the attributes of 65011:100 and 65066:100 respectively.

d. Inside the ISP network, the BGP Speaker will advertise the new attribute values to the local ONOS controller. INDOPRONOS acts based on the attribute values received.

e. Network A and Network C's BGP Routers send BGP UPDATE messages again to ISP network with the attributes of 65011:200 and 65066:200 respectively.

f. Iperf, a simple network performance measurement tool, was used to measure link bandwidth, jitter and packet loss from Network A and Network C to Network B. The iperf performance measurement was carried out by transmitting 1470 bytes User Datagram Protocol (UDP) packets at the speed of 15 Mbps.

5.6. Results

5.6.1. Scenario A Results

In the first scenario, the ONOS CLI and ONOS Graphical User Interface (GUI) were used to testing the framework. The first check carried out tested the framework capability to exchange the Communities values from the *Sender* to the *Receiver* and the second check examined the state of the Intents list in the ONOS controller.

The first check was done to ensure that the changes in Communities attributes can be retrieved by the framework. Using the same command described in Section 4.4.1, the communities values sent from the *Sender* domain were checked. As shown in Figure 5-9, the framework can retrieve the initial value sent, i.e., 200. When the Communities value was changed to 100, it also changed, and when the value is changed back to 200 by the *Sender*, the command successfully verifies that the framework retrieved the latest value.

```
onos> bgp-routes -c
 IPv4 Network       BGP Community           BGP Extended Community
192.168.1.0/24          65011:200                        3:65011:200


 IPv6 Network       BGP Community           BGP Extended Community
onos> bgp-routes -c
 IPv4 Network       BGP Community           BGP Extended Community
192.168.1.0/24          65011:100                        3:65011:100


 IPv6 Network       BGP Community           BGP Extended Community
onos> bgp-routes -c
 IPv4 Network       BGP Community           BGP Extended Community
192.168.1.0/24          65011:200                        3:65011:200


 IPv6 Network       BGP Community           BGP Extended Community
onos>
```

Figure 5-9 Verification of the Communities Values Exchanged between Domains.

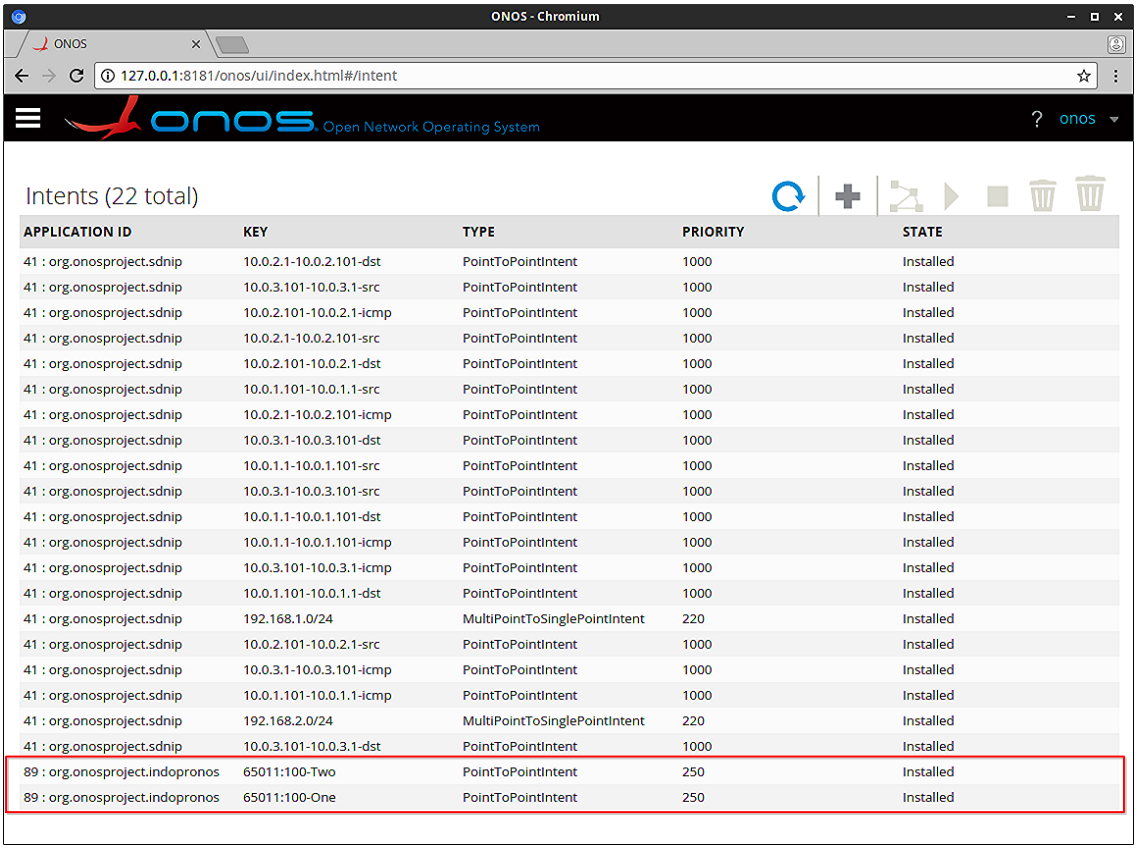
The second check was done to ensure that the framework can install the *point-to-point intents* when they are required. The intents installed by the framework were verified using the GUI, as shown in Figure 5-10. When the *Sender* transmits the initial BGP UPDATE message with Communities value of 200, the ONOS controller installed *multi-points to single point intents* from the SDN-IP application for the default route. When the *Sender* transmits the Communities values of 100, we observed the ONOS controller installed new *point-to-point intents*. The intents were generated by the INDOPRONOS application and had a higher priority value to the previous *multi-points to single point intents*, therefore overriding them. The new intents were used to set the traffic route to the target network

via port 6 of the OpenFlow enabled switch, as defined in Table 5-1. As the *Sender* transmit the Communities values of 200 again, the ONOS controller withdraws the *point to point intents*. Therefore the initial *multi-points to single-point intents* are now valid.

The scenario A test results confirmed that the multi-domain SDN automatic provisioning framework had accomplished the objectives as follows:

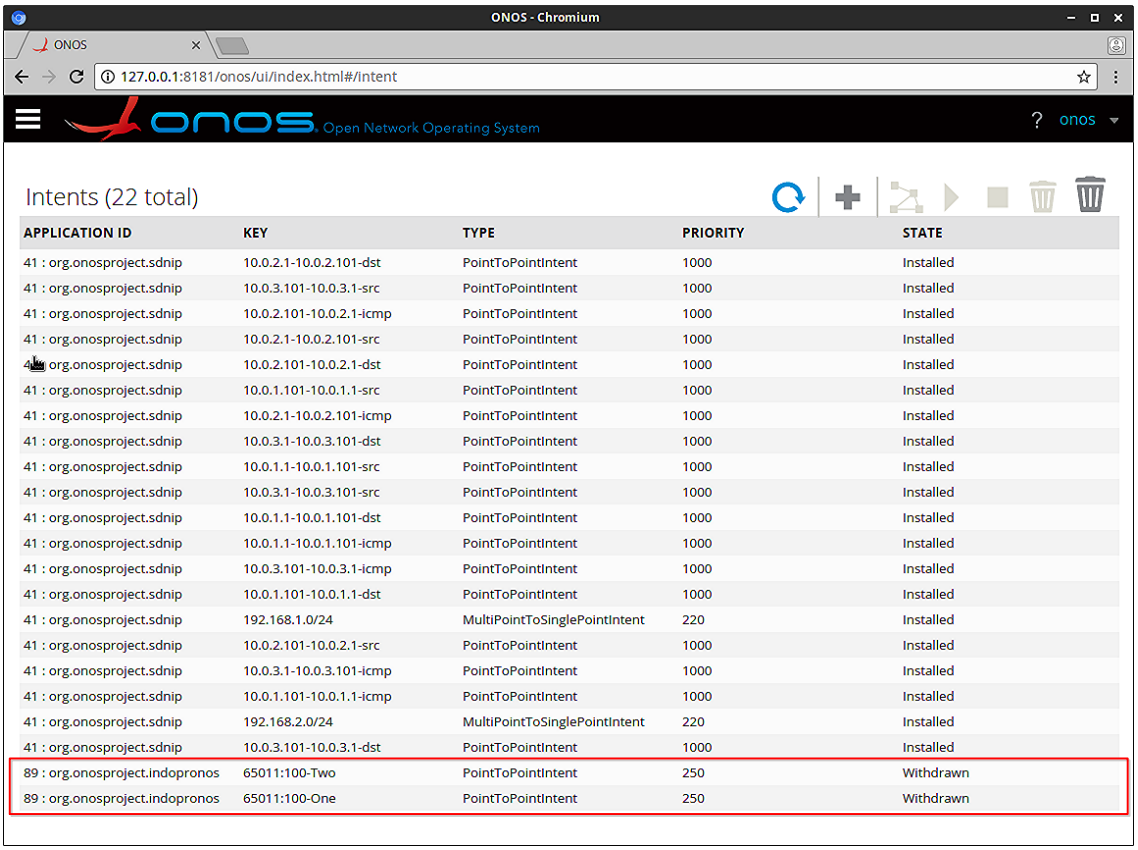
a. The framework successfully applied the inter-domain SDN communication protocol to implement the information exchange mechanism. The BGP Communities and Extended Communities attributes can be utilised to send information to trigger a pre-defined application on the ONOS Controller.

b. The framework successfully conducted the automatic provisioning, by automatically overriding the default behaviour, triggered by the correct information sent by the other domain.



APPLICATION ID	KEY	TYPE	PRIORITY	STATE
41 : org.onosproject.sdnip	10.0.2.1-10.0.2.101-dst	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.3.101-10.0.3.1-src	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.2.101-10.0.2.1-icmp	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.2.1-10.0.2.101-src	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.2.101-10.0.2.1-dst	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.1.101-10.0.1.1-src	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.2.1-10.0.2.101-icmp	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.3.1-10.0.3.101-dst	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.1.1-10.0.1.101-src	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.3.1-10.0.3.101-src	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.1.1-10.0.1.101-dst	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.1.1-10.0.1.101-icmp	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.3.101-10.0.3.1-icmp	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.1.101-10.0.1.1-dst	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	192.168.1.0/24	MultiPointToSinglePointIntent	220	Installed
41 : org.onosproject.sdnip	10.0.2.101-10.0.2.1-src	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.3.1-10.0.3.101-icmp	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.1.101-10.0.1.1-icmp	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	192.168.2.0/24	MultiPointToSinglePointIntent	220	Installed
41 : org.onosproject.sdnip	10.0.3.101-10.0.3.1-dst	PointToPointIntent	1000	Installed
89 : org.onosproject.indopronos	65011:100-Two	PointToPointIntent	250	Installed
89 : org.onosproject.indopronos	65011:100-One	PointToPointIntent	250	Installed

a) View of Intents when Communities and Extended Communities values are changed to 100



APPLICATION ID	KEY	TYPE	PRIORITY	STATE
41 : org.onosproject.sdnip	10.0.2.1-10.0.2.101-dst	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.3.101-10.0.3.1-src	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.2.101-10.0.2.1-icmp	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.2.1-10.0.2.101-src	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.2.101-10.0.2.1-dst	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.1.101-10.0.1.1-src	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.2.1-10.0.2.101-icmp	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.3.1-10.0.3.101-dst	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.1.1-10.0.1.101-src	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.3.1-10.0.3.101-src	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.1.1-10.0.1.101-dst	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.1.1-10.0.1.101-icmp	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.3.101-10.0.3.1-icmp	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.1.101-10.0.1.1-dst	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	192.168.1.0/24	MultiPointToSinglePointIntent	220	Installed
41 : org.onosproject.sdnip	10.0.2.101-10.0.2.1-src	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.3.1-10.0.3.101-icmp	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	10.0.1.101-10.0.1.1-icmp	PointToPointIntent	1000	Installed
41 : org.onosproject.sdnip	192.168.2.0/24	MultiPointToSinglePointIntent	220	Installed
41 : org.onosproject.sdnip	10.0.3.101-10.0.3.1-dst	PointToPointIntent	1000	Installed
89 : org.onosproject.indopronos	65011:100-Two	PointToPointIntent	250	Withdrawn
89 : org.onosproject.indopronos	65011:100-One	PointToPointIntent	250	Withdrawn

b) View of Intents when Communities and Extended Communities values are changed back to 200

Figure 5-10 Verification of New Installed Intents.

5.6.2. Scenario B Results

In the second scenario, the framework was tested with the Bandwidth on Demand use case. In this use case, Network A (192.168.1.0/24) is subscribed to a network service to reach Network B (192.168.2.0/24), and the SLA example list shown in Table 5-1 is used. The SLA agreement permits the administrator of Network A (customer domain) to change the route to Network B by sending a BGP UPDATE message with the appropriate Communities value.

The fping tool was used to send ICMP packets from a Network A host with IP address 192.168.1.10, to the target Network B host, with IP Address 192.168.2.10. During the test, BGP Communities Attributes were sent from the *Sender* to the *Receiver*, and the ping traffic was monitored. The fping tool RTT result is shown in Figure 5-11.

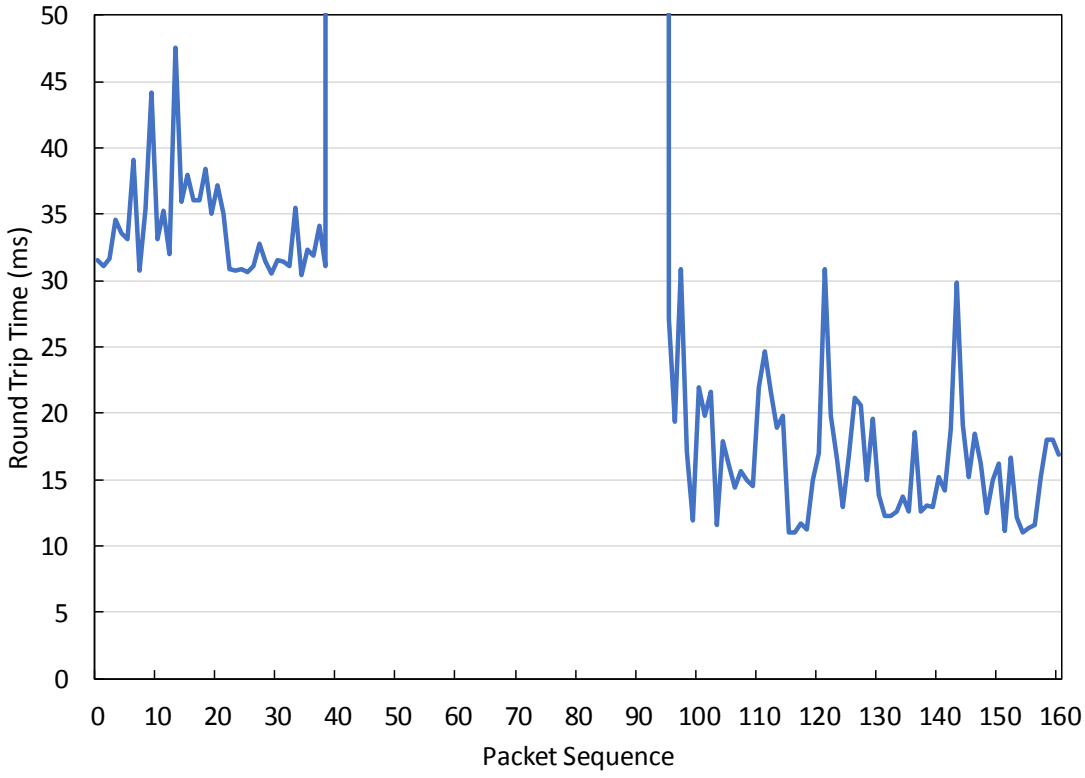


Figure 5-11 RTT Results from Network A to Network B.

The observation showed that the network behaved as expected. As the premium link has a lower latency when compared to the common link, the measurement tools showed that the common link RTT average value is 33.92 milliseconds, while the premium link RTT average value is 16.65 milliseconds. The results presented show that the traffic behaved as expected. The RTT is higher when traffic traverses through the common link and lower when the traffic traverses through the premium link. We can also observe that the time taken for the transit network to reroute traffic was 56 seconds. This was the time used by BGP Route 1 and the transit network for BGP convergence, and intents installation in the OpenFlow enabled switch. This time is not unusual for BGP updates.

The iperf performance measurement was carried out by transmitting 1470 byte User Datagram Protocol (UDP) packets at 10 Mbps and 20 Mbps. As described in the previous section, the common link was provisioned with 10 Mbps throughput, while the premium link was provisioned with 20 Mbps throughput. The throughput bandwidth and packet loss measurement results are shown in Figure 5-12 and Figure 5-13respectively.

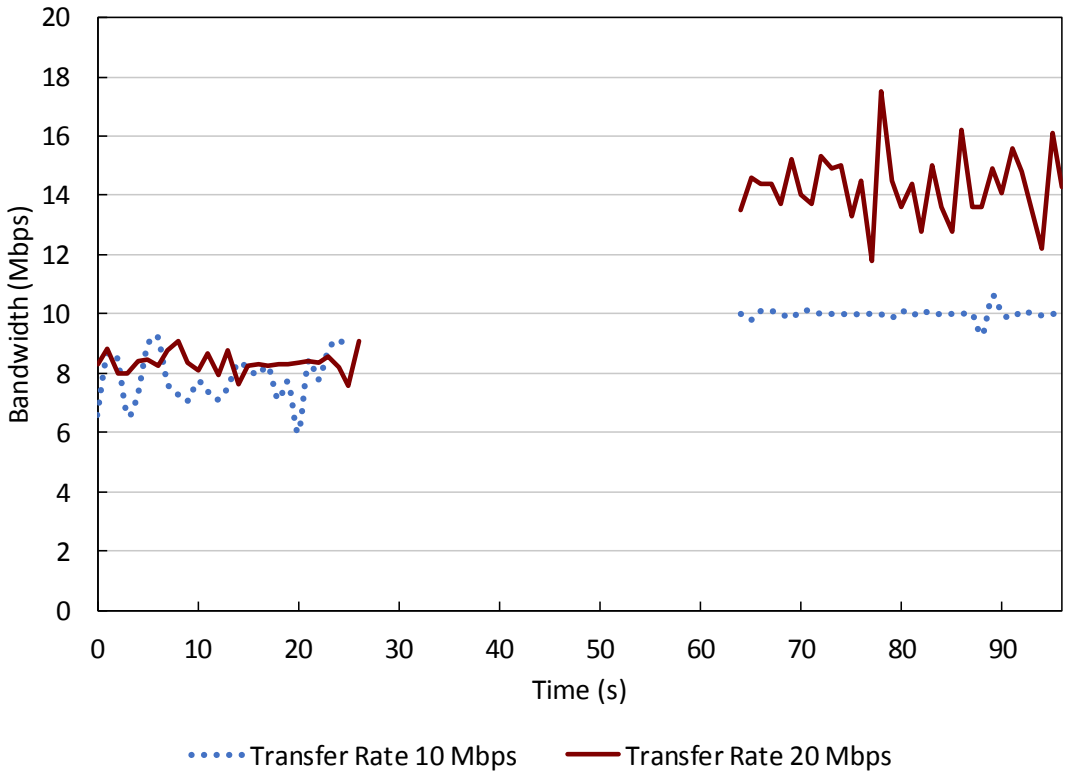


Figure 5-12 Throughput Bandwidth Measured from Network A to Network B.

When the data transfer occurred using a 10 Mbps transfer rate, the route change provided a marginal performance improvement. From the iperf tool data, we noted the common link average throughput bandwidth was 7.85 Mbps, and the premium link was 9.99 Mbps. While the packet loss for the common link was 16.01% and the premium link was 0.01%. The results also demonstrate how over-provisioning can meet the agreed service level, but with a higher network cost.

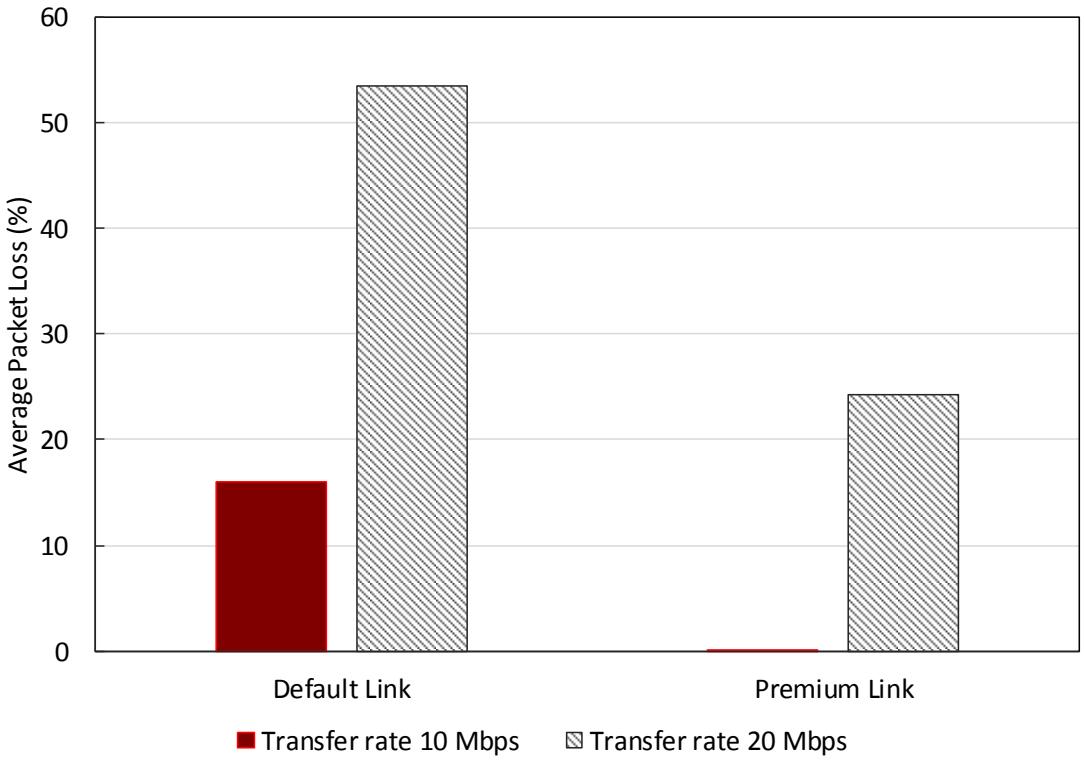


Figure 5-13 Average Packet Loss

When the data transfer occurred using a 20 Mbps transfer rate, the route change provided a significant performance improvement, again as would be expected given the default route provides a throughput of 10 Mbps. Iperf tool measurement result data shows that the common link average throughput bandwidth was measured 8.35 Mbps, and the premium link was 14.28 Mbps. The packet loss for the common link was 53.44%, and the premium link was 24.24%. The 20 Mbps transfer rate testing result showed that when Network A was using the common link, traffic from Network A to Network B was significantly affected and more than half of the packets were lost. When the link is changed to the premium link, the average throughput bandwidth did not reach the maximum 20 Mbps as expected. It could be caused by a bottleneck in the links between the routers. As the data transfer rate was higher than the average throughput bandwith of the premium link, the packet loss performance was still considered to be high, although it has been reduced significantly from the traffic on the common link. The ability for the

Network A administrator to shift traffic to the premium link improved network performance.

These results show several keypoints, i.e.:

a. Network A can change the flow entries in the Service Provider's OpenFlow Switch, by sending the correct BGP UPDATE message. This ability shows that Network A can conduct an automatic provisioning in other network domain, in this case within the Service Provider's network.

b. INDOPRONOS application built within ONOS controller can support the multi-domain provisioning and correctly behave according to the prior setup. Therefore, the test bed could demonstrate the ability of automatic multi-domain provisioning using SDN.

5.6.3. Scenario C Results

In the third scenario, the observation was focused on how the test bed reacted when there was another domain introduced, i.e. Network C. Network C was not subscribed to the Bandwidth on Demand service, but it can send the correct format of the BGP UPDATE message, similar to Network A. The results in terms of link bandwidth can be observed in Figure 5-14. Network A successfully changed its route to the premium link to gain additional link capacity after sending an appropriately formatted BGP UPDATE message. Although Network C sent an appropriately formatted BGP UPDATE message, its link capacity has remained the same, because Network C is not subscribed to the Bandwidth on Demand service. The average throughput bandwidth of Network A was upgraded from 9.32 Mbps to 14.98 Mbps. While the average throughput bandwidth of Network C was 8.12 Mbps initially, and 9.03 Mbps after it sent the BGP UPDATE message. The results showed that Network A gained significant link capacity due to a service change to the

premium link and the Network C improvement occurred but was not significant, which could be the result of the absence of Network A traffic from common link.

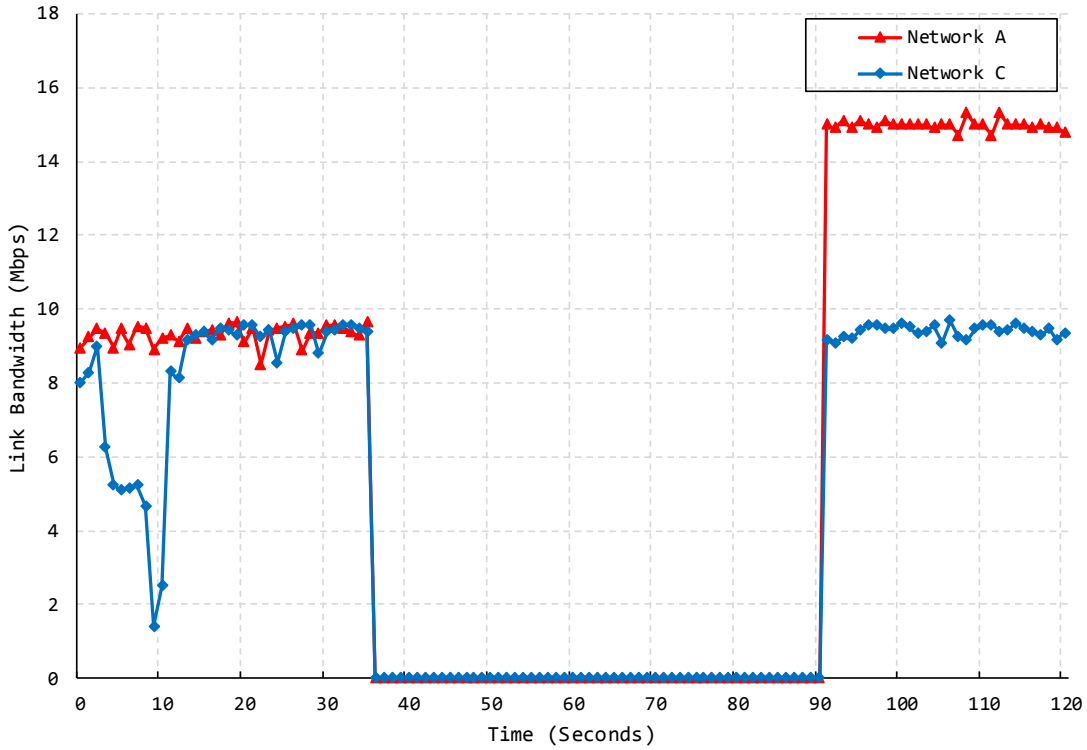
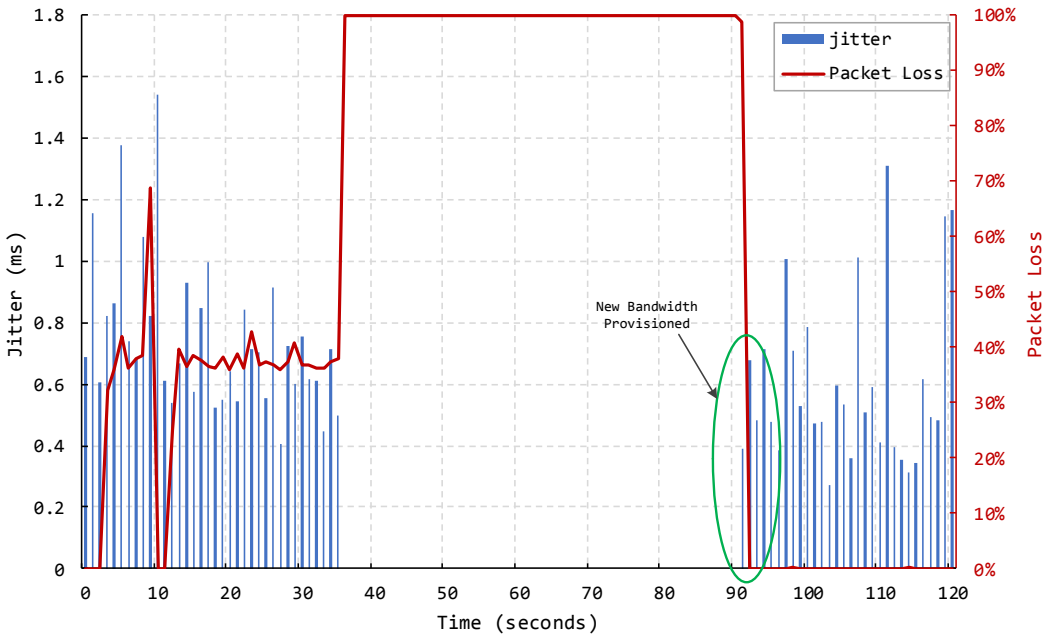
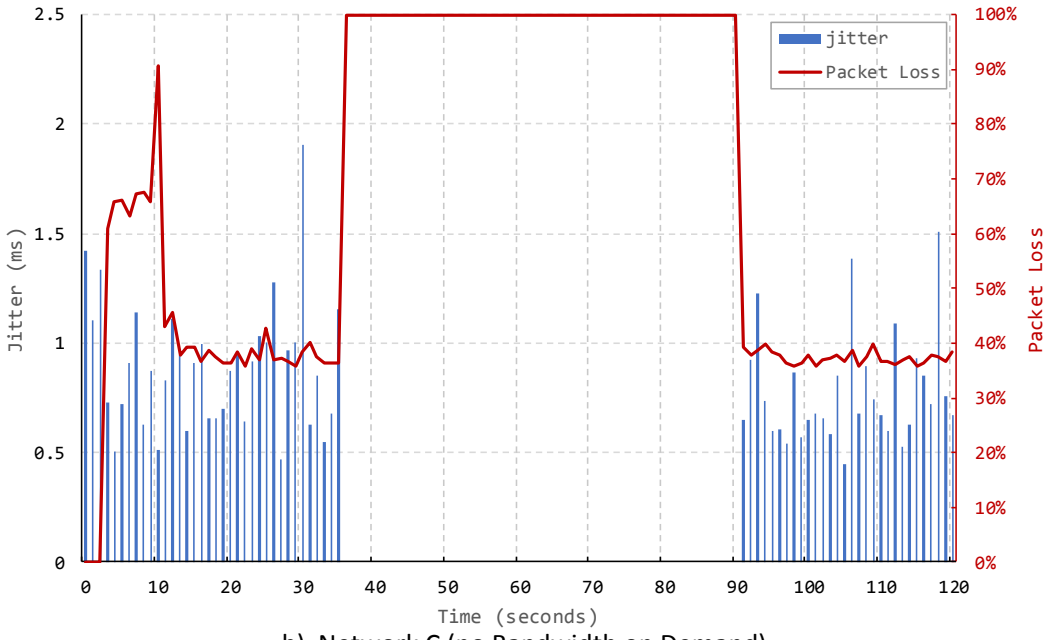


Figure 5-14 Troughput Bandwidth towards Network B

The jitter and packet loss performance results show similar behavior. As shown in Figure 5-15, Network A successfully changed to the premium link which changed the jitter and packet loss performance, while the Network C performance remained the same. The average jitter for Network A was reduced from 0.74 ms to 0.60 ms, while the Network C average jitter was 0.89 initially, and 0.77 ms after the BGP UPDATE messages. The jitter performance was not improved significantly in this experiment, due to the link configuration only defining the link delay and link bandwidth. In terms of packet loss, Network A reduced its average packet loss from 33% down to 3%, while the Network C average packet loss was 42% initially and 39% after the BGP UPDATE messages.



a) Network A (with Bandwidth on Demand)



b) Network C (no Bandwidth on Demand)

Figure 5-15 Jitter vs Packet Loss

It was observed that the network behaved as expected. As the premium link has a higher bandwidth and lower latency when compared to the common link, the performance of Network A improved significantly. It was also observed that the time taken for the transit network to reroute traffic was 55 seconds. This was the time used for BGP convergence

and the ONOS application processes, after BGP Router 1 and BGP Router 6 used the BGP sessions to send BGP UPDATE message to the ISP network.

These results show several keypoints, i.e.:

a. As observed also in previous scenario, Network A can conduct automatic provisioning in another network domain. On the other hand, Network C was not allowed to make any changes to the flow entries in the Service Provider's OpenFlow Switch, although it also sent the correct format of a BGP UPDATE message.

b. The INDOPRONOS application built within an ONOS controller in this test bed demonstrated the ability to match the correct combinations of AS numbers and Communities and Extended Communities values. This ability confirms that the INDOPRONOS application can detect the network domains that are authorised to carry out multi-domain provisioning and behave according to the agreed setup in the different administrative domains. Therefore, the test bed was used to successfully demonstrate automatic multi-domain provisioning using SDN.

5.7. Conclusion

The Automatic Provisioning can be implemented by applying Zero-touch provisioning approach. The Zero-touch provisioning objective is to limit or diminish the human intervention in network operational tasks. As demonstrated by the IP address assigned by the DHCP server, the automatic provisioning requires message exchange mechanism between parties and pre-defined parameters and actions.

In multi-domain SDN, the basic process for automatic provisioning is the information exchange between SDN domains and the flow table content modification based on the

other domain request. The Communities and Extended Communities attributes are the media the enables the information exchange between the interconnected SDN domains using BGP. An application will be triggered by the information carried inside the Communities value to conduct the provisioning request and modify the flow table in the OpenFlow switch inside the SDN domain.

A framework for automatic provisioning in multi-domain SDN was proposed and implemented using the ONOS controller. A new application called INDOPRONOS was developed and acts based on the Communities value exchanged between SDN domains. The framework implementation in ONOS was tested using a basic scenario to verify its behaviours and a use case to test its effect on the network performance of the interconnected domains.

The test results show that the ONOS controller has been successfully used as an implementation platform for multi-domain SDN automatic provisioning. The performance limitation of this framework was caused by the fundamental limitation of BGP as its underlying protocol, which is the convergence time. The time measured for the switching between the common link to the premium link and vice versa is quite noticeable, but not unexpected. This convergence time would be a great challenge for the future research into improving the automatic provisioning framework.

Chapter 6 CONCLUSION AND FUTURE WORK

6.1. Conclusions

This research aims and objective provided in Chapter 1 have been successfully answered and contribute to the body of knowledge in the area of multi-domain SDN communication. A novel framework for automatic provisioning in multi-domain SDN has been presented, which will enhance efficiency and flexibility in managing WAN operation.

A comprehensive review of SDN with its challenges and controller designs has been presented. Classification of approaches to cope with the scalability challenges was discussed thoroughly, with the focus on the design consideration on implementing SDN in WAN. An extensive review on current border gateway protocols that used for inter-domain SDN communication was presented, which motivate the use of BGP as preferred baseline protocol for inter-domain SDN communication.

The requirement for inter-domain SDN communication methods and how BGP can support that requirement had been discussed and presented. BGP could support the capability and reachability information exchange needed in inter-domain SDN communication, and it also supports the backward interoperability for interconnection with non-SDN domains. Despite its lack of operational performance compared to a legacy multi-domain network, the use of BGP in multi-domain SDN offers the potential for improved flexibility and reduced network management interaction.

BGP UPDATE message was identified as the message that can carry the information needed for provisioning between SDN domains. The breakdown of BGP UPDATE messages can suggest the use of Path Attributes part of BGP. The selection process had decided the Communities, and Extended Communities could be used to transport the information from one SDN domain to the other SDN domain. This mechanism had been

successfully implemented on ONOS controller with SDN-IP application activated. The implementation required some modification in several software modules related to BgpSessionManager.

The framework design of automatic provisioning in multi-domain SDN used the information exchange mechanism as one of the core functions. The other core functions of this framework were the automatic provisioning procedures. The framework was implemented on top of ONOS controller with SDN-IP Application and a new application called INDOPRONOS was developed to add the automatic provisioning capability.

The framework basic test result showed the successful behaviour of the framework to conduct the two core functions. The bandwidth on demand use case was used to provide performance measurement results in terms of Round Trip Time, throughput Bandwidth and average packet loss.

6.2. Future Work

Multi-domain SDN communication is a very vast topic, and this research could suggest some research topics for future works, such as:

a. Improving the framework implementation by adding more flexibility to modify the pre-defined parameters and actions. Current framework implementation only supports static parameters embodied inside the script, which was fine for Proof of Concept stage. As the concept has been proved, the application needs to be enhanced to be prepared for real-life implementation

b. Improving the convergence time in the provisioning. The current research limitation came from the inheritance of BGP convergence time to the framework. BGP has always had problems with its convergence time, which can range from 10 seconds until more than 1 minute. Research on how to shorten the convergence time could

help in improving the provisioning performance towards zero-touch and real-time provisioning.

c. This research did not discuss the security factor of the framework. As the domains belong to different entities, security would be one of the main concerns to implement this framework. The authentication method and process between SDN domains are one of the primary challenges in the multi-domain SDN provisioning. Therefore, research on the security frameworks in the multi-domain SDN will become an interesting topic.

BIBLIOGRAPHY

[1] Cisco. (2017, June 6, 2017). *Cisco Visual Networking Index: Forecast and Methodology 2016-2021 (White Paper)*.

[2] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet routing convergence," *ACM SIGCOMM Computer Communication Review,* vol. 30, pp. 175-187, 2000.

[3] N. Feamster, H. Balakrishnan, and J. Rexford, "Some foundational problems in interdomain routing," in *Proceedings of Third Workshop on Hot Topics in Networks (HotNets-III)*, 2004, pp. 41-46.

[4] N. McKeown, "Software-defined networking," *INFOCOM keynote talk,* vol. 17, pp. 30-32, 2009.

[5] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine,* vol. 51, pp. 114-119, 2013.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford*, et al.*, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review,* vol. 38, pp. 69-74, 2008.

[7] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials,* vol. 17, pp. 27-51, 2015.

[8] B. A. A. Nunes, X.-N. Nguyen, T. Turletti, M. Mendonca, and K. Obraczka, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys and Tutorials,* vol. 16, pp. 1617-1634, 2014.

[9] V. Kotronis, A. Gämperli, and X. Dimitropoulos, "Routing centralization across domains via SDN: A model and emulation framework for BGP evolution," *Computer Networks,* vol. 92, Part 2, pp. 227-239, 12/9/ 2015.

[10] V. Kotronis, X. Dimitropoulos, and B. Ager, "Outsourcing the routing control logic: better internet routing based on SDN principles," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012, pp. 55-60.

[11] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri*, et al.*, "GENI: A federated testbed for innovative network experiments," *Computer Networks,* vol. 61, pp. 5-23, 2014.

[12] G. Carrozzo, R. Monno, B. Belter, R. Krzywania, K. Pentikousis, M. Broadbent*, et al.*, "Large-scale SDN experiments in federated environments," in *Smart*

Communications in Network Technologies (SaCoNeT), 2014 International Conference on, 2014, pp. 1-6.

[13] C. Fernandez, C. Bermudo, G. Carrozzo, R. Monno, B. Belter, K. Pentikousis*, et al.*, "A recursive orchestration and control framework for large-scale, federated SDN experiments: the FELIX architecture and use cases," *International Journal of Parallel, Emergent and Distributed Systems,* vol. 30, pp. 428-446, 2015.

[14] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri*, et al.*, "Achieving high utilization with software-driven WAN," in *ACM SIGCOMM Computer Communication Review*, 2013, pp. 15-26.

[15] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh*, et al.*, "B4: Experience with a globally-deployed software defined WAN," in *ACM SIGCOMM Computer Communication Review*, 2013, pp. 3-14.

[16] S. Nickolay, E.-S. Jung, R. Kettimuthu, and I. Foster, "Bridging the gap between peak and average loads on science networks," *Future Generation Computer Systems,* vol. 79, pp. 169-179, 2018.

[17] Y. Jarraya, T. Madi, and M. Debbabi, "A Survey and a Layered Taxonomy of Software-Defined Networking," *Communications Surveys & Tutorials, IEEE,* vol. 16, pp. 1955-1980, 2014.

[18] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE,* vol. 103, pp. 14-76, 2015.

[19] F. X. A. Wibowo and M. A. Gregory, "Updating guaranteed bandwidth in multi-domain software defined networks," in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, 2017, pp. 1-6.

[20] H. Farhady, H. Lee, and A. Nakao, "Software-Defined Networking: A survey," *Computer Networks,* vol. 81, p. 79, 2015.

[21] D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore*, et al.*, "The SwitchWare active network architecture," *Network, IEEE,* vol. 12, pp. 29-36, 1998.

[22] A. T. Campbell, I. Katzela, K. Miki, and J. Vicente, "Open signaling for ATM, internet and mobile networks (OPENSIG'98)," *SIGCOMM Comput. Commun. Rev.,* vol. 29, pp. 97-108, 1999.

[23] E. Haleplidis, J. Hadi Salim, J. M. Halpern, S. Hares, K. Pentikousis, K. Ogawa, *et al.*, "Network Programmability With ForCES," *Communications Surveys & Tutorials, IEEE,* vol. 17, pp. 1423-1440, 2015.

[24] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, *et al.*, "A clean slate 4D approach to network control and management," *ACM SIGCOMM Computer Communication Review,* vol. 35, pp. 41-54, 2005.

[25] J. Yu and I. Al Ajarmeh, "An Empirical Study of the NETCONF Protocol," in *2010 Sixth International Conference on Networking and Services (ICNS)* 2010, pp. 253-258.

[26] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, *et al.*, "SANE: a protection architecture for enterprise networks," presented at the Proceedings of the 15th conference on USENIX Security Symposium - Volume 15, Vancouver, B.C., Canada, 2006.

[27] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review*, 2007, pp. 1-12.

[28] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, and L. Peterson. (2012). *Software-defined networking: the new norm for network*.

[29] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, and T. Gayraud, "Software-Defined Networking: Challenges and research opportunities for Future Internet," *Computer Networks,* vol. 75, Part A, pp. 453-471, 12/24/ 2014.

[30] O. N. F. (ONF), "OpenFlow® Switch Specification Ver 1.5.1," ed, 2015.

[31] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Computer Networks,* vol. 72, pp. 74-98, 2014.

[32] M. Karakus and A. Durresi, "A Scalability Metric for Control Planes in Software Defined Networks (SDNs)," in *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, 2016, pp. 282-289.

[33] L. Shuhao and L. Baochun, "On scaling software-Defined Networking in wide-area networks," *Tsinghua Science and Technology,* vol. 20, pp. 221-232, 2015.

[34] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 7-12.

[35] P. Vizarreta, C. M. Machuca, and W. Kellerer, "Controller placement strategies

for a resilient SDN control plane," in *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*, 2016, pp. 253-259.

[36] A. Sallahi and M. St-Hilaire, "Optimal Model for the Controller Placement Problem in Software Defined Networks," *IEEE Communications Letters,* vol. 19, pp. 30-33, 2015.

[37] Y. Jimenez, C. Cervello-Pastor, and A. J. Garcia, "On the controller placement for designing a distributed SDN control layer," in *Networking Conference, 2014 IFIP*, 2014, pp. 1-9.

[38] T. A. Nguyen, T. Eom, S. An, J. S. Park, J. B. Hong, and D. S. Kim, "Availability Modeling and Analysis for Software Defined Networks," in *2015 IEEE 21st Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2015, pp. 159-168.

[39] Y. Jinke, W. Ying, P. Keke, Z. Shujuan, and L. Jiacong, "A load balancing mechanism for multiple SDN controllers based on load informing strategy," in *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2016, pp. 1-4.

[40] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Software-defined networking security: pros and cons," *IEEE Communications Magazine,* vol. 53, pp. 73-79, 2015.

[41] R. Kaur, A. Singh, S. Singh, and S. Sharma, "Security of software defined networks: Taxonomic modeling, key components and open research area," in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 2016, pp. 2832-2839.

[42] F. X. Wibowo, M. A. Gregory, K. Ahmed, and K. M. Gomez, "Multi-domain software defined networking: research status and challenges," *Journal of Network and Computer Applications,* vol. 87, pp. 32-45, 2017.

[43] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 13-18.

[44] Z. Cai, "Maestro: Achieving scalability and coordination in centralizaed network control plane," ProQuest, UMI Dissertations Publishing, 2011.

[45] *Project Floodlight*. Available: http://www.projectfloodlight.org/

[46] S. Ishii, E. Kawai, T. Takata, Y. Kanaumi, S.-i. Saito, K. Kobayashi*, et al.*, "Extending the RISE controller for the interconnection of RISE and

OS3E/NDDI," in *Networks (ICON), 2012 18th IEEE International Conference on*, 2012, pp. 243-248.

[47] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide*, et al.*, "ONOS: towards an open, distributed SDN OS," in *The third workshop on Hot topics in Software Defined Networking*, 2014, pp. 1-6.

[48] R. Kubo, T. Fujita, Y. Agawa, and H. Suzuki. (2014, 28/08/2014). Ryu SDN Framework - Open-source SDN Platform Software. *12*. Available: https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201408fa4.pdf&mode=show_pdf

[49] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown*, et al.*, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comp. Commun. Rev.,* vol. 38, pp. 105-110, 2008.

[50] A. Singla and B. Rijsman. (2013). *OpenContrail Architecture Document*. Available: http://www.opencontrail.org/opencontrail-architecture-documentation/

[51] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014.

[52] D. Saikia and N. Malik, "An Introduction to Open MUL SDN Suite," ed, 2015.

[53] (2014, 27/08/2015). The Real-Time Cloud. [White Paper]. Available: http://www.ericsson.com/res/docs/whitepapers/wp-sdn-and-cloud.pdf

[54] (2015, 26/08/2015). HP VAN SDN Controller Software. [Data Sheet]. Available: http://h20195.www2.hp.com/v2/getpdf.aspx/4AA4-9827ENW.pdf

[55] "IBM Programmable Network Controller V3.0," 03/10/2012 2012.

[56] (2013, Contrail Architecture. [White Paper]. Available: http://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000535-en.pdf

[57] (2012, 28/08/2015). Cisco Open SDN Controller 1.2. [White Paper]. Available: https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/open-sdn-controller/datasheet-c78-733458.pdf

[58] (2013, 4 September 2015). Huawei IP SDN Controller Open and Application. [Presentation]. Available: http://www.euchina-fire.eu/wp-content/uploads/2015/01/Huawei-IP-SDN-Controller-Open-and-

ApllicationV1-2.pdf

[59] A. Voellmy and J. Wang, "Scalable software defined network controllers," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, 2012, pp. 289-290.

[60] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 19-24.

[61] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain sdn controllers," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, 2014, pp. 1-4.

[62] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson, "Pratyaastha: an efficient elastic distributed SDN control plane," in *Proceedings of the third workshop on Hot topics in software defined networking*, Chicago, Illinois, USA, 2014, pp. 133-138.

[63] O. Michel and E. Keller, "SDN in wide-area networks: A survey," in *Software Defined Systems (SDS), 2017 Fourth International Conference on*, 2017, pp. 37-42.

[64] R. Ahmed and R. Boutaba, "Design considerations for managing wide area software defined networks," *Communications Magazine, IEEE,* vol. 52, pp. 116-123, 2014.

[65] Y. Zhang, L. Cui, W. Wang, and Y. Zhang, "A survey on software defined networking with multiple controllers," *Journal of Network and Computer Applications,* vol. 103, pp. 101-118, 2018/02/01/ 2018.

[66] B. Davie, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. Gude*, et al.*, "A database approach to sdn control plane design," *ACM SIGCOMM Computer Communication Review,* vol. 47, pp. 15-26, 2017.

[67] H. E. Egilmez, "Distributed QoS architectures for multimedia streaming over software defined networks," *Multimedia, IEEE Transactions on,* vol. 16, pp. 1597-1609, 2014.

[68] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *ACM SIGCOMM Computer Communication Review*, 2013, pp. 7-12.

[69] H. Yin, H. Xie, T. Tsou, D. R. Lopez, P. A. Aranda, and R. Sidi. (2012). *SDNi:*

A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains. Available: https://datatracker.ietf.org/doc/html/draft-yin-sdn-sdni-00

[70] P. Lin, J. Bi, and Y. Wang, "WEBridge: west–east bridge for distributed heterogeneous SDN NOSes peering," *Security and Communication Networks,* vol. 8, pp. 1926-1942, 2015.

[71] P. Helebrandt and I. Kotuliak, "Novel SDN multi-domain architecture," in *Emerging eLearning Technologies and Applications (ICETA), 2014 IEEE 12th International Conference on*, 2014, pp. 139-143.

[72] D. Gupta and R. Jahan. (2014). *Inter-sdn controller communication: Using border gateway protocol* [White Paper]. Available: https://www.researchgate.net/publication/324452059

[73] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, *et al.*, "SDX: a software defined internet exchange," *SIGCOMM Comput. Commun. Rev.,* vol. 44, pp. 551-562, 2014.

[74] P. Lin, J. Hart, U. Krishnaswamy, T. Murakami, M. Kobayashi, A. Al-Shabibi, *et al.*, "Seamless interworking of SDN and IP," in *ACM SIGCOMM computer communication review*, 2013, pp. 475-476.

[75] R. Jahan, S. Shaik, K. Kotaru, and D. C. Kuppili. (2014, December). *ODL-SDNi* Available: https://wiki.opendaylight.org/view/ODL-SDNi_App:Main

[76] J. Hart and L. Prete. *SDN-IP - ONOS - Wiki*. Available: https://wiki.onosproject.org/display/ONOS/SDN-IP

[77] Y. Rekhter, S. Hares, and T. Li. (2006). *A Border Gateway Protocol 4 (BGP-4)* [https://doi.org/10.17487/RFC4271]. Available: https://rfc-editor.org/rfc/rfc4271.txt

[78] D. Medhi and K. Ramasamy, *Network Routing : Algorithms, Protocols, and Architectures*. Saint Louis, UNITED STATES: Elsevier Science & Technology, 2017.

[79] T. Li, R. Chandra, and P. S. Traina. (1996). *BGP Communities Attribute* [https://doi.org/10.17487/RFC1997]. Available: https://rfc-editor.org/rfc/rfc1997.txt

[80] D. Tappan, S. S. Ramachandra, and Y. Rekhter. (2006). *BGP Extended Communities Attribute* [https://doi.org/10.17487/RFC4360]. Available: https://rfc-editor.org/rfc/rfc4360.txt

[81] F. X. A. Wibowo and M. A. Gregory, "Software Defined Networking properties in multi-domain networks," in *2016 26th International Telecommunication Networks and Applications Conference (ITNAC)*, 2016, pp. 95-100.

[82] M. Team. *Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet*. Available: http://mininet.org

[83] B. Gurudoss, P. Jakma, T. Teräs, and G. Troxel. *Quagga Software Routing Suite*. Available: https://www.quagga.net/

[84] *GitHub - routeflow_RouteFlow at vandervecken*. Available: https://github.com/routeflow/RouteFlow/tree/vandervecken

[85] R. Bennesby, E. Mota, P. Fonseca, and A. Passito, "Innovating on interdomain routing with an inter-SDN component," in *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, 2014, pp. 131-138.

[86] E. Rosen, G. Nalawade, M. Djernaes, and N. Sheth. (2018). *Border Gateway Protocol (BGP) Parameters*. Available: https://www.iana.org/assignments/bgp-parameters/bgp-parameters.xhtml#bgp-parameters-2

[87] D. Meyer. (2006). *BGP Communities for Data Collection* [https://doi.org/10.17487/RFC4384]. Available: https://rfc-editor.org/rfc/rfc4384.txt

[88] T. M. Knoll. (2018). *BGP Extended Community for QoS Marking*. Available: https://datatracker.ietf.org/doc/html/draft-knoll-idr-qos-attribute-22

[89] P. Mohapatra and R. Fernando. (2018). *BGP Link Bandwidth Extended Community*. Available: https://datatracker.ietf.org/doc/html/draft-ietf-idr-link-bandwidth-07

[90] Introducing ONOS -a SDN network operating system for Service Providers. [White Paper]. Available: http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf

[91] B. Lantz. *Introduction to the ONOS APIs - ONOS - Wiki*. Available: https://wiki.onosproject.org/display/ONOS/Introduction+to+the+ONOS+APIs

[92] *Open Network Operating System*. Available: https://onosproject.org

[93] L. Peterson. (2015, CORD: Central Office Re-Architected as a Datacenter. Available: https://sdn.ieee.org/newsletter/november-2015/cord-central-office-re-architected-as-a-datacenter

[94] L. Prete and A. Campanella. (2018). *ODTN - ODTN - Wiki*. Available:

https://wiki.onosproject.org/display/ODTN/ODTN

[95] A. Koshibe and J. Hart. *SDN-IP Architecture - ONOS - Wiki*. Available: https://wiki.onosproject.org/display/ONOS/SDN-IP+Architecture

[96] *Intent Framework - ONOS - Wiki*. Available: https://wiki.onosproject.org/display/ONOS/Intent+Framework

[97] A. Koshibe and J. Hart. *SDN-IP Developer Guide - ONOS - Wiki*. Available: https://wiki.onosproject.org/display/ONOS/SDN-IP+Developer+Guide

[98] *Onos/BgpConstants.java at master* Available: https://github.com/opennetworkinglab/onos/blob/master/apps/routing/common/src/main/java/org/onosproject/routing/bgp/BgpConstants.java

[99] T. GB921, "Enhanced Telecom Operations Map (eTOM)-The business process framework," in *Version 6.1. TeleManagement Forum, Morristown, NJ*, 2005.

[100] Y. Demchenko, S. Filiposka, R. Tuminauskas, A. Mishev, K. Baumann, D. Regvart*, et al.*, "Enabling Automated Network Services Provisioning for Cloud Based Applications Using Zero Touch Provisioning," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, 2015, pp. 458-464.

[101] K. Dilbeck. Zero-time Orchestration, Operations and Management (ZOOM). Available: https://www.tmforum.org/wp-content/uploads/2014/10/ZoomDownload.pdf

[102] F. X. A. Wibowo and M. A. Gregory, "Bandwidth Update Impact in Multi-Domain Software Defined Networking," *International Journal of Information, Communication Technology and Applications,* vol. 3, pp. 1-14, 2017.

Appendix 1 – BgpConstants Module

BgpConstants.java

```
/*
 * Copyright 2017-present Open Networking Foundation
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.onosproject.routing.bgp;

/**
 * BGP related constants.
 */
public final class BgpConstants {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private BgpConstants() {
    }

    /** BGP port number (RFC 4271). */
    public static final int BGP_PORT = 179;

    /** BGP version. */
    public static final int BGP_VERSION = 4;

    /** BGP OPEN message type. */
    public static final int BGP_TYPE_OPEN = 1;

    /** BGP UPDATE message type. */
    public static final int BGP_TYPE_UPDATE = 2;

    /** BGP NOTIFICATION message type. */
    public static final int BGP_TYPE_NOTIFICATION = 3;

    /** BGP KEEPALIVE message type. */
    public static final int BGP_TYPE_KEEPALIVE = 4;

    /** BGP Header Marker field length. */
    public static final int BGP_HEADER_MARKER_LENGTH = 16;

    /** BGP Header length. */
```

```
public static final int BGP_HEADER_LENGTH = 19;

/** BGP message maximum length. */
public static final int BGP_MESSAGE_MAX_LENGTH = 4096;

/** BGP OPEN message minimum length (BGP Header included). */
public static final int BGP_OPEN_MIN_LENGTH = 29;

/** BGP UPDATE message minimum length (BGP Header included). */
public static final int BGP_UPDATE_MIN_LENGTH = 23;

/** BGP NOTIFICATION message minimum length (BGP Header included). */
public static final int BGP_NOTIFICATION_MIN_LENGTH = 21;

/** BGP KEEPALIVE message expected length (BGP Header included). */
public static final int BGP_KEEPALIVE_EXPECTED_LENGTH = 19;

/** BGP KEEPALIVE messages transmitted per Hold interval. */
public static final int BGP_KEEPALIVE_PER_HOLD_INTERVAL = 3;

/** BGP KEEPALIVE messages minimum Holdtime (in seconds). */
public static final int BGP_KEEPALIVE_MIN_HOLDTIME = 3;

/** BGP KEEPALIVE messages minimum transmission interval (in seconds). */
public static final int BGP_KEEPALIVE_MIN_INTERVAL = 1;

/** BGP AS 0 (zero) value. See draft-ietf-idr-as0-06.txt Internet Draft. */
public static final long BGP_AS_0 = 0;

/**
 * BGP OPEN related constants.
 */
public static final class Open {
   /**
    * Default constructor.
    * <p>
    * The constructor is private to prevent creating an instance of
    * this utility class.
    */
   private Open() {
   }

   /**
    * BGP OPEN: Optional Parameters related constants.
    */
   public static final class OptionalParameters {
   }

   /**
    * BGP OPEN: Capabilities related constants (RFC 5492).
    */
   public static final class Capabilities {
      /** BGP OPEN Optional Parameter Type: Capabilities. */
      public static final int TYPE = 2;

      /** BGP OPEN Optional Parameter minimum length. */
      public static final int MIN_LENGTH = 2;

      /**
       * BGP OPEN: Multiprotocol Extensions Capabilities (RFC 4760).
```

```
         */
        public static final class MultiprotocolExtensions {
            /** BGP OPEN Multiprotocol Extensions code. */
            public static final int CODE = 1;

            /** BGP OPEN Multiprotocol Extensions length. */
            public static final int LENGTH = 4;

            /** BGP OPEN Multiprotocol Extensions AFI: IPv4. */
            public static final int AFI_IPV4 = 1;

            /** BGP OPEN Multiprotocol Extensions AFI: IPv6. */
            public static final int AFI_IPV6 = 2;

            /** BGP OPEN Multiprotocol Extensions SAFI: unicast. */
            public static final int SAFI_UNICAST = 1;

            /** BGP OPEN Multiprotocol Extensions SAFI: multicast. */
            public static final int SAFI_MULTICAST = 2;
        }

        /**
         * BGP OPEN: Support for 4-octet AS Number Capability (RFC 6793).
         */
        public static final class As4Octet {
            /** BGP OPEN Support for 4-octet AS Number Capability code. */
            public static final int CODE = 65;

            /** BGP OPEN 4-octet AS Number Capability length. */
            public static final int LENGTH = 4;
        }
    }
}

/**
 * BGP UPDATE related constants.
 */
public static final class Update {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private Update() {
    }

    /** BGP AS length. */
    public static final int AS_LENGTH = 2;

    /** BGP 4 Octet AS length (RFC 6793). */
    public static final int AS_4OCTET_LENGTH = 4;

    /**
     * BGP UPDATE: ORIGIN related constants.
     */
    public static final class Origin {
        /**
         * Default constructor.
         * <p>
```

```
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private Origin() {
    }

    /** BGP UPDATE Attributes Type Code ORIGIN. */
    public static final int TYPE = 1;

    /** BGP UPDATE Attributes Type Code ORIGIN length. */
    public static final int LENGTH = 1;

    /** BGP UPDATE ORIGIN: IGP. */
    public static final int IGP = 0;

    /** BGP UPDATE ORIGIN: EGP. */
    public static final int EGP = 1;

    /** BGP UPDATE ORIGIN: INCOMPLETE. */
    public static final int INCOMPLETE = 2;

    /**
     * Gets the BGP UPDATE origin type as a string.
     *
     * @param type the BGP UPDATE origin type
     * @return the BGP UPDATE origin type as a string
     */
    public static String typeToString(int type) {
        String typeString = "UNKNOWN";

        switch (type) {
        case IGP:
            typeString = "IGP";
            break;
        case EGP:
            typeString = "EGP";
            break;
        case INCOMPLETE:
            typeString = "INCOMPLETE";
            break;
        default:
            break;
        }
        return typeString;
    }
}

/**
 * BGP UPDATE: AS_PATH related constants.
 */
public static final class AsPath {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private AsPath() {
    }
```

```
    /** BGP UPDATE Attributes Type Code AS_PATH. */
    public static final int TYPE = 2;

    /** BGP UPDATE AS_PATH Type: AS_SET. */
    public static final int AS_SET = 1;

    /** BGP UPDATE AS_PATH Type: AS_SEQUENCE. */
    public static final int AS_SEQUENCE = 2;

    /** BGP UPDATE AS_PATH Type: AS_CONFED_SEQUENCE. */
    public static final int AS_CONFED_SEQUENCE = 3;

    /** BGP UPDATE AS_PATH Type: AS_CONFED_SET. */
    public static final int AS_CONFED_SET = 4;

    /**
     * Gets the BGP AS_PATH type as a string.
     *
     * @param type the BGP AS_PATH type
     * @return the BGP AS_PATH type as a string
     */
    public static String typeToString(int type) {
        String typeString = "UNKNOWN";

        switch (type) {
        case AS_SET:
            typeString = "AS_SET";
            break;
        case AS_SEQUENCE:
            typeString = "AS_SEQUENCE";
            break;
        case AS_CONFED_SEQUENCE:
            typeString = "AS_CONFED_SEQUENCE";
            break;
        case AS_CONFED_SET:
            typeString = "AS_CONFED_SET";
            break;
        default:
            break;
        }
        return typeString;
    }
}

/**
 * BGP UPDATE: NEXT_HOP related constants.
 */
public static final class NextHop {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private NextHop() {
    }

    /** BGP UPDATE Attributes Type Code NEXT_HOP. */
    public static final int TYPE = 3;
```

```
    /** BGP UPDATE Attributes Type Code NEXT_HOP length. */
    public static final int LENGTH = 4;
}

/**
 * BGP UPDATE: MULTI_EXIT_DISC related constants.
 */
public static final class MultiExitDisc {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private MultiExitDisc() {
    }

    /** BGP UPDATE Attributes Type Code MULTI_EXIT_DISC. */
    public static final int TYPE = 4;

    /** BGP UPDATE Attributes Type Code MULTI_EXIT_DISC length. */
    public static final int LENGTH = 4;

    /** BGP UPDATE Attributes lowest MULTI_EXIT_DISC value. */
    public static final int LOWEST_MULTI_EXIT_DISC = 0;
}

/**
 * BGP UPDATE: LOCAL_PREF related constants.
 */
public static final class LocalPref {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private LocalPref() {
    }

    /** BGP UPDATE Attributes Type Code LOCAL_PREF. */
    public static final int TYPE = 5;

    /** BGP UPDATE Attributes Type Code LOCAL_PREF length. */
    public static final int LENGTH = 4;
}

/**
 * BGP UPDATE: ATOMIC_AGGREGATE related constants.
 */
public static final class AtomicAggregate {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private AtomicAggregate() {
    }
```

```
    /** BGP UPDATE Attributes Type Code ATOMIC_AGGREGATE. */
    public static final int TYPE = 6;

    /** BGP UPDATE Attributes Type Code ATOMIC_AGGREGATE length. */
    public static final int LENGTH = 0;
}

/**
 * BGP UPDATE: AGGREGATOR related constants.
 */
public static final class Aggregator {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private Aggregator() {
    }

    /** BGP UPDATE Attributes Type Code AGGREGATOR. */
    public static final int TYPE = 7;

    /** BGP UPDATE Attributes Type Code AGGREGATOR length: 2 octet AS. */
    public static final int AS2_LENGTH = 6;

    /** BGP UPDATE Attributes Type Code AGGREGATOR length: 4 octet AS. */
    public static final int AS4_LENGTH = 8;
}

/**
 * BGP UPDATE: COMMUNITY related constants.
 */
public static final class Community {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private Community() {
    }

    /** BGP UPDATE Attributes Type Code COMMUNITY. */
    public static final int TYPE = 8;

    /** BGP UPDATE Attributes Type Code COMMUNITY minimum length: 4 octet. */
    public static final int MIN_COMM_LENGTH = 4;
}

/**
 * BGP UPDATE: EXTENDED COMMUNITY related constants.
 */
public static final class ExtendedCommunity {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
```

```
        private ExtendedCommunity() {
        }

        /** BGP UPDATE Attributes Type Code ExtendedCOMMUNITY. */
        public static final int TYPE = 16;

        /** BGP UPDATE Attributes Type Code ExtendedCOMMUNITY minimum length: 8 octet. */
        public static final int MIN_COMM_LENGTH = 8;
    }


    /**
     * BGP UPDATE: MP_REACH_NLRI related constants.
     */
    public static final class MpReachNlri {
        /**
         * Default constructor.
         * <p>
         * The constructor is private to prevent creating an instance of
         * this utility class.
         */
        private MpReachNlri() {
        }

        /** BGP UPDATE Attributes Type Code MP_REACH_NLRI. */
        public static final int TYPE = 14;

        /** BGP UPDATE Attributes Type Code MP_REACH_NLRI min length. */
        public static final int MIN_LENGTH = 5;
    }

    /**
     * BGP UPDATE: MP_UNREACH_NLRI related constants.
     */
    public static final class MpUnreachNlri {
        /**
         * Default constructor.
         * <p>
         * The constructor is private to prevent creating an instance of
         * this utility class.
         */
        private MpUnreachNlri() {
        }

        /** BGP UPDATE Attributes Type Code MP_UNREACH_NLRI. */
        public static final int TYPE = 15;

        /** BGP UPDATE Attributes Type Code MP_UNREACH_NLRI min length. */
        public static final int MIN_LENGTH = 3;
    }
}

/**
 * BGP NOTIFICATION related constants.
 */
public static final class Notifications {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
```

```
 * this utility class.
 */
private Notifications() {
}

/**
 * BGP NOTIFICATION: Message Header Error constants.
 */
public static final class MessageHeaderError {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private MessageHeaderError() {
    }

    /** Message Header Error code. */
    public static final int ERROR_CODE = 1;

    /** Message Header Error subcode: Connection Not Synchronized. */
    public static final int CONNECTION_NOT_SYNCHRONIZED = 1;

    /** Message Header Error subcode: Bad Message Length. */
    public static final int BAD_MESSAGE_LENGTH = 2;

    /** Message Header Error subcode: Bad Message Type. */
    public static final int BAD_MESSAGE_TYPE = 3;
}

/**
 * BGP NOTIFICATION: OPEN Message Error constants.
 */
public static final class OpenMessageError {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private OpenMessageError() {
    }

    /** OPEN Message Error code. */
    public static final int ERROR_CODE = 2;

    /** OPEN Message Error subcode: Unsupported Version Number. */
    public static final int UNSUPPORTED_VERSION_NUMBER = 1;

    /** OPEN Message Error subcode: Bad PEER AS. */
    public static final int BAD_PEER_AS = 2;

    /** OPEN Message Error subcode: Unacceptable Hold Time. */
    public static final int UNACCEPTABLE_HOLD_TIME = 6;
}

/**
 * BGP NOTIFICATION: UPDATE Message Error constants.
 */
```

```
public static final class UpdateMessageError {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private UpdateMessageError() {
    }

    /** UPDATE Message Error code. */
    public static final int ERROR_CODE = 3;

    /** UPDATE Message Error subcode: Malformed Attribute List. */
    public static final int MALFORMED_ATTRIBUTE_LIST = 1;

    /** UPDATE Message Error subcode: Unrecognized Well-known Attribute. */
    public static final int UNRECOGNIZED_WELL_KNOWN_ATTRIBUTE = 2;

    /** UPDATE Message Error subcode: Missing Well-known Attribute. */
    public static final int MISSING_WELL_KNOWN_ATTRIBUTE = 3;

   /** UPDATE Message Error subcode: Attribute Flags Error. */
    public static final int ATTRIBUTE_FLAGS_ERROR = 4;

    /** UPDATE Message Error subcode: Attribute Length Error. */
    public static final int ATTRIBUTE_LENGTH_ERROR = 5;

    /** UPDATE Message Error subcode: Invalid ORIGIN Attribute. */
    public static final int INVALID_ORIGIN_ATTRIBUTE = 6;

    /** UPDATE Message Error subcode: Invalid NEXT_HOP Attribute. */
    public static final int INVALID_NEXT_HOP_ATTRIBUTE = 8;

    /** UPDATE Message Error subcode: Optional Attribute Error. Unused. */
    public static final int OPTIONAL_ATTRIBUTE_ERROR = 9;

    /** UPDATE Message Error subcode: Invalid Network Field. */
    public static final int INVALID_NETWORK_FIELD = 10;

    /** UPDATE Message Error subcode: Malformed AS_PATH. */
    public static final int MALFORMED_AS_PATH = 11;
}

/**
 * BGP NOTIFICATION: Hold Timer Expired constants.
 */
public static final class HoldTimerExpired {
    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private HoldTimerExpired() {
    }

    /** Hold Timer Expired code. */
    public static final int ERROR_CODE = 4;
}
```

```
        /** BGP NOTIFICATION message Error subcode: Unspecific. */
        public static final int ERROR_SUBCODE_UNSPECIFIC = 0;
    }
}
```

Appendix 2 – BgpUpdate Module

BgpUpdate.java

```
/*
 * Copyright 2017-present Open Networking Foundation
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.onosproject.routing.bgp;

import org.apache.commons.lang3.tuple.Pair;
import org.apache.commons.lang3.tuple.Triple;
import org.javatuples.Quartet;
import org.jboss.netty.buffer.ChannelBuffer;
import org.jboss.netty.buffer.ChannelBuffers;
import org.jboss.netty.channel.ChannelHandlerContext;
import org.onlab.packet.Ip4Address;
import org.onlab.packet.Ip4Prefix;
import org.onlab.packet.Ip6Address;
import org.onlab.packet.Ip6Prefix;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

/**
 * A class for handling BGP UPDATE messages.
 */
final class BgpUpdate {
    private static final Logger log = LoggerFactory.getLogger(BgpUpdate.class);

    /**
     * Default constructor.
     * <p>
     * The constructor is private to prevent creating an instance of
     * this utility class.
     */
    private BgpUpdate() {
    }

    /**
     * Processes BGP UPDATE message.
     *
```

```
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param message the message to process
 */
static void processBgpUpdate(BgpSession bgpSession,
                             ChannelHandlerContext ctx,
                             ChannelBuffer message) {
    DecodedBgpRoutes decodedBgpRoutes = new DecodedBgpRoutes();

    int minLength =
        BgpConstants.BGP_UPDATE_MIN_LENGTH - BgpConstants.BGP_HEADER_LENGTH;
    if (message.readableBytes() < minLength) {
        log.debug("BGP RX UPDATE Error from {}: " +
                  "Message length {} too short. Must be at least {}",
                  bgpSession.remoteInfo().address(),
                  message.readableBytes(), minLength);
        //
        // ERROR: Bad Message Length
        //
        // Send NOTIFICATION and close the connection
        ChannelBuffer txMessage =
            BgpNotification.prepareBgpNotificationBadMessageLength(
            message.readableBytes() + BgpConstants.BGP_HEADER_LENGTH);
        ctx.getChannel().write(txMessage);
        bgpSession.closeSession(ctx);
        return;
    }

    log.debug("BGP RX UPDATE message from {}",
              bgpSession.remoteInfo().address());

    //
    // Parse the UPDATE message
    //

    //
    // Parse the Withdrawn Routes
    //
    int withdrawnRoutesLength = message.readUnsignedShort();
    if (withdrawnRoutesLength > message.readableBytes()) {
        // ERROR: Malformed Attribute List
        actionsBgpUpdateMalformedAttributeList(bgpSession, ctx);
        return;
    }
    Collection<Ip4Prefix> withdrawnPrefixes = null;
    try {
        withdrawnPrefixes = parsePackedIp4Prefixes(withdrawnRoutesLength,
                                                   message);
    } catch (BgpMessage.BgpParseException e) {
        // ERROR: Invalid Network Field
        log.debug("Exception parsing Withdrawn Prefixes from BGP peer {}: ",
                  bgpSession.remoteInfo().bgpId(), e);
        actionsBgpUpdateInvalidNetworkField(bgpSession, ctx);
        return;
    }
    for (Ip4Prefix prefix : withdrawnPrefixes) {
        log.debug("BGP RX UPDATE message WITHDRAWN from {}: {}",
                  bgpSession.remoteInfo().address(), prefix);
        BgpRouteEntry bgpRouteEntry = bgpSession.findBgpRoute(prefix);
        if (bgpRouteEntry != null) {
```

```
                decodedBgpRoutes.deletedUnicastRoutes4.put(prefix,
                                                bgpRouteEntry);
            }
        }

        //
        // Parse the Path Attributes
        //
        try {
            parsePathAttributes(bgpSession, ctx, message, decodedBgpRoutes);
        } catch (BgpMessage.BgpParseException e) {
            log.debug("Exception parsing Path Attributes from BGP peer {}: ",
                    bgpSession.remoteInfo().bgpId(), e);
            // NOTE: The session was already closed, so nothing else to do
            return;
        }

        //
        // Update the BGP RIB-IN
        //
        for (Ip4Prefix ip4Prefix :
                decodedBgpRoutes.deletedUnicastRoutes4.keySet()) {
            bgpSession.removeBgpRoute(ip4Prefix);
        }
        //
        for (BgpRouteEntry bgpRouteEntry :
                decodedBgpRoutes.addedUnicastRoutes4.values()) {
            bgpSession.addBgpRoute(bgpRouteEntry);
        }
        //
        for (Ip6Prefix ip6Prefix :
                decodedBgpRoutes.deletedUnicastRoutes6.keySet()) {
            bgpSession.removeBgpRoute(ip6Prefix);
        }
        //
        for (BgpRouteEntry bgpRouteEntry :
                decodedBgpRoutes.addedUnicastRoutes6.values()) {
            bgpSession.addBgpRoute(bgpRouteEntry);
        }

        //
        // Push the updates to the BGP Merged RIB
        //
        BgpRouteSelector bgpRouteSelector =
            bgpSession.getBgpSessionManager().getBgpRouteSelector();
        bgpRouteSelector.routeUpdates(
                            decodedBgpRoutes.addedUnicastRoutes4.values(),
                            decodedBgpRoutes.deletedUnicastRoutes4.values());
        bgpRouteSelector.routeUpdates(
                            decodedBgpRoutes.addedUnicastRoutes6.values(),
                            decodedBgpRoutes.deletedUnicastRoutes6.values());

        // Start the Session Timeout timer
        bgpSession.restartSessionTimeoutTimer(ctx);
    }

    /**
     * Parse BGP Path Attributes from the BGP UPDATE message.
     *
     * @param bgpSession the BGP Session to use
```

```
 * @param ctx the Channel Handler Context
 * @param message the message to parse
 * @param decodedBgpRoutes the container to store the decoded BGP Route
 * Entries. It might already contain some route entries such as withdrawn
 * IPv4 prefixes
 * @throws BgpMessage.BgpParseException
 */
// CHECKSTYLE IGNORE MethodLength FOR NEXT 300 LINES
private static void parsePathAttributes(
                            BgpSession bgpSession,
                            ChannelHandlerContext ctx,
                            ChannelBuffer message,
                            DecodedBgpRoutes decodedBgpRoutes)
    throws BgpMessage.BgpParseException {

    //
    // Parsed values
    //
    Short origin = -1;                    // Mandatory
    BgpRouteEntry.AsPath asPath = null;     // Mandatory
    // Legacy NLRI (RFC 4271). Mandatory NEXT_HOP if legacy NLRI is used
    MpNlri legacyNlri = new MpNlri(
          BgpConstants.Open.Capabilities.MultiprotocolExtensions.AFI_IPV4,
                        BgpConstants.Open.Capabilities.MultiprotocolExtensions.SAFI_UNICAST);
    long multiExitDisc =                  // Optional
       BgpConstants.Update.MultiExitDisc.LOWEST_MULTI_EXIT_DISC;
    Long localPref = null;                // Mandatory
    Long aggregatorAsNumber = null;         // Optional: unused
    Ip4Address aggregatorIpAddress = null;  // Optional: unused
    Long communityAsNumber = null;          // Optional
    Long communityLocalIdentifier = null;   // Optional
    Pair<Long, Long> community = null;      // Optional
    Long extCommunityValueOne = null;       // Optional
    Long extCommunityValueTwo = null;       // Optional
    Long extCommunityValueThree = null;       // Optional
    Triple<Long, Long, Long> extCommunity = null;       // Optional
    BgpRouteEntry.Communities communities = null;              // Optional
    BgpRouteEntry.extCommunities extCommunities = null;         // Optional
    Collection<MpNlri> mpNlriReachList = new ArrayList<>();     // Optional
    Collection<MpNlri> mpNlriUnreachList = new ArrayList<>();   // Optional

    //
    // Get and verify the Path Attributes Length
    //
    int pathAttributeLength = message.readUnsignedShort();
    if (pathAttributeLength > message.readableBytes()) {
       // ERROR: Malformed Attribute List
       actionsBgpUpdateMalformedAttributeList(bgpSession, ctx);
       String errorMsg = "Malformed Attribute List";
       throw new BgpMessage.BgpParseException(errorMsg);
    }
    if (pathAttributeLength == 0) {
       return;
    }

    //
    // Parse the Path Attributes
    //
    int pathAttributeEnd = message.readerIndex() + pathAttributeLength;
    while (message.readerIndex() < pathAttributeEnd) {
```

```
int attrFlags = message.readUnsignedByte();
if (message.readerIndex() >= pathAttributeEnd) {
   // ERROR: Malformed Attribute List
   actionsBgpUpdateMalformedAttributeList(bgpSession, ctx);
   String errorMsg = "Malformed Attribute List";
   throw new BgpMessage.BgpParseException(errorMsg);
}
int attrTypeCode = message.readUnsignedByte();

// The Attribute Flags
boolean optionalBit =       ((0x80 & attrFlags) != 0);
boolean transitiveBit =     ((0x40 & attrFlags) != 0);
boolean partialBit =        ((0x20 & attrFlags) != 0);
boolean extendedLengthBit = ((0x10 & attrFlags) != 0);

// The Attribute Length
int attrLen = 0;
int attrLenOctets = 1;
if (extendedLengthBit) {
   attrLenOctets = 2;
}
if (message.readerIndex() + attrLenOctets > pathAttributeEnd) {
   // ERROR: Malformed Attribute List
   actionsBgpUpdateMalformedAttributeList(bgpSession, ctx);
   String errorMsg = "Malformed Attribute List";
   throw new BgpMessage.BgpParseException(errorMsg);
}
if (extendedLengthBit) {
   attrLen = message.readUnsignedShort();
} else {
   attrLen = message.readUnsignedByte();
}
if (message.readerIndex() + attrLen > pathAttributeEnd) {
   // ERROR: Malformed Attribute List
   actionsBgpUpdateMalformedAttributeList(bgpSession, ctx);
   String errorMsg = "Malformed Attribute List";
   throw new BgpMessage.BgpParseException(errorMsg);
}

// Verify the Attribute Flags
verifyBgpUpdateAttributeFlags(bgpSession, ctx, attrTypeCode,
                        attrLen, attrFlags, message);

//
// Extract the Attribute Value based on the Attribute Type Code
//
switch (attrTypeCode) {

case BgpConstants.Update.Origin.TYPE:
   // Attribute Type Code ORIGIN
   origin = parseAttributeTypeOrigin(bgpSession, ctx,
                              attrTypeCode, attrLen,
                              attrFlags, message);
   break;

case BgpConstants.Update.AsPath.TYPE:
   // Attribute Type Code AS_PATH
   asPath = parseAttributeTypeAsPath(bgpSession, ctx,
                              attrTypeCode, attrLen,
                              attrFlags, message);
```

```
    break;

case BgpConstants.Update.NextHop.TYPE:
    // Attribute Type Code NEXT_HOP
    legacyNlri.nextHop4 =
        parseAttributeTypeNextHop(bgpSession, ctx,
                                  attrTypeCode, attrLen,
                                  attrFlags, message);
    break;

case BgpConstants.Update.MultiExitDisc.TYPE:
    // Attribute Type Code MULTI_EXIT_DISC
    multiExitDisc =
        parseAttributeTypeMultiExitDisc(bgpSession, ctx,
                                        attrTypeCode, attrLen,
                                        attrFlags, message);
    break;

case BgpConstants.Update.LocalPref.TYPE:
    // Attribute Type Code LOCAL_PREF
    localPref =
        parseAttributeTypeLocalPref(bgpSession, ctx,
                                    attrTypeCode, attrLen,
                                    attrFlags, message);
    break;

case BgpConstants.Update.AtomicAggregate.TYPE:
    // Attribute Type Code ATOMIC_AGGREGATE
    parseAttributeTypeAtomicAggregate(bgpSession, ctx,
                                      attrTypeCode, attrLen,
                                      attrFlags, message);
    // Nothing to do: this attribute is primarily informational
    break;

case BgpConstants.Update.Aggregator.TYPE:
    // Attribute Type Code AGGREGATOR
    Pair<Long, Ip4Address> aggregator =
        parseAttributeTypeAggregator(bgpSession, ctx,
                                     attrTypeCode, attrLen,
                                     attrFlags, message);
    aggregatorAsNumber = aggregator.getLeft();
    aggregatorIpAddress = aggregator.getRight();
    break;

case BgpConstants.Update.Community.TYPE:
    // Attribute Type Code COMMUNITY
    communities =
          parseAttributeTypeCommunity(bgpSession, ctx,
                attrTypeCode, attrLen,
                attrFlags, message);
    break;

case BgpConstants.Update.ExtendedCommunity.TYPE:
      // Attribute Type Code Extended COMMUNITY
      extCommunities =
            parseAttributeTypeExtCommunity(bgpSession, ctx,
                  attrTypeCode, attrLen,
                  attrFlags, message);
      break;
```

```
    case BgpConstants.Update.MpReachNlri.TYPE:
        // Attribute Type Code MP_REACH_NLRI
        MpNlri mpNlriReach =
            parseAttributeTypeMpReachNlri(bgpSession, ctx,
                                          attrTypeCode,
                                          attrLen,
                                          attrFlags, message);
        if (mpNlriReach != null) {
            mpNlriReachList.add(mpNlriReach);
        }
        break;

    case BgpConstants.Update.MpUnreachNlri.TYPE:
        // Attribute Type Code MP_UNREACH_NLRI
        MpNlri mpNlriUnreach =
            parseAttributeTypeMpUnreachNlri(bgpSession, ctx,
                                            attrTypeCode, attrLen,
                                            attrFlags, message);
        if (mpNlriUnreach != null) {
            mpNlriUnreachList.add(mpNlriUnreach);
        }
        break;

    default:
        // NOTE: Parse any new Attribute Types if needed
        if (!optionalBit) {
            // ERROR: Unrecognized Well-known Attribute
            actionsBgpUpdateUnrecognizedWellKnownAttribute(
                bgpSession, ctx, attrTypeCode, attrLen, attrFlags,
                message);
            String errorMsg = "Unrecognized Well-known Attribute: " +
                attrTypeCode;
            throw new BgpMessage.BgpParseException(errorMsg);
        }

        // Skip the data from the unrecognized attribute
        log.debug("BGP RX UPDATE message from {}: " +
                  "Unrecognized Attribute Type {}",
                  bgpSession.remoteInfo().address(), attrTypeCode);
        message.skipBytes(attrLen);
        break;
    }
}

//
// Parse the NLRI (Network Layer Reachability Information)
//
int nlriLength = message.readableBytes();
try {
    Collection<Ip4Prefix> addedPrefixes4 =
        parsePackedIp4Prefixes(nlriLength, message);
    // Store it inside the legacy NLRI wrapper
    legacyNlri.nlri4 = addedPrefixes4;
} catch (BgpMessage.BgpParseException e) {
    // ERROR: Invalid Network Field
    log.debug("Exception parsing NLRI from BGP peer {}: ",
              bgpSession.remoteInfo().bgpId(), e);
    actionsBgpUpdateInvalidNetworkField(bgpSession, ctx);
    // Rethrow the exception
    throw e;
```

```
}

// Verify the Well-known Attributes
verifyBgpUpdateWellKnownAttributes(bgpSession, ctx, origin, asPath,
                                   localPref, legacyNlri,
                                   mpNlriReachList);

//
// Generate the deleted routes
//
for (MpNlri mpNlri : mpNlriUnreachList) {
   BgpRouteEntry bgpRouteEntry;

   // The deleted IPv4 routes
   for (Ip4Prefix prefix : mpNlri.nlri4) {
      bgpRouteEntry = bgpSession.findBgpRoute(prefix);
      if (bgpRouteEntry != null) {
         decodedBgpRoutes.deletedUnicastRoutes4.put(prefix,
                                            bgpRouteEntry);
      }
   }

   // The deleted IPv6 routes
   for (Ip6Prefix prefix : mpNlri.nlri6) {
      bgpRouteEntry = bgpSession.findBgpRoute(prefix);
      if (bgpRouteEntry != null) {
         decodedBgpRoutes.deletedUnicastRoutes6.put(prefix,
                                            bgpRouteEntry);
      }
   }
}

//
// Generate the added routes
//
mpNlriReachList.add(legacyNlri);
for (MpNlri mpNlri : mpNlriReachList) {
   BgpRouteEntry bgpRouteEntry;

   // The added IPv4 routes
   for (Ip4Prefix prefix : mpNlri.nlri4) {
      bgpRouteEntry =
         new BgpRouteEntry(bgpSession, prefix, mpNlri.nextHop4,
                           origin.byteValue(), asPath, localPref,
                           communities, extCommunities);
      bgpRouteEntry.setMultiExitDisc(multiExitDisc);
      if (bgpRouteEntry.hasAsPathLoop(bgpSession.localInfo().asNumber())) {
         log.debug("BGP RX UPDATE message IGNORED from {}: {} " +
                   "nextHop {}: contains AS Path loop",
                   bgpSession.remoteInfo().address(), prefix,
                   mpNlri.nextHop4);
         continue;
      } else {
         log.debug("BGP RX UPDATE message ADDED from {}: {} nextHop {}",
                   bgpSession.remoteInfo().address(), prefix,
                   mpNlri.nextHop4);
      }
      // Remove from the collection of deleted routes
      decodedBgpRoutes.deletedUnicastRoutes4.remove(prefix);
      decodedBgpRoutes.addedUnicastRoutes4.put(prefix,
```

```
                                    bgpRouteEntry);
        }

        // The added IPv6 routes
        for (Ip6Prefix prefix : mpNlri.nlri6) {
          bgpRouteEntry =
            new BgpRouteEntry(bgpSession, prefix, mpNlri.nextHop6,
                    origin.byteValue(), asPath, localPref,
                    communities, extCommunities);
          bgpRouteEntry.setMultiExitDisc(multiExitDisc);
          if (bgpRouteEntry.hasAsPathLoop(bgpSession.localInfo().asNumber())) {
            log.debug("BGP RX UPDATE message IGNORED from {}: {} " +
                "nextHop {}: contains AS Path loop",
                bgpSession.remoteInfo().address(), prefix,
                mpNlri.nextHop6);
            continue;
          } else {
            log.debug("BGP RX UPDATE message ADDED from {}: {} nextHop {}",
                bgpSession.remoteInfo().address(), prefix,
                mpNlri.nextHop6);
          }
          // Remove from the collection of deleted routes
          decodedBgpRoutes.deletedUnicastRoutes6.remove(prefix);
          decodedBgpRoutes.addedUnicastRoutes6.put(prefix,
                                    bgpRouteEntry);
        }
      }
    }

    /**
     * Verifies BGP UPDATE Well-known Attributes.
     *
     * @param bgpSession the BGP Session to use
     * @param ctx the Channel Handler Context
     * @param origin the ORIGIN well-known mandatory attribute
     * @param asPath the AS_PATH well-known mandatory attribute
     * @param localPref the LOCAL_PREF required attribute
     * @param legacyNlri the legacy NLRI. Encapsulates the NEXT_HOP well-known
     * mandatory attribute (mandatory if legacy NLRI is used).
     * @param mpNlriReachList the Multiprotocol NLRI attributes
     * @throws BgpMessage.BgpParseException
     */
    private static void verifyBgpUpdateWellKnownAttributes(
                        BgpSession bgpSession,
                        ChannelHandlerContext ctx,
                        Short origin,
                        BgpRouteEntry.AsPath asPath,
                        Long localPref,
                        MpNlri legacyNlri,
                        Collection<MpNlri> mpNlriReachList)
        throws BgpMessage.BgpParseException {
        boolean hasNlri = false;
        boolean hasLegacyNlri = false;

        //
        // Convenience flags that are used to check for missing attributes.
        //
        // NOTE: The hasLegacyNlri flag is always set to true if the
        // Multiprotocol Extensions are not enabled, even if the UPDATE
        // message doesn't contain the legacy NLRI (per RFC 4271).
```

```
    //
    if (!bgpSession.mpExtensions()) {
        hasNlri = true;
        hasLegacyNlri = true;
    } else {
        if (!legacyNlri.nlri4.isEmpty()) {
            hasNlri = true;
            hasLegacyNlri = true;
        }
        if (!mpNlriReachList.isEmpty()) {
            hasNlri = true;
        }
    }

    //
    // Check for Missing Well-known Attributes
    //
    if (hasNlri && ((origin == null) || (origin == -1))) {
        // Missing Attribute Type Code ORIGIN
        int type = BgpConstants.Update.Origin.TYPE;
        actionsBgpUpdateMissingWellKnownAttribute(bgpSession, ctx, type);
        String errorMsg = "Missing Well-known Attribute: ORIGIN";
        throw new BgpMessage.BgpParseException(errorMsg);
    }
    if (hasNlri && (asPath == null)) {
        // Missing Attribute Type Code AS_PATH
        int type = BgpConstants.Update.AsPath.TYPE;
        actionsBgpUpdateMissingWellKnownAttribute(bgpSession, ctx, type);
        String errorMsg = "Missing Well-known Attribute: AS_PATH";
        throw new BgpMessage.BgpParseException(errorMsg);
    }
    if (hasNlri && (localPref == null)) {
        // Missing Attribute Type Code LOCAL_PREF
        // NOTE: Required for iBGP
        int type = BgpConstants.Update.LocalPref.TYPE;
        actionsBgpUpdateMissingWellKnownAttribute(bgpSession, ctx, type);
        String errorMsg = "Missing Well-known Attribute: LOCAL_PREF";
        throw new BgpMessage.BgpParseException(errorMsg);
    }
    if (hasLegacyNlri && (legacyNlri.nextHop4 == null)) {
        // Missing Attribute Type Code NEXT_HOP
        int type = BgpConstants.Update.NextHop.TYPE;
        actionsBgpUpdateMissingWellKnownAttribute(bgpSession, ctx, type);
        String errorMsg = "Missing Well-known Attribute: NEXT_HOP";
        throw new BgpMessage.BgpParseException(errorMsg);
    }
}

/**
 * Verifies the BGP UPDATE Attribute Flags.
 *
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param attrTypeCode the attribute type code
 * @param attrLen the attribute length (in octets)
 * @param attrFlags the attribute flags
 * @param message the message to parse
 * @throws BgpMessage.BgpParseException
 */
private static void verifyBgpUpdateAttributeFlags(
```

```
                    BgpSession bgpSession,
                    ChannelHandlerContext ctx,
                    int attrTypeCode,
                    int attrLen,
                    int attrFlags,
                    ChannelBuffer message)
throws BgpMessage.BgpParseException {

//
// Assign the Attribute Type Name and the Well-known flag
//
String typeName = "UNKNOWN";
boolean isWellKnown = false;
switch (attrTypeCode) {
case BgpConstants.Update.Origin.TYPE:
   isWellKnown = true;
   typeName = "ORIGIN";
   break;
case BgpConstants.Update.AsPath.TYPE:
   isWellKnown = true;
   typeName = "AS_PATH";
   break;
case BgpConstants.Update.NextHop.TYPE:
   isWellKnown = true;
   typeName = "NEXT_HOP";
   break;
case BgpConstants.Update.MultiExitDisc.TYPE:
   isWellKnown = false;
   typeName = "MULTI_EXIT_DISC";
   break;
case BgpConstants.Update.LocalPref.TYPE:
   isWellKnown = true;
   typeName = "LOCAL_PREF";
   break;
case BgpConstants.Update.AtomicAggregate.TYPE:
   isWellKnown = true;
   typeName = "ATOMIC_AGGREGATE";
   break;
case BgpConstants.Update.Aggregator.TYPE:
   isWellKnown = false;
   typeName = "AGGREGATOR";
   break;
case BgpConstants.Update.Community.TYPE:
   isWellKnown = false;
   typeName = "COMMUNITY";
   break;
case BgpConstants.Update.MpReachNlri.TYPE:
   isWellKnown = false;
   typeName = "MP_REACH_NLRI";
   break;
case BgpConstants.Update.MpUnreachNlri.TYPE:
   isWellKnown = false;
   typeName = "MP_UNREACH_NLRI";
   break;
default:
   isWellKnown = false;
   typeName = "UNKNOWN(" + attrTypeCode + ")";
   break;
}
```

```
    //
    // Verify the Attribute Flags
    //
    boolean optionalBit =       ((0x80 & attrFlags) != 0);
    boolean transitiveBit =     ((0x40 & attrFlags) != 0);
    boolean partialBit =        ((0x20 & attrFlags) != 0);
    if ((isWellKnown && optionalBit) ||
        (isWellKnown && (!transitiveBit)) ||
        (isWellKnown && partialBit) ||
        (optionalBit && (!transitiveBit) && partialBit)) {
        //
        // ERROR: The Optional bit cannot be set for Well-known attributes
        // ERROR: The Transtive bit MUST be 1 for well-known attributes
        // ERROR: The Partial bit MUST be 0 for well-known attributes
        // ERROR: The Partial bit MUST be 0 for optional non-transitive
        //        attributes
        //
        actionsBgpUpdateAttributeFlagsError(
            bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
        String errorMsg = "Attribute Flags Error for " + typeName + ": " +
            attrFlags;
        throw new BgpMessage.BgpParseException(errorMsg);
    }
}

/**
 * Parses BGP UPDATE Attribute Type ORIGIN.
 *
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param attrTypeCode the attribute type code
 * @param attrLen the attribute length (in octets)
 * @param attrFlags the attribute flags
 * @param message the message to parse
 * @return the parsed ORIGIN value
 * @throws BgpMessage.BgpParseException
 */
private static short parseAttributeTypeOrigin(
                        BgpSession bgpSession,
                        ChannelHandlerContext ctx,
                        int attrTypeCode,
                        int attrLen,
                        int attrFlags,
                        ChannelBuffer message)
    throws BgpMessage.BgpParseException {

    // Check the Attribute Length
    if (attrLen != BgpConstants.Update.Origin.LENGTH) {
        // ERROR: Attribute Length Error
        actionsBgpUpdateAttributeLengthError(
            bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
        String errorMsg = "Attribute Length Error";
        throw new BgpMessage.BgpParseException(errorMsg);
    }

    message.markReaderIndex();
    short origin = message.readUnsignedByte();
    switch (origin) {
    case BgpConstants.Update.Origin.IGP:
        // FALLTHROUGH
```

```
    case BgpConstants.Update.Origin.EGP:
        // FALLTHROUGH
    case BgpConstants.Update.Origin.INCOMPLETE:
        break;
    default:
        // ERROR: Invalid ORIGIN Attribute
        message.resetReaderIndex();
        actionsBgpUpdateInvalidOriginAttribute(
            bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message,
            origin);
        String errorMsg = "Invalid ORIGIN Attribute: " + origin;
        throw new BgpMessage.BgpParseException(errorMsg);
    }

    return origin;
}

/**
 * Parses BGP UPDATE Attribute AS Path.
 *
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param attrTypeCode the attribute type code
 * @param attrLen the attribute length (in octets)
 * @param attrFlags the attribute flags
 * @param message the message to parse
 * @return the parsed AS Path
 * @throws BgpMessage.BgpParseException
 */
private static BgpRouteEntry.AsPath parseAttributeTypeAsPath(
                                    BgpSession bgpSession,
                                    ChannelHandlerContext ctx,
                                    int attrTypeCode,
                                    int attrLen,
                                    int attrFlags,
                                    ChannelBuffer message)
    throws BgpMessage.BgpParseException {
    ArrayList<BgpRouteEntry.PathSegment> pathSegments = new ArrayList<>();

    //
    // Parse the message
    //
    while (attrLen > 0) {
        if (attrLen < 2) {
            // ERROR: Malformed AS_PATH
            actionsBgpUpdateMalformedAsPath(bgpSession, ctx);
            String errorMsg = "Malformed AS Path";
            throw new BgpMessage.BgpParseException(errorMsg);
        }
        // Get the Path Segment Type and Length (in number of ASes)
        short pathSegmentType = message.readUnsignedByte();
        short pathSegmentLength = message.readUnsignedByte();
        attrLen -= 2;

        // Verify the Path Segment Type
        switch (pathSegmentType) {
        case BgpConstants.Update.AsPath.AS_SET:
            // FALLTHROUGH
        case BgpConstants.Update.AsPath.AS_SEQUENCE:
            // FALLTHROUGH
```

```
        case BgpConstants.Update.AsPath.AS_CONFED_SEQUENCE:
          // FALLTHROUGH
        case BgpConstants.Update.AsPath.AS_CONFED_SET:
          break;
        default:
          // ERROR: Invalid Path Segment Type
          //
          // NOTE: The BGP Spec (RFC 4271) doesn't contain Error Subcode
          // for "Invalid Path Segment Type", hence we return
          // the error as "Malformed AS_PATH".
          //
          actionsBgpUpdateMalformedAsPath(bgpSession, ctx);
          String errorMsg =
              "Invalid AS Path Segment Type: " + pathSegmentType;
          throw new BgpMessage.BgpParseException(errorMsg);
        }

        // 4-octet AS number handling.
        int asPathLen;
        if (bgpSession.isAs4OctetCapable()) {
          asPathLen = BgpConstants.Update.AS_4OCTET_LENGTH;
        } else {
          asPathLen = BgpConstants.Update.AS_LENGTH;
        }

        // Parse the AS numbers
        if (asPathLen * pathSegmentLength > attrLen) {
          // ERROR: Malformed AS_PATH
          actionsBgpUpdateMalformedAsPath(bgpSession, ctx);
          String errorMsg = "Malformed AS Path";
          throw new BgpMessage.BgpParseException(errorMsg);
        }
        attrLen -= (asPathLen * pathSegmentLength);
        ArrayList<Long> segmentAsNumbers = new ArrayList<>();
        while (pathSegmentLength-- > 0) {
          long asNumber;
          if (asPathLen == BgpConstants.Update.AS_4OCTET_LENGTH) {
            asNumber = message.readUnsignedInt();
          } else {
            asNumber = message.readUnsignedShort();
          }
          segmentAsNumbers.add(asNumber);
        }

        BgpRouteEntry.PathSegment pathSegment =
          new BgpRouteEntry.PathSegment((byte) pathSegmentType,
                                        segmentAsNumbers);
        pathSegments.add(pathSegment);
      }

      return new BgpRouteEntry.AsPath(pathSegments);
    }

    /**
     * Parses BGP UPDATE Attribute Type NEXT_HOP.
     *
     * @param bgpSession the BGP Session to use
     * @param ctx the Channel Handler Context
     * @param attrTypeCode the attribute type code
     * @param attrLen the attribute length (in octets)
```

```
 * @param attrFlags the attribute flags
 * @param message the message to parse
 * @return the parsed NEXT_HOP value
 * @throws BgpMessage.BgpParseException
 */
private static Ip4Address parseAttributeTypeNextHop(
                            BgpSession bgpSession,
                            ChannelHandlerContext ctx,
                            int attrTypeCode,
                            int attrLen,
                            int attrFlags,
                            ChannelBuffer message)
    throws BgpMessage.BgpParseException {

    // Check the Attribute Length
    if (attrLen != BgpConstants.Update.NextHop.LENGTH) {
        // ERROR: Attribute Length Error
        actionsBgpUpdateAttributeLengthError(
            bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
        String errorMsg = "Attribute Length Error";
        throw new BgpMessage.BgpParseException(errorMsg);
    }

    message.markReaderIndex();
    Ip4Address nextHopAddress =
        Ip4Address.valueOf((int) message.readUnsignedInt());
    //
    // Check whether the NEXT_HOP IP address is semantically correct.
    // As per RFC 4271, Section 6.3:
    //
    //  a) It MUST NOT be the IP address of the receiving speaker
    //  b) In the case of an EBGP ....
    //
    // Here we check only (a), because (b) doesn't apply for us: all our
    // peers are iBGP.
    //
    if (nextHopAddress.equals(bgpSession.localInfo().ip4Address())) {
        // ERROR: Invalid NEXT_HOP Attribute
        message.resetReaderIndex();
        actionsBgpUpdateInvalidNextHopAttribute(
            bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message,
            nextHopAddress);
        String errorMsg = "Invalid NEXT_HOP Attribute: " + nextHopAddress;
        throw new BgpMessage.BgpParseException(errorMsg);
    }

    return nextHopAddress;
}

/**
 * Parses BGP UPDATE Attribute Type MULTI_EXIT_DISC.
 *
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param attrTypeCode the attribute type code
 * @param attrLen the attribute length (in octets)
 * @param attrFlags the attribute flags
 * @param message the message to parse
 * @return the parsed MULTI_EXIT_DISC value
 * @throws BgpMessage.BgpParseException
```

```
 */
private static long parseAttributeTypeMultiExitDisc(
                        BgpSession bgpSession,
                        ChannelHandlerContext ctx,
                        int attrTypeCode,
                        int attrLen,
                        int attrFlags,
                        ChannelBuffer message)
    throws BgpMessage.BgpParseException {

    // Check the Attribute Length
    if (attrLen != BgpConstants.Update.MultiExitDisc.LENGTH) {
        // ERROR: Attribute Length Error
        actionsBgpUpdateAttributeLengthError(
            bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
        String errorMsg = "Attribute Length Error";
        throw new BgpMessage.BgpParseException(errorMsg);
    }

    long multiExitDisc = message.readUnsignedInt();
    return multiExitDisc;
}

/**
 * Parses BGP UPDATE Attribute Type LOCAL_PREF.
 *
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param attrTypeCode the attribute type code
 * @param attrLen the attribute length (in octets)
 * @param attrFlags the attribute flags
 * @param message the message to parse
 * @return the parsed LOCAL_PREF value
 * @throws BgpMessage.BgpParseException
 */
private static long parseAttributeTypeLocalPref(
                        BgpSession bgpSession,
                        ChannelHandlerContext ctx,
                        int attrTypeCode,
                        int attrLen,
                        int attrFlags,
                        ChannelBuffer message)
    throws BgpMessage.BgpParseException {

    // Check the Attribute Length
    if (attrLen != BgpConstants.Update.LocalPref.LENGTH) {
        // ERROR: Attribute Length Error
        actionsBgpUpdateAttributeLengthError(
            bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
        String errorMsg = "Attribute Length Error";
        throw new BgpMessage.BgpParseException(errorMsg);
    }

    long localPref = message.readUnsignedInt();
    return localPref;
}

/**
 * Parses BGP UPDATE Attribute Type ATOMIC_AGGREGATE.
 *
```

```
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param attrTypeCode the attribute type code
 * @param attrLen the attribute length (in octets)
 * @param attrFlags the attribute flags
 * @param message the message to parse
 * @throws BgpMessage.BgpParseException
 */
private static void parseAttributeTypeAtomicAggregate(
                    BgpSession bgpSession,
                    ChannelHandlerContext ctx,
                    int attrTypeCode,
                    int attrLen,
                    int attrFlags,
                    ChannelBuffer message)
    throws BgpMessage.BgpParseException {

    // Check the Attribute Length
    if (attrLen != BgpConstants.Update.AtomicAggregate.LENGTH) {
        // ERROR: Attribute Length Error
        actionsBgpUpdateAttributeLengthError(
            bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
        String errorMsg = "Attribute Length Error";
        throw new BgpMessage.BgpParseException(errorMsg);
    }

    // Nothing to do: this attribute is primarily informational
}

/**
 * Parses BGP UPDATE Attribute Type AGGREGATOR.
 *
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param attrTypeCode the attribute type code
 * @param attrLen the attribute length (in octets)
 * @param attrFlags the attribute flags
 * @param message the message to parse
 * @return the parsed AGGREGATOR value: a tuple of <AS-Number, IP-Address>
 * @throws BgpMessage.BgpParseException
 */
private static Pair<Long, Ip4Address> parseAttributeTypeAggregator(
                                BgpSession bgpSession,
                                ChannelHandlerContext ctx,
                                int attrTypeCode,
                                int attrLen,
                                int attrFlags,
                                ChannelBuffer message)
    throws BgpMessage.BgpParseException {
    int expectedAttrLen;

    if (bgpSession.isAs4OctetCapable()) {
        expectedAttrLen = BgpConstants.Update.Aggregator.AS4_LENGTH;
    } else {
        expectedAttrLen = BgpConstants.Update.Aggregator.AS2_LENGTH;
    }

    // Check the Attribute Length
    if (attrLen != expectedAttrLen) {
        // ERROR: Attribute Length Error
```

```
        actionsBgpUpdateAttributeLengthError(
            bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
        String errorMsg = "Attribute Length Error";
        throw new BgpMessage.BgpParseException(errorMsg);
    }

    // The AGGREGATOR AS number
    long aggregatorAsNumber;
    if (bgpSession.isAs4OctetCapable()) {
        aggregatorAsNumber = message.readUnsignedInt();
    } else {
        aggregatorAsNumber = message.readUnsignedShort();
    }
    // The AGGREGATOR IP address
    Ip4Address aggregatorIpAddress =
        Ip4Address.valueOf((int) message.readUnsignedInt());

    Pair<Long, Ip4Address> aggregator = Pair.of(aggregatorAsNumber,
                                        aggregatorIpAddress);
    return aggregator;
}


/**
 * Parses BGP UPDATE Attribute Type COMMUNITY.
 *
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param attrTypeCode the attribute type code
 * @param attrLen the attribute length (in octets)
 * @param attrFlags the attribute flags
 * @param message the message to parse
 * @throws BgpMessage.BgpParseException
 */
private static BgpRouteEntry.Communities parseAttributeTypeCommunity(
        BgpSession bgpSession,
        ChannelHandlerContext ctx,
        int attrTypeCode,
        int attrLen,
        int attrFlags,
        ChannelBuffer message)
        throws BgpMessage.BgpParseException {

    ArrayList<BgpRouteEntry.Community> communityList = new ArrayList<>();

    // Parse the message
    while (attrLen > 0) {
        // Check the Attribute Length
        if (attrLen < BgpConstants.Update.Community.MIN_COMM_LENGTH) {
            // ERROR: Attribute Length Error
            actionsBgpUpdateAttributeLengthError(
                    bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
            String errorMsg = "Attribute Length Error";
            throw new BgpMessage.BgpParseException(errorMsg);
        }

        //The COMMUNITY AS Number
        long communityAsNumber = message.readUnsignedShort();
        //The COMMUNITY Local Identifier
        long communityLocalIdentifier = message.readUnsignedShort();
```

```
        attrLen -= 4;

        // The Community Attribute
        BgpRouteEntry.Community community = new BgpRouteEntry.Community(
              Pair.of(communityAsNumber, communityLocalIdentifier));
        communityList.add(community);
    }
    return new BgpRouteEntry.Communities(communityList);
}

/**
 * Parses BGP UPDATE Attribute Type EXTENDED COMMUNITY.
 *
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param attrTypeCode the attribute type code
 * @param attrLen the attribute length (in octets)
 * @param attrFlags the attribute flags
 * @param message the message to parse
 * @throws BgpMessage.BgpParseException
 */
private static BgpRouteEntry.extCommunities parseAttributeTypeExtCommunity(
        BgpSession bgpSession,
        ChannelHandlerContext ctx,
        int attrTypeCode,
        int attrLen,
        int attrFlags,
        ChannelBuffer message)
        throws BgpMessage.BgpParseException {

    ArrayList<BgpRouteEntry.extCommunity> extCommunityList = new ArrayList<>();

    // Parse the message
    while (attrLen > 0) {
        // Check the Attribute Length
        if (attrLen < BgpConstants.Update.ExtendedCommunity.MIN_COMM_LENGTH) {
            // ERROR: Attribute Length Error
            actionsBgpUpdateAttributeLengthError(
                  bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
            String errorMsg = "Attribute Length Error";
            throw new BgpMessage.BgpParseException(errorMsg);
        }

        //The EXTENDED COMMUNITY Value One
        long extCommunityValueOne = message.readUnsignedShort();
        //The EXTENDED COMMUNITY Value Two
        long extCommunityValueTwo = message.readUnsignedShort();
        //The EXTENDED COMMUNITY Value Three
        long extCommunityValueThree = message.readUnsignedInt();
        attrLen -= 8;

        // The Community Attribute
        BgpRouteEntry.extCommunity extCommunity =
              new BgpRouteEntry.extCommunity(Triple.of(extCommunityValueOne,
                                          extCommunityValueTwo,
                                          extCommunityValueThree));
        extCommunityList.add(extCommunity);
    }
    return new BgpRouteEntry.extCommunities(extCommunityList);
}
```

```
/**
 * Parses BGP UPDATE Attribute Type MP_REACH_NLRI.
 *
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param attrTypeCode the attribute type code
 * @param attrLen the attribute length (in octets)
 * @param attrFlags the attribute flags
 * @param message the message to parse
 * @return the parsed MP_REACH_NLRI information if recognized, otherwise
 * null
 * @throws BgpMessage.BgpParseException
 */
private static MpNlri parseAttributeTypeMpReachNlri(
                                    BgpSession bgpSession,
                                    ChannelHandlerContext ctx,
                                    int attrTypeCode,
                                    int attrLen,
                                    int attrFlags,
                                    ChannelBuffer message)
    throws BgpMessage.BgpParseException {
    int attributeEnd = message.readerIndex() + attrLen;

    // Check the Attribute Length
    if (attrLen < BgpConstants.Update.MpReachNlri.MIN_LENGTH) {
        // ERROR: Attribute Length Error
        actionsBgpUpdateAttributeLengthError(
            bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
        String errorMsg = "Attribute Length Error";
        throw new BgpMessage.BgpParseException(errorMsg);
    }

    message.markReaderIndex();
    int afi = message.readUnsignedShort();
    int safi = message.readUnsignedByte();
    int nextHopLen = message.readUnsignedByte();

    //
    // Verify the AFI/SAFI, and skip the attribute if not recognized.
    // NOTE: Currently, we support only IPv4/IPv6 UNICAST
    //
    if (((afi != BgpConstants.Open.Capabilities.MultiprotocolExtensions.AFI_IPV4) &&
         (afi != BgpConstants.Open.Capabilities.MultiprotocolExtensions.AFI_IPV6)) ||
        (safi != BgpConstants.Open.Capabilities.MultiprotocolExtensions.SAFI_UNICAST)) {
        // Skip the attribute
        message.resetReaderIndex();
        message.skipBytes(attrLen);
        return null;
    }

    //
    // Verify the next-hop length
    //
    int expectedNextHopLen = 0;
    switch (afi) {
    case BgpConstants.Open.Capabilities.MultiprotocolExtensions.AFI_IPV4:
        expectedNextHopLen = Ip4Address.BYTE_LENGTH;
        break;
```

```
case BgpConstants.Open.Capabilities.MultiprotocolExtensions.AFI_IPV6:
   expectedNextHopLen = Ip6Address.BYTE_LENGTH;
   break;
default:
   // UNREACHABLE
   break;
}
if (nextHopLen != expectedNextHopLen) {
   // ERROR: Optional Attribute Error
   message.resetReaderIndex();
   actionsBgpUpdateOptionalAttributeError(
      bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
   String errorMsg = "Invalid next-hop network address length. " +
      "Received " + nextHopLen + " expected " + expectedNextHopLen;
   throw new BgpMessage.BgpParseException(errorMsg);
}
// NOTE: We use "+ 1" to take into account the Reserved field (1 octet)
if (message.readerIndex() + nextHopLen + 1 >= attributeEnd) {
   // ERROR: Optional Attribute Error
   message.resetReaderIndex();
   actionsBgpUpdateOptionalAttributeError(
      bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
   String errorMsg = "Malformed next-hop network address";
   throw new BgpMessage.BgpParseException(errorMsg);
}

//
// Get the Next-hop address, skip the Reserved field, and get the NLRI
//
byte[] nextHopBuffer = new byte[nextHopLen];
message.readBytes(nextHopBuffer, 0, nextHopLen);
int reserved = message.readUnsignedByte();
MpNlri mpNlri = new MpNlri(afi, safi);
try {
   switch (afi) {
   case BgpConstants.Open.Capabilities.MultiprotocolExtensions.AFI_IPV4:
      // The next-hop address
      mpNlri.nextHop4 = Ip4Address.valueOf(nextHopBuffer);
      // The NLRI
      mpNlri.nlri4 = parsePackedIp4Prefixes(
                        attributeEnd - message.readerIndex(),
                        message);
      break;
   case BgpConstants.Open.Capabilities.MultiprotocolExtensions.AFI_IPV6:
      // The next-hop address
      mpNlri.nextHop6 = Ip6Address.valueOf(nextHopBuffer);
      // The NLRI
      mpNlri.nlri6 = parsePackedIp6Prefixes(
                        attributeEnd - message.readerIndex(),
                        message);
      break;
   default:
      // UNREACHABLE
      break;
   }
} catch (BgpMessage.BgpParseException e) {
   // ERROR: Optional Attribute Error
   message.resetReaderIndex();
   actionsBgpUpdateOptionalAttributeError(
      bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
```

```
        String errorMsg = "Malformed network layer reachability information";
        throw new BgpMessage.BgpParseException(errorMsg);
    }

    return mpNlri;
}

/**
 * Parses BGP UPDATE Attribute Type MP_UNREACH_NLRI.
 *
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param attrTypeCode the attribute type code
 * @param attrLen the attribute length (in octets)
 * @param attrFlags the attribute flags
 * @param message the message to parse
 * @return the parsed MP_UNREACH_NLRI information if recognized, otherwise
 * null
 * @throws BgpMessage.BgpParseException
 */
private static MpNlri parseAttributeTypeMpUnreachNlri(
                                    BgpSession bgpSession,
                                    ChannelHandlerContext ctx,
                                    int attrTypeCode,
                                    int attrLen,
                                    int attrFlags,
                                    ChannelBuffer message)
    throws BgpMessage.BgpParseException {
    int attributeEnd = message.readerIndex() + attrLen;

    // Check the Attribute Length
    if (attrLen < BgpConstants.Update.MpUnreachNlri.MIN_LENGTH) {
        // ERROR: Attribute Length Error
        actionsBgpUpdateAttributeLengthError(
            bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
        String errorMsg = "Attribute Length Error";
        throw new BgpMessage.BgpParseException(errorMsg);
    }

    message.markReaderIndex();
    int afi = message.readUnsignedShort();
    int safi = message.readUnsignedByte();

    //
    // Verify the AFI/SAFI, and skip the attribute if not recognized.
    // NOTE: Currently, we support only IPv4/IPv6 UNICAST
    //
    if (((afi != BgpConstants.Open.Capabilities.MultiprotocolExtensions.AFI_IPV4) &&
         (afi != BgpConstants.Open.Capabilities.MultiprotocolExtensions.AFI_IPV6)) ||
        (safi != BgpConstants.Open.Capabilities.MultiprotocolExtensions.SAFI_UNICAST)) {
        // Skip the attribute
        message.resetReaderIndex();
        message.skipBytes(attrLen);
        return null;
    }

    //
    // Get the Withdrawn Routes
    //
    MpNlri mpNlri = new MpNlri(afi, safi);
```

```
    try {
        switch (afi) {
        case BgpConstants.Open.Capabilities.MultiprotocolExtensions.AFI_IPV4:
            // The Withdrawn Routes
            mpNlri.nlri4 = parsePackedIp4Prefixes(
                                attributeEnd - message.readerIndex(),
                                message);
            break;
        case BgpConstants.Open.Capabilities.MultiprotocolExtensions.AFI_IPV6:
            // The Withdrawn Routes
            mpNlri.nlri6 = parsePackedIp6Prefixes(
                                attributeEnd - message.readerIndex(),
                                message);
            break;
        default:
            // UNREACHABLE
            break;
        }
    } catch (BgpMessage.BgpParseException e) {
        // ERROR: Optional Attribute Error
        message.resetReaderIndex();
        actionsBgpUpdateOptionalAttributeError(
            bgpSession, ctx, attrTypeCode, attrLen, attrFlags, message);
        String errorMsg = "Malformed withdrawn routes";
        throw new BgpMessage.BgpParseException(errorMsg);
    }

    return mpNlri;
}

/**
 * Parses a message that contains encoded IPv4 network prefixes.
 * <p>
 * The IPv4 prefixes are encoded in the form:
 * <Length, Prefix> where Length is the length in bits of the IPv4 prefix,
 * and Prefix is the IPv4 prefix (padded with trailing bits to the end
 * of an octet).
 *
 * @param totalLength the total length of the data to parse
 * @param message the message with data to parse
 * @return a collection of parsed IPv4 network prefixes
 * @throws BgpMessage.BgpParseException
 */
private static Collection<Ip4Prefix> parsePackedIp4Prefixes(
                                    int totalLength,
                                    ChannelBuffer message)
    throws BgpMessage.BgpParseException {
    Collection<Ip4Prefix> result = new ArrayList<>();

    if (totalLength == 0) {
        return result;
    }

    // Parse the data
    byte[] buffer = new byte[Ip4Address.BYTE_LENGTH];
    int dataEnd = message.readerIndex() + totalLength;
    while (message.readerIndex() < dataEnd) {
        int prefixBitlen = message.readUnsignedByte();
        int prefixBytelen = (prefixBitlen + 7) / 8;     // Round-up
        if (message.readerIndex() + prefixBytelen > dataEnd) {
```

```
            String errorMsg = "Malformed Network Prefixes";
            throw new BgpMessage.BgpParseException(errorMsg);
        }

        message.readBytes(buffer, 0, prefixBytelen);
        Ip4Prefix prefix = Ip4Prefix.valueOf(Ip4Address.valueOf(buffer),
                                             prefixBitlen);
        result.add(prefix);
    }

    return result;
}

/**
 * Parses a message that contains encoded IPv6 network prefixes.
 * <p>
 * The IPv6 prefixes are encoded in the form:
 * <Length, Prefix> where Length is the length in bits of the IPv6 prefix,
 * and Prefix is the IPv6 prefix (padded with trailing bits to the end
 * of an octet).
 *
 * @param totalLength the total length of the data to parse
 * @param message the message with data to parse
 * @return a collection of parsed IPv6 network prefixes
 * @throws BgpMessage.BgpParseException
 */
private static Collection<Ip6Prefix> parsePackedIp6Prefixes(
                                    int totalLength,
                                    ChannelBuffer message)
    throws BgpMessage.BgpParseException {
    Collection<Ip6Prefix> result = new ArrayList<>();

    if (totalLength == 0) {
        return result;
    }

    // Parse the data
    byte[] buffer = new byte[Ip6Address.BYTE_LENGTH];
    int dataEnd = message.readerIndex() + totalLength;
    while (message.readerIndex() < dataEnd) {
        int prefixBitlen = message.readUnsignedByte();
        int prefixBytelen = (prefixBitlen + 7) / 8;     // Round-up
        if (message.readerIndex() + prefixBytelen > dataEnd) {
            String errorMsg = "Malformed Network Prefixes";
            throw new BgpMessage.BgpParseException(errorMsg);
        }

        message.readBytes(buffer, 0, prefixBytelen);
        Ip6Prefix prefix = Ip6Prefix.valueOf(Ip6Address.valueOf(buffer),
                                             prefixBitlen);
        result.add(prefix);
    }

    return result;
}

/**
 * Applies the appropriate actions after detecting BGP UPDATE
 * Invalid Network Field Error: send NOTIFICATION and close the channel.
 *
```

```
     * @param bgpSession the BGP Session to use
     * @param ctx the Channel Handler Context
     */
    private static void actionsBgpUpdateInvalidNetworkField(
                            BgpSession bgpSession,
                            ChannelHandlerContext ctx) {
        log.debug("BGP RX UPDATE Error from {}: Invalid Network Field",
                bgpSession.remoteInfo().address());

        //
        // ERROR: Invalid Network Field
        //
        // Send NOTIFICATION and close the connection
        int errorCode = BgpConstants.Notifications.UpdateMessageError.ERROR_CODE;
        int                                                 errorSubcode                                =
BgpConstants.Notifications.UpdateMessageError.INVALID_NETWORK_FIELD;
        ChannelBuffer txMessage =
            BgpNotification.prepareBgpNotification(errorCode, errorSubcode,
                                       null);
        ctx.getChannel().write(txMessage);
        bgpSession.closeSession(ctx);
    }

    /**
     * Applies the appropriate actions after detecting BGP UPDATE
     * Malformed Attribute List Error: send NOTIFICATION and close the channel.
     *
     * @param bgpSession the BGP Session to use
     * @param ctx the Channel Handler Context
     */
    private static void actionsBgpUpdateMalformedAttributeList(
                            BgpSession bgpSession,
                            ChannelHandlerContext ctx) {
        log.debug("BGP RX UPDATE Error from {}: Malformed Attribute List",
                bgpSession.remoteInfo().address());

        //
        // ERROR: Malformed Attribute List
        //
        // Send NOTIFICATION and close the connection
        int errorCode = BgpConstants.Notifications.UpdateMessageError.ERROR_CODE;
        int                                                 errorSubcode                                =
BgpConstants.Notifications.UpdateMessageError.MALFORMED_ATTRIBUTE_LIST;
        ChannelBuffer txMessage =
            BgpNotification.prepareBgpNotification(errorCode, errorSubcode,
                                       null);
        ctx.getChannel().write(txMessage);
        bgpSession.closeSession(ctx);
    }

    /**
     * Applies the appropriate actions after detecting BGP UPDATE
     * Missing Well-known Attribute Error: send NOTIFICATION and close the
     * channel.
     *
     * @param bgpSession the BGP Session to use
     * @param ctx the Channel Handler Context
     * @param missingAttrTypeCode the missing attribute type code
     */
    private static void actionsBgpUpdateMissingWellKnownAttribute(
```

```
                        BgpSession bgpSession,
                        ChannelHandlerContext ctx,
                        int missingAttrTypeCode) {
        log.debug("BGP RX UPDATE Error from {}: Missing Well-known Attribute: {}",
                bgpSession.remoteInfo().address(), missingAttrTypeCode);

        //
        // ERROR: Missing Well-known Attribute
        //
        // Send NOTIFICATION and close the connection
        int errorCode = BgpConstants.Notifications.UpdateMessageError.ERROR_CODE;
        int                                                errorSubcode                                                =
BgpConstants.Notifications.UpdateMessageError.MISSING_WELL_KNOWN_ATTRIBUTE;
        ChannelBuffer data = ChannelBuffers.buffer(1);
        data.writeByte(missingAttrTypeCode);
        ChannelBuffer txMessage =
            BgpNotification.prepareBgpNotification(errorCode, errorSubcode,
                                        data);
        ctx.getChannel().write(txMessage);
        bgpSession.closeSession(ctx);
    }

    /**
     * Applies the appropriate actions after detecting BGP UPDATE
     * Invalid ORIGIN Attribute Error: send NOTIFICATION and close the channel.
     *
     * @param bgpSession the BGP Session to use
     * @param ctx the Channel Handler Context
     * @param attrTypeCode the attribute type code
     * @param attrLen the attribute length (in octets)
     * @param attrFlags the attribute flags
     * @param message the message with the data
     * @param origin the ORIGIN attribute value
     */
    private static void actionsBgpUpdateInvalidOriginAttribute(
                        BgpSession bgpSession,
                        ChannelHandlerContext ctx,
                        int attrTypeCode,
                        int attrLen,
                        int attrFlags,
                        ChannelBuffer message,
                        short origin) {
        log.debug("BGP RX UPDATE Error from {}: Invalid ORIGIN Attribute",
                bgpSession.remoteInfo().address());

        //
        // ERROR: Invalid ORIGIN Attribute
        //
        // Send NOTIFICATION and close the connection
        int errorCode = BgpConstants.Notifications.UpdateMessageError.ERROR_CODE;
        int                                                errorSubcode                                                =
BgpConstants.Notifications.UpdateMessageError.INVALID_ORIGIN_ATTRIBUTE;
        ChannelBuffer data =
            prepareBgpUpdateNotificationDataPayload(attrTypeCode, attrLen,
                                        attrFlags, message);
        ChannelBuffer txMessage =
            BgpNotification.prepareBgpNotification(errorCode, errorSubcode,
                                        data);
        ctx.getChannel().write(txMessage);
        bgpSession.closeSession(ctx);
```

```
    }

    /**
     * Applies the appropriate actions after detecting BGP UPDATE
     * Attribute Flags Error: send NOTIFICATION and close the channel.
     *
     * @param bgpSession the BGP Session to use
     * @param ctx the Channel Handler Context
     * @param attrTypeCode the attribute type code
     * @param attrLen the attribute length (in octets)
     * @param attrFlags the attribute flags
     * @param message the message with the data
     */
    private static void actionsBgpUpdateAttributeFlagsError(
                            BgpSession bgpSession,
                            ChannelHandlerContext ctx,
                            int attrTypeCode,
                            int attrLen,
                            int attrFlags,
                            ChannelBuffer message) {
        log.debug("BGP RX UPDATE Error from {}: Attribute Flags Error",
                bgpSession.remoteInfo().address());

        //
        // ERROR: Attribute Flags Error
        //
        // Send NOTIFICATION and close the connection
        int errorCode = BgpConstants.Notifications.UpdateMessageError.ERROR_CODE;
        int                                                errorSubcode                                =
BgpConstants.Notifications.UpdateMessageError.ATTRIBUTE_FLAGS_ERROR;
        ChannelBuffer data =
            prepareBgpUpdateNotificationDataPayload(attrTypeCode, attrLen,
                                    attrFlags, message);
        ChannelBuffer txMessage =
            BgpNotification.prepareBgpNotification(errorCode, errorSubcode,
                                    data);
        ctx.getChannel().write(txMessage);
        bgpSession.closeSession(ctx);
    }

    /**
     * Applies the appropriate actions after detecting BGP UPDATE
     * Invalid NEXT_HOP Attribute Error: send NOTIFICATION and close the
     * channel.
     *
     * @param bgpSession the BGP Session to use
     * @param ctx the Channel Handler Context
     * @param attrTypeCode the attribute type code
     * @param attrLen the attribute length (in octets)
     * @param attrFlags the attribute flags
     * @param message the message with the data
     * @param nextHop the NEXT_HOP attribute value
     */
    private static void actionsBgpUpdateInvalidNextHopAttribute(
                            BgpSession bgpSession,
                            ChannelHandlerContext ctx,
                            int attrTypeCode,
                            int attrLen,
                            int attrFlags,
                            ChannelBuffer message,
```

```
                    Ip4Address nextHop) {
        log.debug("BGP RX UPDATE Error from {}: Invalid NEXT_HOP Attribute {}",
                bgpSession.remoteInfo().address(), nextHop);

        //
        // ERROR: Invalid NEXT_HOP Attribute
        //
        // Send NOTIFICATION and close the connection
        int errorCode = BgpConstants.Notifications.UpdateMessageError.ERROR_CODE;
        int                                                errorSubcode                                                =
BgpConstants.Notifications.UpdateMessageError.INVALID_NEXT_HOP_ATTRIBUTE;
        ChannelBuffer data =
            prepareBgpUpdateNotificationDataPayload(attrTypeCode, attrLen,
                                    attrFlags, message);
        ChannelBuffer txMessage =
            BgpNotification.prepareBgpNotification(errorCode, errorSubcode,
                                    data);
        ctx.getChannel().write(txMessage);
        bgpSession.closeSession(ctx);
    }

    /**
     * Applies the appropriate actions after detecting BGP UPDATE
     * Unrecognized Well-known Attribute Error: send NOTIFICATION and close
     * the channel.
     *
     * @param bgpSession the BGP Session to use
     * @param ctx the Channel Handler Context
     * @param attrTypeCode the attribute type code
     * @param attrLen the attribute length (in octets)
     * @param attrFlags the attribute flags
     * @param message the message with the data
     */
    private static void actionsBgpUpdateUnrecognizedWellKnownAttribute(
                    BgpSession bgpSession,
                    ChannelHandlerContext ctx,
                    int attrTypeCode,
                    int attrLen,
                    int attrFlags,
                    ChannelBuffer message) {
        log.debug("BGP RX UPDATE Error from {}: " +
                "Unrecognized Well-known Attribute Error: {}",
                bgpSession.remoteInfo().address(), attrTypeCode);

        //
        // ERROR: Unrecognized Well-known Attribute
        //
        // Send NOTIFICATION and close the connection
        int errorCode = BgpConstants.Notifications.UpdateMessageError.ERROR_CODE;
        int errorSubcode =

BgpConstants.Notifications.UpdateMessageError.UNRECOGNIZED_WELL_KNOWN_ATTRIBUTE;
        ChannelBuffer data =
            prepareBgpUpdateNotificationDataPayload(attrTypeCode, attrLen,
                                    attrFlags, message);
        ChannelBuffer txMessage =
            BgpNotification.prepareBgpNotification(errorCode, errorSubcode,
                                    data);
        ctx.getChannel().write(txMessage);
        bgpSession.closeSession(ctx);
```

```
}

/**
 * Applies the appropriate actions after detecting BGP UPDATE
 * Optional Attribute Error: send NOTIFICATION and close
 * the channel.
 *
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param attrTypeCode the attribute type code
 * @param attrLen the attribute length (in octets)
 * @param attrFlags the attribute flags
 * @param message the message with the data
 */
private static void actionsBgpUpdateOptionalAttributeError(
                        BgpSession bgpSession,
                        ChannelHandlerContext ctx,
                        int attrTypeCode,
                        int attrLen,
                        int attrFlags,
                        ChannelBuffer message) {
    log.debug("BGP RX UPDATE Error from {}: Optional Attribute Error: {}",
            bgpSession.remoteInfo().address(), attrTypeCode);

    //
    // ERROR: Optional Attribute Error
    //
    // Send NOTIFICATION and close the connection
    int errorCode = BgpConstants.Notifications.UpdateMessageError.ERROR_CODE;
    int errorSubcode =
        BgpConstants.Notifications.UpdateMessageError.OPTIONAL_ATTRIBUTE_ERROR;
    ChannelBuffer data =
        prepareBgpUpdateNotificationDataPayload(attrTypeCode, attrLen,
                                        attrFlags, message);
    ChannelBuffer txMessage =
        BgpNotification.prepareBgpNotification(errorCode, errorSubcode,
                                       data);
    ctx.getChannel().write(txMessage);
    bgpSession.closeSession(ctx);
}

/**
 * Applies the appropriate actions after detecting BGP UPDATE
 * Attribute Length Error: send NOTIFICATION and close the channel.
 *
 * @param bgpSession the BGP Session to use
 * @param ctx the Channel Handler Context
 * @param attrTypeCode the attribute type code
 * @param attrLen the attribute length (in octets)
 * @param attrFlags the attribute flags
 * @param message the message with the data
 */
private static void actionsBgpUpdateAttributeLengthError(
                        BgpSession bgpSession,
                        ChannelHandlerContext ctx,
                        int attrTypeCode,
                        int attrLen,
                        int attrFlags,
                        ChannelBuffer message) {
    log.debug("BGP RX UPDATE Error from {}: Attribute Length Error",
```

```
            bgpSession.remoteInfo().address());

        //
        // ERROR: Attribute Length Error
        //
        // Send NOTIFICATION and close the connection
        int errorCode = BgpConstants.Notifications.UpdateMessageError.ERROR_CODE;
        int                                                errorSubcode                                                =
BgpConstants.Notifications.UpdateMessageError.ATTRIBUTE_LENGTH_ERROR;
        ChannelBuffer data =
            prepareBgpUpdateNotificationDataPayload(attrTypeCode, attrLen,
                                    attrFlags, message);
        ChannelBuffer txMessage =
            BgpNotification.prepareBgpNotification(errorCode, errorSubcode,
                                    data);
        ctx.getChannel().write(txMessage);
        bgpSession.closeSession(ctx);
    }

    /**
     * Applies the appropriate actions after detecting BGP UPDATE
     * Malformed AS_PATH Error: send NOTIFICATION and close the channel.
     *
     * @param bgpSession the BGP Session to use
     * @param ctx the Channel Handler Context
     */
    private static void actionsBgpUpdateMalformedAsPath(
                        BgpSession bgpSession,
                        ChannelHandlerContext ctx) {
        log.debug("BGP RX UPDATE Error from {}: Malformed AS Path",
                bgpSession.remoteInfo().address());

        //
        // ERROR: Malformed AS_PATH
        //
        // Send NOTIFICATION and close the connection
        int errorCode = BgpConstants.Notifications.UpdateMessageError.ERROR_CODE;
        int errorSubcode = BgpConstants.Notifications.UpdateMessageError.MALFORMED_AS_PATH;
        ChannelBuffer txMessage =
            BgpNotification.prepareBgpNotification(errorCode, errorSubcode,
                                    null);
        ctx.getChannel().write(txMessage);
        bgpSession.closeSession(ctx);
    }

    /**
     * Prepares BGP UPDATE Notification data payload.
     *
     * @param attrTypeCode the attribute type code
     * @param attrLen the attribute length (in octets)
     * @param attrFlags the attribute flags
     * @param message the message with the data
     * @return the buffer with the data payload for the BGP UPDATE Notification
     */
    private static ChannelBuffer prepareBgpUpdateNotificationDataPayload(
                                int attrTypeCode,
                                int attrLen,
                                int attrFlags,
                                ChannelBuffer message) {
        // Compute the attribute length field octets
```

```
        boolean extendedLengthBit = ((0x10 & attrFlags) != 0);
        int attrLenOctets = 1;
        if (extendedLengthBit) {
            attrLenOctets = 2;
        }
        ChannelBuffer data =
            ChannelBuffers.buffer(attrLen + attrLenOctets + 1);
        data.writeByte(attrTypeCode);
        if (extendedLengthBit) {
            data.writeShort(attrLen);
        } else {
            data.writeByte(attrLen);
        }
        data.writeBytes(message, attrLen);
        return data;
    }

    /**
     * Helper class for storing Multiprotocol Network Layer Reachability
     * information.
     */
    private static final class MpNlri {
        private final int afi;
        private final int safi;
        private Ip4Address nextHop4;
        private Ip6Address nextHop6;
        private Collection<Ip4Prefix> nlri4 = new ArrayList<>();
        private Collection<Ip6Prefix> nlri6 = new ArrayList<>();

        /**
         * Constructor.
         *
         * @param afi the Address Family Identifier
         * @param safi the Subsequent Address Family Identifier
         */
        private MpNlri(int afi, int safi) {
            this.afi = afi;
            this.safi = safi;
        }
    }

    /**
     * Helper class for storing decoded BGP routing information.
     */
    private static final class DecodedBgpRoutes {
        private final Map<Ip4Prefix, BgpRouteEntry> addedUnicastRoutes4 =
            new HashMap<>();
        private final Map<Ip6Prefix, BgpRouteEntry> addedUnicastRoutes6 =
            new HashMap<>();
        private final Map<Ip4Prefix, BgpRouteEntry> deletedUnicastRoutes4 =
            new HashMap<>();
        private final Map<Ip6Prefix, BgpRouteEntry> deletedUnicastRoutes6 =
            new HashMap<>();
    }
}
```

Appendix 3 – BgpRouteEntry Module

BgpRouteEntry.java

```
/*
 * Copyright 2017-present Open Networking Foundation
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.onosproject.routing.bgp;

import com.google.common.base.MoreObjects;
import org.apache.commons.lang3.tuple.Pair;
import org.apache.commons.lang3.tuple.Triple;
import org.javatuples.Quartet;
import org.onlab.packet.Ip4Address;
import org.onlab.packet.IpAddress;
import org.onlab.packet.IpPrefix;

import java.util.ArrayList;
import java.util.Objects;

import static com.google.common.base.Preconditions.checkNotNull;

/**
 * Represents a route in BGP.
 */
public class BgpRouteEntry extends RouteEntry {
    private final BgpSession bgpSession; // The BGP Session the route was
                                         // received on
    private final byte origin;          // Route ORIGIN: IGP, EGP, INCOMPLETE
    private final AsPath asPath;        // The AS Path
    private final long localPref;       // The local preference for the route
    private long multiExitDisc = BgpConstants.Update.MultiExitDisc.LOWEST_MULTI_EXIT_DISC;
    //private final Pair<Long, Long> community;
    private final Communities communities;
    private final extCommunities extCommunities;

    /**
     * Class constructor.
     *
     * @param bgpSession the BGP Session the route was received on
     * @param prefix the prefix of the route
     * @param nextHop the next hop of the route
     * @param origin the route origin: 0=IGP, 1=EGP, 2=INCOMPLETE
     * @param asPath the AS path
     * @param localPref the route local preference
```

```
 * @param communities the basic community attribute
 * @param extCommunities the extended community attribute
 */
public BgpRouteEntry(BgpSession bgpSession, IpPrefix prefix,
                     IpAddress nextHop, byte origin,
                     BgpRouteEntry.AsPath asPath, long localPref,
                     Communities communities, extCommunities extCommunities) {
   super(prefix, nextHop);
   this.bgpSession = checkNotNull(bgpSession);
   this.origin = origin;
   this.asPath = checkNotNull(asPath);
   this.localPref = localPref;
   this.communities = communities;
   this.extCommunities = extCommunities;
}

/**
 * Gets the BGP Session the route was received on.
 *
 * @return the BGP Session the route was received on
 */
public BgpSession getBgpSession() {
   return bgpSession;
}

/**
 * Gets the route origin: 0=IGP, 1=EGP, 2=INCOMPLETE.
 *
 * @return the route origin: 0=IGP, 1=EGP, 2=INCOMPLETE
 */
public byte getOrigin() {
   return origin;
}

/**
 * Gets the route AS path.
 *
 * @return the route AS path
 */
public BgpRouteEntry.AsPath getAsPath() {
   return asPath;
}

/**
 * Gets the route local preference.
 *
 * @return the route local preference
 */
public long getLocalPref() {
   return localPref;
}

/**
 * Gets the route MED (Multi-Exit Discriminator).
 *
 * @return the route MED (Multi-Exit Discriminator)
 */
public long getMultiExitDisc() {
   return multiExitDisc;
}
```

```
/**
 * Sets the route MED (Multi-Exit Discriminator).
 *
 * @param multiExitDisc the route MED (Multi-Exit Discriminator) to set
 */
void setMultiExitDisc(long multiExitDisc) {
   this.multiExitDisc = multiExitDisc;
}

/**
 * Gets the route community.
 *
 * @return the route community
 */
public BgpRouteEntry.Communities getCommunities() {
   return communities;
}

/**
 * Gets the route extended community.
 *
 * @return the route extended community
 */
public BgpRouteEntry.extCommunities getExtendedCommunities() {
   return extCommunities;
}


/**
 * Tests whether the route is originated from the local AS.
 * <p>
 * The route is considered originated from the local AS if the AS Path
 * is empty or if it begins with an AS_SET (after skipping
 * AS_CONFED_SEQUENCE and AS_CONFED_SET).
 * </p>
 *
 * @return true if the route is originated from the local AS, otherwise
 * false
 */
boolean isLocalRoute() {
   PathSegment firstPathSegment = null;

   // Find the first Path Segment by ignoring the AS_CONFED_* segments
   for (PathSegment pathSegment : asPath.getPathSegments()) {
      if ((pathSegment.getType() == BgpConstants.Update.AsPath.AS_SET) ||
         (pathSegment.getType() == BgpConstants.Update.AsPath.AS_SEQUENCE)) {
         firstPathSegment = pathSegment;
         break;
      }
   }
   if (firstPathSegment == null) {
      return true;              // Local route: no path segments
   }
   // If the first path segment is AS_SET, the route is considered local
   if (firstPathSegment.getType() == BgpConstants.Update.AsPath.AS_SET) {
      return true;
   }

   return false;                // The route is not local
```

```
}

/**
 * Gets the BGP Neighbor AS number the route was received from.
 * <p>
 * If the router is originated from the local AS, the return value is
 * zero (BGP_AS_0).
 * </p>
 *
 * @return the BGP Neighbor AS number the route was received from.
 */
long getNeighborAs() {
   PathSegment firstPathSegment = null;

   if (isLocalRoute()) {
      return BgpConstants.BGP_AS_0;
   }

   // Find the first Path Segment by ignoring the AS_CONFED_* segments
   for (PathSegment pathSegment : asPath.getPathSegments()) {
      if ((pathSegment.getType() == BgpConstants.Update.AsPath.AS_SET) ||
         (pathSegment.getType() == BgpConstants.Update.AsPath.AS_SEQUENCE)) {
         firstPathSegment = pathSegment;
         break;
      }
   }
   if (firstPathSegment == null) {
      // NOTE: Shouldn't happen - should be captured by isLocalRoute()
      return BgpConstants.BGP_AS_0;
   }

   if (firstPathSegment.getSegmentAsNumbers().isEmpty()) {
      // NOTE: Shouldn't happen. Should check during the parsing.
      return BgpConstants.BGP_AS_0;
   }
   return firstPathSegment.getSegmentAsNumbers().get(0);
}

/**
 * Tests whether the AS Path contains a loop.
 * <p>
 * The test is done by comparing whether the AS Path contains the
 * local AS number.
 * </p>
 *
 * @param localAsNumber the local AS number to compare against
 * @return true if the AS Path contains a loop, otherwise false
 */
boolean hasAsPathLoop(long localAsNumber) {
   for (PathSegment pathSegment : asPath.getPathSegments()) {
      for (Long asNumber : pathSegment.getSegmentAsNumbers()) {
         if (asNumber.equals(localAsNumber)) {
            return true;
         }
      }
   }
   return false;
}

/**
```

```
 * Compares this BGP route against another BGP route by using the
 * BGP Decision Process.
 * <p>
 * NOTE: The comparison needs to be performed only on routes that have
 * same IP Prefix.
 * </p>
 *
 * @param other the BGP route to compare against
 * @return true if this BGP route is better than the other BGP route
 * or same, otherwise false
 */
boolean isBetterThan(BgpRouteEntry other) {
    if (this == other) {
        return true;        // Return true if same route
    }

    // Compare the LOCAL_PREF values: larger is better
    if (getLocalPref() != other.getLocalPref()) {
        return (getLocalPref() > other.getLocalPref());
    }

    // Compare the AS number in the path: smaller is better
    if (getAsPath().getAsPathLength() !=
        other.getAsPath().getAsPathLength()) {
        return getAsPath().getAsPathLength() <
            other.getAsPath().getAsPathLength();
    }

    // Compare the Origin number: lower is better
    if (getOrigin() != other.getOrigin()) {
        return (getOrigin() < other.getOrigin());
    }

    // Compare the MED if the neighbor AS is same: larger is better
    medLabel: {
        if (isLocalRoute() || other.isLocalRoute()) {
            // Compare MEDs for non-local routes only
            break medLabel;
        }
        long thisNeighborAs = getNeighborAs();
        if (thisNeighborAs != other.getNeighborAs()) {
            break medLabel;             // AS number is different
        }
        if (thisNeighborAs == BgpConstants.BGP_AS_0) {
            break medLabel;             // Invalid AS number
        }

        // Compare the MED
        if (getMultiExitDisc() != other.getMultiExitDisc()) {
            return (getMultiExitDisc() > other.getMultiExitDisc());
        }
    }

    // Compare the peer BGP ID: lower is better
    Ip4Address peerBgpId = getBgpSession().remoteInfo().bgpId();
    Ip4Address otherPeerBgpId = other.getBgpSession().remoteInfo().bgpId();
    if (!peerBgpId.equals(otherPeerBgpId)) {
        return (peerBgpId.compareTo(otherPeerBgpId) < 0);
    }
```

```
    // Compare the peer BGP address: lower is better
    Ip4Address peerAddress = getBgpSession().remoteInfo().ip4Address();
    Ip4Address otherPeerAddress =
        other.getBgpSession().remoteInfo().ip4Address();
    if (!peerAddress.equals(otherPeerAddress)) {
        return (peerAddress.compareTo(otherPeerAddress) < 0);
    }

    return true;            // Routes are same. Shouldn't happen?
}


/**
 * A class to represent a Single Community Attribute.
 */
public static class Community {
    private final Pair<Long, Long> community;

    /**
     * Constructor.
     *
     * @param community the pair of community attribute (ASN and Local ID)
     */

    Community(Pair<Long, Long> community) {
        this.community = community;
    }

    /**
     * Gets the community attribute.
     *
     * @return the community attribute
     */
    public Pair<Long, Long> getCommunity() {
        return community;
    }

    @Override
    public String toString() {
        return MoreObjects.toStringHelper(getClass())
                .add("CommunityAttribute", this.community)
                .toString();
    }
}

/**
 * A class to represent a List of Community Attributes.
 */
public static class Communities {
    private final ArrayList<Community> communities;

    /**
     * Constructor.
     *
     * @param communities the list of community attribute pairs
     */

    Communities(ArrayList<Community> communities) {
        this.communities = communities;
    }
```

```
    /**
     * Gets the list of community attributes.
     *
     * @return the list of community attribute
     */
    public ArrayList<Community> getCommunities() {
        return communities;
    }

    @Override
    public String toString() {
        return MoreObjects.toStringHelper(getClass())
                .add("CommunityList", this.communities)
                .toString();
    }
}

/**
 * A class to represent a Single Extended Community Attribute.
 */
public static class extCommunity {
    private final Triple<Long,Long,Long> extCommunity;

    /**
     * Constructor.
     *
     * @param extCommunity the pair of community attribute (ASN and Local ID)
     */

    extCommunity(Triple<Long,Long,Long> extCommunity) {
        this.extCommunity = extCommunity;
    }

    /**
     * Gets the extended community attribute.
     *
     * @return the extended community attribute
     */
    public Triple<Long,Long,Long> getExtendedCommunity() {
        return extCommunity;
    }

    @Override
    public String toString() {
        return MoreObjects.toStringHelper(getClass())
                .add("ExtendedCommunityAttribute", this.extCommunity)
                .toString();
    }
}

/**
 * A class to represent a List of Extended Community Attributes.
 */
public static class extCommunities {
    private final ArrayList<extCommunity> extCommunities;

    /**
     * Constructor.
     *
```

```
     * @param extCommunities the list of extended community attribute pairs
     */

    extCommunities(ArrayList<extCommunity> extCommunities) {
        this.extCommunities = extCommunities;
    }

    /**
     * Gets the list of extended community attributes.
     *
     * @return the list of extended community attribute
     */
    public ArrayList<extCommunity> getExtendedCommunities() {
        return extCommunities;
    }

    @Override
    public String toString() {
        return MoreObjects.toStringHelper(getClass())
                .add("ExtendedCommunityList", this.extCommunities)
                .toString();
    }
}



/**
 * A class to represent AS Path Segment.
 */
public static class PathSegment {
    // Segment type: AS_SET(1), AS_SEQUENCE(2), AS_CONFED_SEQUENCE(3),
    // AS_CONFED_SET(4)
    private final byte type;
    private final ArrayList<Long> segmentAsNumbers;   // Segment AS numbers

    /**
     * Constructor.
     *
     * @param type the Path Segment Type: AS_SET(1), AS_SEQUENCE(2),
     * AS_CONFED_SEQUENCE(3), AS_CONFED_SET(4)
     * @param segmentAsNumbers the Segment AS numbers
     */
    PathSegment(byte type, ArrayList<Long> segmentAsNumbers) {
        this.type = type;
        this.segmentAsNumbers = checkNotNull(segmentAsNumbers);
    }

    /**
     * Gets the Path Segment Type: AS_SET(1), AS_SEQUENCE(2),
     * AS_CONFED_SEQUENCE(3), AS_CONFED_SET(4).
     *
     * @return the Path Segment Type: AS_SET(1), AS_SEQUENCE(2),
     * AS_CONFED_SEQUENCE(3), AS_CONFED_SET(4)
     */
    public byte getType() {
        return type;
    }

    /**
     * Gets the Path Segment AS Numbers.
```

```
     *
     * @return the Path Segment AS Numbers
     */
    public ArrayList<Long> getSegmentAsNumbers() {
        return segmentAsNumbers;
    }

    @Override
    public boolean equals(Object other) {
        if (this == other) {
            return true;
        }

        if (!(other instanceof PathSegment)) {
            return false;
        }

        PathSegment otherPathSegment = (PathSegment) other;
        return Objects.equals(this.type, otherPathSegment.type) &&
            Objects.equals(this.segmentAsNumbers,
                           otherPathSegment.segmentAsNumbers);
    }

    @Override
    public int hashCode() {
        return Objects.hash(type, segmentAsNumbers);
    }

    @Override
    public String toString() {
        return MoreObjects.toStringHelper(getClass())
            .add("type", BgpConstants.Update.AsPath.typeToString(type))
            .add("segmentAsNumbers", this.segmentAsNumbers)
            .toString();
    }
}

/**
 * A class to represent AS Path.
 */
public static class AsPath {
    private final ArrayList<PathSegment> pathSegments;
    private final int asPathLength;         // Precomputed AS Path Length

    /**
     * Constructor.
     *
     * @param pathSegments the Path Segments of the Path
     */
     AsPath(ArrayList<PathSegment> pathSegments) {
         this.pathSegments = checkNotNull(pathSegments);

         //
         // Precompute the AS Path Length:
         // - AS_SET counts as 1
         // - AS_SEQUENCE counts how many AS numbers are included
         // - AS_CONFED_SEQUENCE and AS_CONFED_SET are ignored
         //
         int pl = 0;
         for (PathSegment pathSegment : pathSegments) {
```

```
        switch (pathSegment.getType()) {
        case BgpConstants.Update.AsPath.AS_SET:
            pl++;           // AS_SET counts as 1
            break;
        case BgpConstants.Update.AsPath.AS_SEQUENCE:
            // Count each AS number
            pl += pathSegment.getSegmentAsNumbers().size();
            break;
        case BgpConstants.Update.AsPath.AS_CONFED_SEQUENCE:
            break;          // Ignore
        case BgpConstants.Update.AsPath.AS_CONFED_SET:
            break;          // Ignore
        default:
            // NOTE: What to do if the Path Segment type is unknown?
            break;
        }
    }
    asPathLength = pl;
}

/**
 * Gets the AS Path Segments.
 *
 * @return the AS Path Segments
 */
public ArrayList<PathSegment> getPathSegments() {
    return pathSegments;
}

/**
 * Gets the AS Path Length as considered by the BGP Decision Process.
 *
 * @return the AS Path Length as considered by the BGP Decision Process
 */
int getAsPathLength() {
    return asPathLength;
}

@Override
public boolean equals(Object other) {
    if (this == other) {
        return true;
    }

    if (!(other instanceof AsPath)) {
        return false;
    }

    AsPath otherAsPath = (AsPath) other;
    return Objects.equals(this.pathSegments, otherAsPath.pathSegments);
}

@Override
public int hashCode() {
    return pathSegments.hashCode();
}

@Override
public String toString() {
    return MoreObjects.toStringHelper(getClass())
```

```
            .add("pathSegments", this.pathSegments)
            .toString();
    }
}

/**
 * Compares whether two objects are equal.
 * <p>
 * NOTE: The bgpSession field is excluded from the comparison.
 * </p>
 *
 * @return true if the two objects are equal, otherwise false.
 */
@Override
public boolean equals(Object other) {
    if (this == other) {
        return true;
    }

    //
    // NOTE: Subclasses are considered as change of identity, hence
    // equals() will return false if the class type doesn't match.
    //
    if (other == null || getClass() != other.getClass()) {
        return false;
    }

    if (!super.equals(other)) {
        return false;
    }

    // NOTE: The bgpSession field is excluded from the comparison
    BgpRouteEntry otherRoute = (BgpRouteEntry) other;
    return (this.origin == otherRoute.origin) &&
        Objects.equals(this.asPath, otherRoute.asPath) &&
        (this.localPref == otherRoute.localPref) &&
        (this.multiExitDisc == otherRoute.multiExitDisc);
}

/**
 * Computes the hash code.
 * <p>
 * NOTE: We return the base class hash code to avoid expensive computation
 * </p>
 *
 * @return the object hash code
 */
@Override
public int hashCode() {
    return super.hashCode();
}

@Override
public String toString() {
    return MoreObjects.toStringHelper(getClass())
        .add("prefix", prefix())
        .add("nextHop", nextHop())
        .add("bgpId", bgpSession.remoteInfo().bgpId())
        .add("origin", BgpConstants.Update.Origin.typeToString(origin))
        .add("asPath", asPath)
```

```
            .add("localPref", localPref)
            .add("multiExitDisc", multiExitDisc)
            .add("community", communities)
            .add("extCommunity", extCommunities)
            .toString();
    }
}
```

Appendix 4 – BgpRoutesListCommand Module

BgpRoutesListCommand.java

```
/*
 * Copyright 2017-present Open Networking Foundation
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.onosproject.routing.cli;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ArrayNode;
import com.fasterxml.jackson.databind.node.ObjectNode;
import org.apache.karaf.shell.commands.Command;
import org.apache.karaf.shell.commands.Option;
import org.onosproject.cli.AbstractShellCommand;
import org.onosproject.routing.bgp.BgpConstants;
import org.onosproject.routing.bgp.BgpInfoService;
import org.onosproject.routing.bgp.BgpRouteEntry;
import org.onosproject.routing.bgp.BgpSession;

import java.util.ArrayList;
import java.util.Collection;

/**
 * Command to show the routes learned through BGP.
 */
@Command(scope = "onos", name = "bgp-routes",
        description = "Lists all BGP best routes")
public class BgpRoutesListCommand extends AbstractShellCommand {
    @Option(name = "-s", aliases = "--summary",
            description = "BGP routes summary",
            required = false, multiValued = false)
    private boolean routesSummary = false;

    @Option(name = "-n", aliases = "--neighbor",
            description = "Routes from a BGP neighbor",
            required = false, multiValued = false)
    private String bgpNeighbor;

    @Option(name = "-c", aliases = "--show-community",
            description = "Community attribute from BGP routes",
            required = false, multiValued = false)
    private boolean showCommunity;
```

```
private static final String FORMAT_SUMMARY_V4 =
   "Total BGP IPv4 routes = %d";
private static final String FORMAT_SUMMARY_V6 =
   "Total BGP IPv6 routes = %d";
private static final String FORMAT_HEADER =
   "   Network           Next Hop       Origin LocalPref      MED BGP-ID    ";
private static final String FORMAT_ROUTE_LINE1 =
   "   %-18s %-15s %6s %9s %9s %-15s   ";
private static final String FORMAT_ROUTE_LINE2 =
   "                    AsPath %s";

@Override
protected void execute() {
   BgpInfoService service = AbstractShellCommand.get(BgpInfoService.class);

   // Print summary of the routes
   if (routesSummary) {
      printSummary(service.getBgpRoutes4(), service.getBgpRoutes6());
      return;
   }

   if (showCommunity) {
      printCommunity(service.getBgpRoutes4(), service.getBgpRoutes6());
      return;
   }

   BgpSession foundBgpSession = null;
   if (bgpNeighbor != null) {
      // Print the routes from a single neighbor (if found)
      for (BgpSession bgpSession : service.getBgpSessions()) {
         if (bgpSession.remoteInfo().bgpId().toString().equals(bgpNeighbor)) {
            foundBgpSession = bgpSession;
            break;
         }
      }
      if (foundBgpSession == null) {
         print("BGP neighbor %s not found", bgpNeighbor);
         return;
      }
   }

   // Print the routes
   if (foundBgpSession != null) {
      printRoutes(foundBgpSession.getBgpRibIn4(),
                  foundBgpSession.getBgpRibIn6());
   } else {
      printRoutes(service.getBgpRoutes4(), service.getBgpRoutes6());
   }
}

/**
 * Prints summary of the routes.
 *
 * @param routes4 the IPv4 routes
 * @param routes6 the IPv6 routes
 */
private void printSummary(Collection<BgpRouteEntry> routes4,
                          Collection<BgpRouteEntry> routes6) {
   if (outputJson()) {
      ObjectMapper mapper = new ObjectMapper();
```

```
        ObjectNode result = mapper.createObjectNode();
        result.put("totalRoutes4", routes4.size());
        result.put("totalRoutes6", routes6.size());
        print("%s", result);
    } else {
        print(FORMAT_SUMMARY_V4, routes4.size());
        print(FORMAT_SUMMARY_V6, routes6.size());
    }
}

/**
 * Prints all routes.
 *
 * @param routes4 the IPv4 routes to print
 * @param routes6 the IPv6 routes to print
 */
private void printRoutes(Collection<BgpRouteEntry> routes4,
                         Collection<BgpRouteEntry> routes6) {
    if (outputJson()) {
        ObjectMapper mapper = new ObjectMapper();
        ObjectNode result = mapper.createObjectNode();
        result.set("routes4", json(routes4));
        result.set("routes6", json(routes6));
        print("%s", result);
    } else {
        // The IPv4 routes
        print(FORMAT_HEADER);
        for (BgpRouteEntry route : routes4) {
            printRoute(route);
        }
        print(FORMAT_SUMMARY_V4, routes4.size());
        print("");                // Empty separator line
        // The IPv6 routes
        print(FORMAT_HEADER);
        for (BgpRouteEntry route : routes6) {
            printRoute(route);
        }
        print(FORMAT_SUMMARY_V6, routes6.size());
    }
}

/**
 * Prints a BGP route.
 *
 * @param route the route to print
 */
private void printRoute(BgpRouteEntry route) {
    if (route != null) {
        print(FORMAT_ROUTE_LINE1, route.prefix(), route.nextHop(),
              BgpConstants.Update.Origin.typeToString(route.getOrigin()),
              route.getLocalPref(), route.getMultiExitDisc(),
              route.getBgpSession().remoteInfo().bgpId());
        print(FORMAT_ROUTE_LINE2, asPath4Cli(route.getAsPath()));
    }
}

/**
 * Prints routes BGP community.
 *
 * @param routes4 the IPv4 routes to print
```

```
 * @param routes6 the IPv4 routes to print
 */
private void printCommunity(Collection<BgpRouteEntry> routes4,
                    Collection<BgpRouteEntry> routes6) {

   // The IPv4 routes
   print(" IPv4 Network     BGP Community          BGP Extended Community");

   for (BgpRouteEntry route : routes4) {

        if (route.getCommunities() == null) {
           print("%-18s    null    null",
                route.prefix().toString());
        } else {
           print("%-18s    %-32s   %-32s",
                route.prefix().toString(),
                communityCli(route.getCommunities()),
                extCommunityCli(route.getExtendedCommunities()));
        }
   }

   print ("");               // Empty separator line

   // The IPv6 routes
   print(" IPv6 Network     BGP Community          BGP Extended Community");

   for (BgpRouteEntry route : routes6) {

      if (route.getCommunities() == null) {
         print("%-18s    null    null",
              route.prefix().toString());
      } else {
         print("%-18s    %-32s   %-32s",
              route.prefix().toString(),
               communityCli(route.getCommunities()),
               extCommunityCli(route.getExtendedCommunities()));
      }
   }
}


/**
 * Formats the AS Path as a string that can be shown on the CLI.
 *
 * @param asPath the AS Path to format
 * @return the AS Path as a string
 */
private String asPath4Cli(BgpRouteEntry.AsPath asPath) {
   ArrayList<BgpRouteEntry.PathSegment> pathSegments =
      asPath.getPathSegments();

   if (pathSegments.isEmpty()) {
      return "[none]";
   }

   final StringBuilder builder = new StringBuilder();
   for (BgpRouteEntry.PathSegment pathSegment : pathSegments) {
      String prefix = null;
      String suffix = null;
      switch (pathSegment.getType()) {
```

```
        case BgpConstants.Update.AsPath.AS_SET:
            prefix = "[AS-Set";
            suffix = "]";
            break;
        case BgpConstants.Update.AsPath.AS_SEQUENCE:
            break;
        case BgpConstants.Update.AsPath.AS_CONFED_SEQUENCE:
            prefix = "[AS-Confed-Seq";
            suffix = "]";
            break;
        case BgpConstants.Update.AsPath.AS_CONFED_SET:
            prefix = "[AS-Confed-Set";
            suffix = "]";
            break;
        default:
            builder.append(String.format("(type = %s)",
                    BgpConstants.Update.AsPath.typeToString(pathSegment.getType())));
            break;
        }

        if (prefix != null) {
            if (builder.length() > 0) {
                builder.append(" ");        // Separator
            }
            builder.append(prefix);
        }
        // Print the AS numbers
        for (Long asn : pathSegment.getSegmentAsNumbers()) {
            if (builder.length() > 0) {
                builder.append(" ");        // Separator
            }
            builder.append(String.format("%d", asn));
        }
        if (suffix != null) {
            // No need for separator
            builder.append(prefix);
        }
    }
    return builder.toString();
}

/**
 * Formats the BGP Communities as a string that can be shown on the CLI.
 *
 * @param communities the community attributes to format
 * @return the community attributes as a string
 */
private String communityCli(BgpRouteEntry.Communities communities) {

    ArrayList<BgpRouteEntry.Community> communityList = communities.getCommunities();

    final StringBuilder builder = new StringBuilder();

    for (BgpRouteEntry.Community community : communityList) {

        builder.append(community.getCommunity().getLeft().toString() + ":"
                + community.getCommunity().getRight().toString() +" ");

    }
```

```
    return builder.toString();

}

/**
 * Formats the BGP Extended Communities as a string that can be shown on the CLI.
 *
 * @param extCommunities the extcommunity attributes to format
 * @return the extcommunity attributes as a string
 */
private String extCommunityCli(BgpRouteEntry.extCommunities extCommunities) {

    log.info(extCommunities.getExtendedCommunities().toString());

    ArrayList<BgpRouteEntry.extCommunity> extCommunityList =
          extCommunities.getExtendedCommunities();

    final StringBuilder builder = new StringBuilder();

    for (BgpRouteEntry.extCommunity extCommunity : extCommunityList) {

       builder.append(extCommunity.getExtendedCommunity().getLeft().toString() + ":"
             + extCommunity.getExtendedCommunity().getMiddle().toString() + ":"
             + extCommunity.getExtendedCommunity().getRight().toString() + " ");
    }

    return builder.toString();

}


/**
 * Produces a JSON array of routes.
 *
 * @param routes the routes with the data
 * @return JSON array with the routes
 */
private JsonNode json(Collection<BgpRouteEntry> routes) {
    ObjectMapper mapper = new ObjectMapper();
    ArrayNode result = mapper.createArrayNode();

    for (BgpRouteEntry route : routes) {
       result.add(json(mapper, route));
    }
    return result;
}

/**
 * Produces JSON object for a route.
 *
 * @param mapper the JSON object mapper to use
 * @param route the route with the data
 * @return JSON object for the route
 */
private ObjectNode json(ObjectMapper mapper, BgpRouteEntry route) {
    ObjectNode result = mapper.createObjectNode();

    result.put("prefix", route.prefix().toString());
    result.put("nextHop", route.nextHop().toString());
    result.put("bgpId",
```

```
                route.getBgpSession().remoteInfo().bgpId().toString());
        result.put("origin", BgpConstants.Update.Origin.typeToString(route.getOrigin()));
        result.set("asPath", json(mapper, route.getAsPath()));
        result.put("localPref", route.getLocalPref());
        result.put("multiExitDisc", route.getMultiExitDisc());

        return result;
    }

    /**
     * Produces JSON object for an AS path.
     *
     * @param mapper the JSON object mapper to use
     * @param asPath the AS path with the data
     * @return JSON object for the AS path
     */
    private ObjectNode json(ObjectMapper mapper, BgpRouteEntry.AsPath asPath) {
        ObjectNode result = mapper.createObjectNode();
        ArrayNode pathSegmentsJson = mapper.createArrayNode();
        for (BgpRouteEntry.PathSegment pathSegment : asPath.getPathSegments()) {
            ObjectNode pathSegmentJson = mapper.createObjectNode();
            pathSegmentJson.put("type",
                                BgpConstants.Update.AsPath.typeToString(pathSegment.getType()));
            ArrayNode segmentAsNumbersJson = mapper.createArrayNode();
            for (Long asNumber : pathSegment.getSegmentAsNumbers()) {
                segmentAsNumbersJson.add(asNumber);
            }
            pathSegmentJson.set("segmentAsNumbers", segmentAsNumbersJson);
            pathSegmentsJson.add(pathSegmentJson);
        }
        result.set("pathSegments", pathSegmentsJson);

        return result;
    }
}
```

Appendix 5 – Indopronos Module

Indopronos.java

```
/*
 * Copyright 2018-Franciscus Ari Wibowo
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.onosproject.indopronos;

import org.apache.felix.scr.annotations.Activate;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Deactivate;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.ReferenceCardinality;
import org.onosproject.app.ApplicationService;
import org.onosproject.core.ApplicationId;
import org.onosproject.core.CoreService;
import org.onosproject.intentsync.IntentSynchronizationService;
import org.onosproject.net.intent.IntentService;
import org.onosproject.routeservice.RouteService;
import org.onosproject.routing.bgp.BgpInfoService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Application component for Inter-Domain Provisioning for Policy Routing in ONOS (Indopronos)
 */

@Component(immediate = true)
public class Indopronos {

   public static final String INDOPRONOS_APP = "org.onosproject.indopronos";
   private final Logger log = LoggerFactory.getLogger(getClass());

   @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
   protected BgpInfoService bgpService;

   @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
   protected CoreService coreService;

   @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
   protected IntentService intentService;

   @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
   protected IntentSynchronizationService intentSyncService;
```

```
    @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
    protected ApplicationService applicationService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
    protected RouteService routeService;

    private IndopronosConnectivityManager IndopronosConnectivity;

    private ApplicationId appId;

    @Activate
    protected void activate() {

       appId = coreService.registerApplication(INDOPRONOS_APP);
       IndopronosConnectivity = new IndopronosConnectivityManager(appId,
                                        bgpService,
                                        coreService,
                                        intentService,
                                        intentSyncService,
                                        routeService);
       IndopronosConnectivity.start();

       applicationService.registerDeactivateHook(appId,
                    () -> intentSyncService.removeIntentsByAppId(appId));

       log.info("Started");
    }

    @Deactivate
    protected void deactivate() {
       IndopronosConnectivity.stop();
       log.info("Stopped");
    }

}
```

Appendix 6 – IndopronosConnectivityManager Module

IndopronosConnectivityManager.java

```
/*
 * Copyright 2018-Franciscus Ari Wibowo
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.onosproject.indopronos;

import org.onlab.packet.MacAddress;
import org.onosproject.core.ApplicationId;
import org.onosproject.core.CoreService;
import org.onosproject.intentsync.IntentSynchronizationService;
import org.onosproject.net.ConnectPoint;
import org.onosproject.net.DeviceId;
import org.onosproject.net.FilteredConnectPoint;
import org.onosproject.net.PortNumber;
import org.onosproject.net.flow.DefaultTrafficSelector;
import org.onosproject.net.flow.DefaultTrafficTreatment;
import org.onosproject.net.flow.TrafficTreatment;
import org.onosproject.net.intent.Intent;
import org.onosproject.net.intent.IntentService;
import org.onosproject.net.intent.Key;
import org.onosproject.net.intent.MultiPointToSinglePointIntent;
import org.onosproject.net.intent.PointToPointIntent;
import org.onosproject.routeservice.ResolvedRoute;
import org.onosproject.routeservice.RouteEvent;
import org.onosproject.routeservice.RouteListener;
import org.onosproject.routeservice.RouteService;
import org.onosproject.routing.bgp.BgpInfoService;
import org.onosproject.routing.bgp.BgpRouteEntry;
import org.onosproject.routing.bgp.RouteEntry;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.ArrayList;
import java.util.Collection;

/**
 * FIB component of INDOPRONOS Application.
 */

public class IndopronosConnectivityManager {
```

```
// Condition to execute policy routing
private static final String ACTION_COMMUNITY_LOCAL_ID = "100";

// Intent Priority to override MP2SP of SDN-IP
private static final Integer PRIORITY = 250;

// Premium Link / Private Peer Definition
private static final String SWITCH_DPID = "of:0000000000000001";
private static final Integer PORT_NUMBER = 6;
private static final FilteredConnectPoint EGRESS_POINT_PRIVATE =
      new FilteredConnectPoint
            (new ConnectPoint(
                  DeviceId.deviceId(SWITCH_DPID),
                  PortNumber.portNumber(PORT_NUMBER)));
private static final MacAddress NEXT_HOP_MAC_PRIVATE =
      MacAddress.valueOf("00:00:00:00:00:A3");

private final BgpInfoService bgpService;
private final IntentService intentService;
private final CoreService coreService;
private final IntentSynchronizationService intentSyncService;
private final RouteService routeService;

private static final Logger log = LoggerFactory.getLogger(
      IndopronosConnectivityManager.class);

private final ApplicationId appId;

private final InternalRouteListener routeListener = new InternalRouteListener();

/**
 * Creates a new Indopronos ConnectivityManager.
 *
 * @param appId            the application ID
 * @param bgpService       the BGP service
 * @param coreService      the core Service
 * @param intentService    the intent service
 * @param intentSyncService the intent synchronizer service
 */
public IndopronosConnectivityManager(ApplicationId appId,
                           BgpInfoService bgpService,
                           CoreService coreService,
                           IntentService intentService,
                           IntentSynchronizationService intentSyncService,
                           RouteService routeService) {
   this.appId = appId;
   this.bgpService = bgpService;
   this.coreService = coreService;
   this.intentService = intentService;
   this.intentSyncService = intentSyncService;
   this.routeService = routeService;
}

/**
 * Starts the Indopronos connectivity manager.
 */
public void start() {
   routeService.addListener(routeListener);
```

```
    Collection<BgpRouteEntry> routes4 = bgpService.getBgpRoutes4();
    Collection<BgpRouteEntry> routes6 = bgpService.getBgpRoutes6();

    for (BgpRouteEntry route : routes4) {
        setUpConnectivity(route);
    }

    for (BgpRouteEntry route : routes6) {
        setUpConnectivity(route);
    }
}

/**
 * Stops the Indopronos connectivity manager.
 */
public void stop() {
    routeService.removeListener(routeListener);
}

/**
 * Stops the Indopronos connectivity manager.
 */
public void update(ResolvedRoute resolvedRoute) {

    log.info("Resolved route: {}", resolvedRoute.prefix());

    Iterable<Intent> intents = intentService.getIntents();

    for (Intent intent : intents) {
        if (intent.appId().equals(appId)) {
            intentSyncService.withdraw(intent);
        }
    }

    Collection<BgpRouteEntry> routes4 = bgpService.getBgpRoutes4();
    Collection<BgpRouteEntry> routes6 = bgpService.getBgpRoutes6();

    for (BgpRouteEntry route : routes4) {

        if (route.prefix().equals(resolvedRoute.prefix())) {
            setUpConnectivity(route);
        }

    }

    for (BgpRouteEntry route : routes6) {

        if (route.prefix().equals(resolvedRoute.prefix())) {
            setUpConnectivity(route);
        }
    }

}

/**
 * Sets up paths to override the connectivity between BGP router who send the
 * community and private peer BGP router.
 * @param route
 */
private void setUpConnectivity(BgpRouteEntry route) {
```

```
//Collection<BgpRouteEntry> routes4 = bgpService.getBgpRoutes4();
//Collection<BgpRouteEntry> routes6 = bgpService.getBgpRoutes6();


Iterable<Intent> intents;
intents = intentService.getIntents();


if (route.getCommunities() == null) {
   log.info("No communities found in prefix {}", route.prefix());
   return;
}


if (route.getExtendedCommunities() == null) {
   log.info("No extended communities found in prefix {}", route.prefix());
   return;
}


BgpRouteEntry.PathSegment firstPathSegment =
      route.getAsPath().getPathSegments().get(0);
long originatingAsn = firstPathSegment.getSegmentAsNumbers().get(0);
log.info("Prefix {} originated from AS {}", route.prefix(), originatingAsn);


ArrayList<BgpRouteEntry.Community> routeCommunities =
      route.getCommunities().getCommunities();
ArrayList<BgpRouteEntry.extCommunity> routeExtCommunities =
      route.getExtendedCommunities().getExtendedCommunities();


for (BgpRouteEntry.Community routeCommunity : routeCommunities) {


   Long communityAsn = routeCommunity.getCommunity().getLeft();
   Long communityLocalId = routeCommunity.getCommunity().getRight();


   for (BgpRouteEntry.extCommunity routeExtCommunity : routeExtCommunities) {


      if (String.valueOf(communityAsn).equals(String.valueOf(originatingAsn)) &&
            (String.valueOf(communityLocalId).equals(ACTION_COMMUNITY_LOCAL_ID)) &&
            (routeExtCommunity.getExtendedCommunity().getMiddle().equals(communityAsn)) &&
            (routeExtCommunity.getExtendedCommunity().getRight().equals(communityLocalId))) {


         log.info("Matched both communities ASN and Local ID for prefix {}", route.prefix());
         MultiPointToSinglePointIntent routeIntent = getRouteIntent(route, intents);


         // Build treatment: rewrite the destination MAC address
         TrafficTreatment.Builder treatment = DefaultTrafficTreatment.builder()
               .setEthDst(NEXT_HOP_MAC_PRIVATE);


         String intentKeyOne = String.valueOf(communityAsn) + ":"
```

```
                    + String.valueOf(communityLocalId) + "-One";
            PointToPointIntent newIntentOne = PointToPointIntent.builder()
                    .appId(appId)
                    .key(Key.of(intentKeyOne, appId))
                    .filteredIngressPoint(routeIntent.filteredEgressPoint())
                    .filteredEgressPoint(EGRESS_POINT_PRIVATE)
                    .selector(DefaultTrafficSelector.builder().build())
                    .treatment(treatment.build())
                    .priority(PRIORITY)
                    .build();
            log.info("Intent to be installed {}", newIntentOne);
            intentSyncService.submit(newIntentOne);


            String intentKeyTwo = String.valueOf(communityAsn) + ":"
                    + String.valueOf(communityLocalId) + "-Two";
            PointToPointIntent newIntentTwo = PointToPointIntent.builder()
                    .appId(appId)
                    .key(Key.of(intentKeyTwo, appId))
                    .filteredIngressPoint(EGRESS_POINT_PRIVATE)
                    .filteredEgressPoint(routeIntent.filteredEgressPoint())
                    .selector(DefaultTrafficSelector.builder().build())
                    .treatment(routeIntent.treatment())
                    .priority(PRIORITY)
                    .build();
            log.info("Intent to be installed {}", newIntentTwo);
            intentSyncService.submit(newIntentTwo);
        } else {
            log.info("Both communities ASN and Local ID in prefix {} are not matched",
                    route.prefix());
        }
      }
    }


}
/**
 * Get route multi-point-to-single-point intent from SDN IP.
 *
 * @param intents the community attribute to format
 * @return the Community as a string with colon
 */

private MultiPointToSinglePointIntent getRouteIntent(BgpRouteEntry route,
                                     Iterable<Intent> intents) {

    MultiPointToSinglePointIntent routeIntent = null;

    for (Intent intent : intents) {

       if (intent instanceof MultiPointToSinglePointIntent) {

          if (route.prefix().toString().equals(intent.key().toString())) {
             routeIntent = (MultiPointToSinglePointIntent) intent;
          }
       }
    }
    return routeIntent;
}
```

```
    private class InternalRouteListener implements RouteListener {
        @Override
        public void event(RouteEvent event) {
            switch (event.type()) {
                case ROUTE_ADDED:
                    update(event.subject());
                    break;
                case ROUTE_UPDATED:
                    update(event.subject());
                    break;
                case ROUTE_REMOVED:
                    update(event.subject());
                    break;
                default:
                    break;
            }
        }
    }
}
```