Software-Defined Networking Application for Inter-domain Routing in Transit ISPs

by

Trung Truong

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2020

Abstract

Today, the Internet plays an vital part in our society. We rely greatly on the Internet to work, to communicate and to entertain. The Internet is a very large and complex computer network, consisting of tens of thousands of networks called autonomous systems (ASes). The key routing protocol for interdomain routing between ASes, Border Gateway Protocol (BGP), was invented three decades ago. Although BGP has undergone many improvements, many fundamental problems and limitations of BGP still exist today. For instance, BGP does not have the resiliency to attacks and good support for traffic engineering. To date, many evolutionary and revolutionary solutions have been proposed to address these problems. However, very few were adopted. While there are many reasons for this limited adoption, one may blame for the lack of deployability, scalability and more importantly adequate functionality.

SDN is a new networking paradigm that decouples the control plane and data plane. SDN breaks the ossification of the Internet and enables network innovations. SDN has been thoroughly investigated for the enterprise environment. This research investigates the application of SDN for interdomain routing in transit ASes. Specifically, the main goal is to study how SDN capabilities can be utilised to develop a scalable, programmable and flexible routing architecture for transit ISPs.

Acknowledgement

This has been the most challenging journey I have ever had. The thesis would not have been successfully completed without tremendous support from my two supervisors A/Prof Ian Welch and Dr. Bryan Ng. I would like to express deepest appreciation to A/Prof Ian Welch who was always available to give me valuable advice and to discuss. His contributions to the methodology and case study were valuable. Special thanks to Dr. Bryan for his support, particularly with the mathematical analysis and overall structure of the thesis.

I would also like to extend my gratitude to Google and Josh Bailey who was Google representative at the time. The research was made possible by their generous sponsorship. The opportunity to take part in Faucet project with Josh also gained me great technical and industrial experiences that are useful for the research. Thanks also to Dr. Diana Siwiak for her advice and support in dealing with various issues with the thesis.

Last but not least, special thanks to my lovely and diligent wife. Her encouragement, support and delicious meals enabled me to go to the end of this long journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet is composed of tens of thousands of autonomous systems. Each **Autonomous System (AS)** is independently operated. A small number of ASes forms the "Internet core". They are Internet Service Providers (ISPs) that provide **transit services** to other ASes. The majority of ASes rely solely on the transit services to get connected to the Internet. Typically, an end-to-end Internet path traverses at least one or more transit ASes.

Currently, **BGP** is the de facto protocol for interdomain routing. BGP facilitates the establishment of end-to-end connectivity between ASes in the Internet. Unfortunately, it was designed with limited support for quality of service and security [2]. Moreover, it converges slowly and can take several minutes to compute a new path after a failure [3]. As a result, the computed interdomain paths are ill-suited for many emerging applications such as real-time control and communications.

In the last decade, **Software-Defined Networking (SDN)** has emerged as a new networking paradigm advocating for the decouple of the control and forwarding function in the network. SDN is composed of a logically centralised controller and a network of forwarding devices. The controller makes decisions based on a complete view of the network. SDN is a promising approach to many networking problems [4]. OpenFlow is a prominent SDN framework which is widely supported by both the academia and industry.

This thesis presents an **incremental and backward-compatible SDN-based solution** that can be deployed per transit provider. In contrast to clean-slate solutions, this would allow a transit ISP to improve its transit service without cooperation from the others. This potentially sparks a competition and leads to a large scale adoption of SDN among ISPs. Having enough number of SDN-enabled ISPs would enable the deployment of a clean-slate solution which is the ultimate goal among the Internet community.

1.1 Motivation

SDN has been thoroughly studied in the literature to develop solutions to many existing networking problems [5]. Interestingly, solutions proposed so far mainly focus on security or traffic engineering problems in enterprise and datacenter networks. With regard to transit ISPs, previous research has looked at interdomain routing [6, 7, 8]. However, their main focus is an Internet-wide solution.

Other works such as SDN-IP [9] and RouteFlow [10] offer no enhancement to the functionality and operation of interdomain routing, as they merely investigate how SDN can be incrementally deployed in an ISP environment. In particular, work on software-defined internet exchange (SDX) [11] which improves interdomain routing within an Internet Exchange Point (IXP) has inspired this research. However, SDX cannot be directly applied to an ISP environment because it was designed for work on a single-switch network. This research aims to utilise SDN and develop a working solution for transit ISPs.

Transit ISPs play a crucial role in the Internet infrastructure. They sit at the backbone of the Internet, have connectivity to every destination in the Internet and carry a large portion of interdomain traffic between end (stub) networks [12]. Since the commercialisation of the Internet, transit services remain as “best-effort” delivery: they merely provide the connectivity with no QoS guarantee. In practice, transit services are often associated with a service level agreement (SLA). However, the scope of SLAs is limited to a network’s boundary as a transit operator has no mechanism to ensure end-to-end SLAs. Meanwhile, network applications have become more diverse, ranging from high-bandwidth (e.g. data-intensive applications, on-demand videos) to low-latency ones (e.g. real-time communications). The best-effort transit service may work for some applications but not the others.

This research develops a SDN-based solution for interdomain routing in transit ISPs called SDN for Interdomain Routing (SDIRO). Figure 1.1 depicts the high-level architecture of the system in which a logically centralised controller with complete visibility of the path performance and availability status, computes routes for all border routers in the network. The controller enables the operator to programmatically define fine-grained routing policy for customers. For instance, traffic from customer A, B and C to the same destination can be routed over different paths in order to meet the customers’ requirements. While this architecture can possibly be implemented using existing technologies such as Virtual Routing and Forwarding (VRF) and Policy-Based Routing (PBR) as suggested by past research [13, 14],

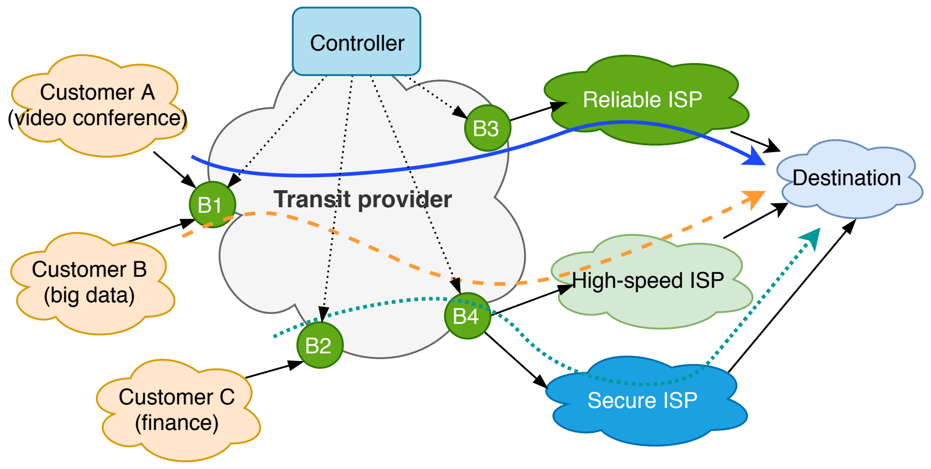


Figure 1.1: High-level architecture of the solution which consists of a logically centralised controller. Each border router (B1..,B4) can use a different path to a remote destination.

there are a number of technical challenges that need to be addressed for the sake of practicality (i.e. simplicity, efficiency and scalability). The following describes these challenges.

Routing programmability. Currently, high-level routing objectives (e.g. "use of the cheapest path or low latency one") are manually translated into BGP commands (e.g. to set the BGP local-preference) which tell a router to prefer one particular path over the others for traffic forwarding [15]. However, conventional routers support a limited set of commands (associated with the BGP path attributes). Thus, many policies that are dynamic, temporal or performance-aware cannot be directly implemented in BGP. Instead of writing policy with a set of predefined commands, the operator should be able to express policy as software applications. This *software-defined policy* allows the operator to realise arbitrary policy. Existing SDN programming frameworks such as Procera [16] and Maple [17] designed for enterprise networks do not offer the right programming language and abstractions for interdomain routing. Solutions built for conventional networks [18, 14] are rather complex to implement and do not benefit from the advantages of SDN. This research fills this gap by designing and implementing a prototype of a logically centralised controller for interdomain routing in transit ISPs. The controller leverages graph models and a graph database to provide the operator with a declarative language for writing arbitrary policy in a simple manner while ensuring scalability (see Chapter 3).

Routing flexibility. BGP routers route packets based on the destination. This greatly limits the operator from realising fine-grained routing (i.e. customer-specific or application-specific). For example, a router can have multiple attached paths but the destination-based forwarding limits it from forwarding different traffic via different path. This limitation can be overcome by using PBR and VRF [13, 14]. However, using these technologies for realizing fine-grained policy is overkill and too complex to manage (see Chapter 6). In this research, these issues are addressed by re-designing the ISP border routers using SDN and OpenFlow. It uses the same concept of RouteFlow [10] and SDN-IP [9] but incorporates with the multi-path routing capability to support flexible policy (see Chapter 4).

Routing scalability. The flexibility of being able to define finer-grained policy increases the forwarding state in routers. This exacerbates the already-a-concern scalability with regard to exponential growth of the forwarding table in transit ISPs [19]. Past research has attempted to address this scalability issue using many different approaches such as routing table compression [20], clean-slate design [21] and coordinated route aggregation [22]. This research takes a different approach which decomposes the forwarding table into sub-tables and distributes across a interconnected network of switches (see Chapter 5). This allows the forwarding table to scale vertically or horizontally by replacing or adding more switches to the network.

1.2 Research Framework

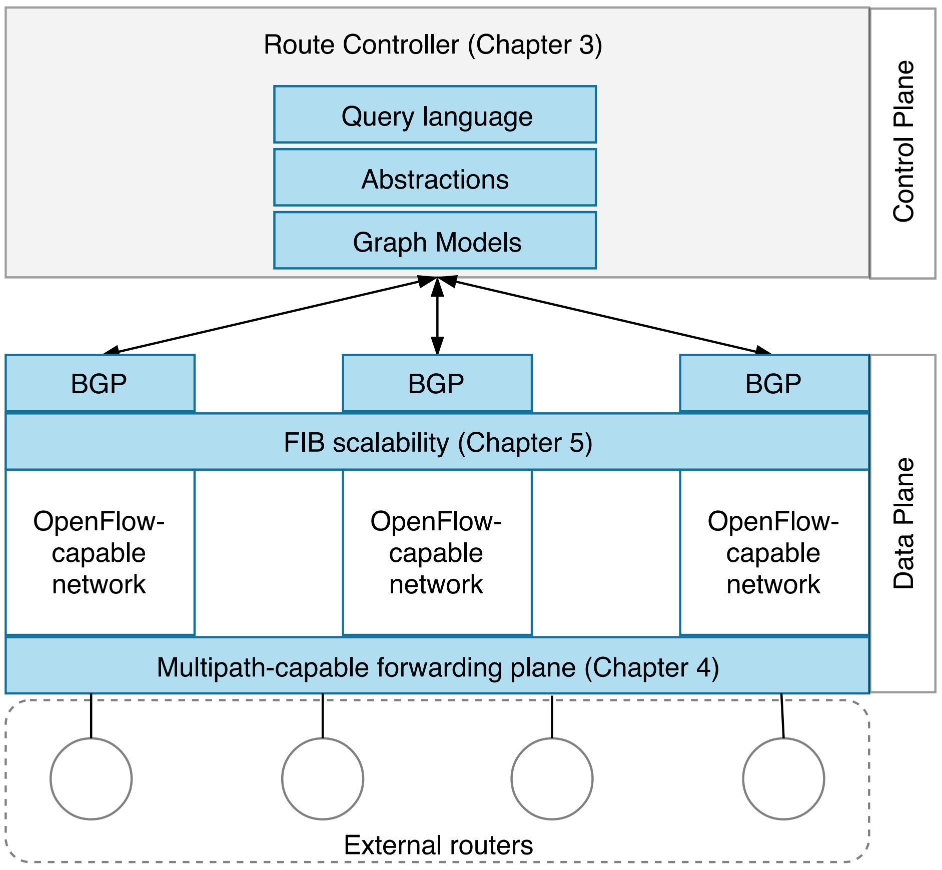


Figure 1.2: Research Framework

The main focus in the research is the applicability of SDN and OpenFlow into interdomain routing in transit ISPs. The results are solutions which address the key routing challenges. The research framework is depicted in

Figure 1.2. It describes components in the control and data plane being covered in and the contributions of the research. On top is the controller which provides routing abstractions and a query-like Application Programming Interface (API) to the applications. The data plane is made up of a distributed network of BGP routers built on top OpenFlow. It also includes a mechanism and algorithm for scaling the Forwarding Information Base (FIB) table and a design and an implementation for the multipath forwarding capability.

1.3 Contributions

1. **Design and evaluation of a centralised controller for programmable interdomain routing in transit ISPs.**
   In a transit AS, routers are autonomous which select the best routes based on a statically configured policy. The route selection of the conventional BGP does not consider metrics such as performance or security. A router may continue to send traffic via path which is suffering from congestion but it is not deemed broken in the control plane. Moreover, policy expression using low-level, vendor-based languages and an indirect and overloaded mechanism (i.e. route ranking using local-preferences) is difficult to write and to reason about.

   a) A design and implementation of a centralised controller for policy programmability through a graph-based representation of routing and a declarative programming language is developed. A working implementation of the controller has been developed using Neo4J graph database.

   b) Extensive evaluation of the controller using synthesis and real data traces was performed. The evaluation looks at a number of metrics including convergence and policy expressiveness.

2. **Design and evaluation of BGP multipath routing on OpenFlow-enabled networks.**
   Multipath routing enables the network to utilise all available paths for traffic delivery, as opposed to the current single-path model of BGP which forces traffic from multiple customers to be sent via the same path.

   a) The research presents a design for multipath capability with minimal rule overhead compared to the conventional techniques. By leveraging OpenFlow and its multi-table capability, a multipath-capable BGP router, built on top of a distributed network of switches, is designed and implemented.

   b) A fully functional (open-source) OpenFlow-based BGP router (controller) was implemented in Python which has become a popular language for network programming [23]. The main logic (BGP controller) is implemented as a Ryu application which communicates with Faucet being responsible for managing a distributed layer 2 network. Managing traffic within the layer 2 network is the job of Faucet, whereas the BGP controller performs the standard BGP operation and react to the route controller's commands.

   c) A physical testbed was deployed simulating a transit ISP with three demarcation points (i.e. Point of Presence (POP)) using both hardware and software switches. This demonstrates that the system can be deployed incrementally as one of the POPs is using traditional routers. Evaluation on performance of both the control plane (processing capacity, convergence) and data plane (forwarding performance) was performed on this testbed.

3. **Design and evaluation of a FIB distribution mechanism for scaling the data plane.**
   The exponential growth of the global Internet routing table in transit ASes has raised scalability concerns. Attempts to reduce the size of the table require global coordination. Common practices to filter routes can lead to black-holes.

   a) The thesis presents a mathematical model of the problem of distributing FIB rules over a distributed switch and model it as a linear optimisation problem.

   b) A design of a heuristic algorithm which trades the efficiency for speed and perform a comprehensive evaluation of the algorithm using synthesis and real data traces.

1.4 Thesis Organisation

The remainder of this thesis is structured as follows:

Chapter 2 first provides an introduction to *interdomain routing* and its core routing protocol, BGP and to the concept of *SDN* and existing SDN architectures. In the second part, the chapter reviews existing solutions to the three issues of the interdomain routing in transit ISPs: *routing programmability*, *routing flexibility* and *routing scalability.*

Chapter 3 provides an overview of the design and implementation of the centralised routing controller *gRCP*. gRCP controller addresses the routing programmability of the proposed solution for interdomain routing.

Chapter 4 provides an overview of the design and implementation of *fBGP*, a flow-based border router. fBGP offers the flexibility for performing customer-specific routing. fBGP addresses the routing flexibility of the proposed system. Chapter 5 presents a method and algorithm for addressing the scalability issue by decomposing and distributing the FIB table across a distributed switching fabric.

Chapter 6 provides an in-depth comparison between the proposed solution and the potential legacy approach to the routing problem of interdomain routing. Finally, chapter 7 summaries the thesis, the contribution made by the thesis and discusses the future work.

Chapter 2

Background and Related Work

This thesis investigates ***methods to improve the capability and performance of interdomain routing*** in a transit ISP and the ***application of SDN into realising an incrementally deployable solution***. There are a wide range of clean-slate proposals addressing various aspects of interdomain routing such as Pathlet [24] (scalability and flexibility), NIRA [25] (path programmability) and SCION [26] (path programmability, scalability and security). However, they do not meet the requirement for incremental deployment, therefore they are not covered in this review.

This chapter is organised into three main sections. Section 2.1 provides an introduction to interdomain routing and BGP. The definition of SDN, an introduction to several popular SDN architectures, particularly OpenFlow - the most popular implementation of SDN in the literature, and existing SDN solutions to interdomain routing in general and in transit ISPs in particular, are also provided in Section 2.2. The performance attributes and capabilities of interdomain, namely routing programmability, flexibility and scalability, are discussed in Section 2.3.

2.1 Introduction to Interdomain Routing

This section gives a brief introduction to interdomain routing, and BGP. It also provides a discussion of the challenges facing the ISP operators.

2.1.1 Overview

Interdomain routing is described as the routing between ASes, in contrast to routing within an AS which is called *intradomain routing*. Interdomain routing plays a critical role of being the backbone of the Internet that provides connectivity between any host or device in the Internet.

In interdomain routing, participating ASes are operating in the same global address space (i.e. IPv4 and IPv6). The address space is divided into small portions called prefixes which are allocated to different ASes. Interdomain routing is the process of exchanging information and establishing paths from an AS to prefixes of other ASes.

An AS who has ownership of a prefix originates routing information (called route) about the prefix to its neighbouring ASes. A route contains information about the sequence of ASes used to reach to the prefix. These neighbours, in turn, propagate the route after appending its own ASN to the path. This process continues until the route is propagated to all ASes in the Internet. An AS makes autonomous decisions on who it wants to propagate a route to as well as how a route from a neighbour will be processed. Routing decisions are typically associated with the business relationship between two ASes. Specifically, the types of interconnection between ASes determine how interdomain paths are formed in the Internet.

BGP is the dominant protocol used for propagating the routing information between ASes. BGP and interconnection are two main components of the current interdomain routing in the Internet. The next sections discuss Internet interconnection and BGP in more detail.

2.1.2 Internet Interconnection

Links between ASes are traditionally categorised into three classes: transit, peering and sibling. A *transit link* reflects a *Customer-to-Provider* (c2p) relationship between an AS (the customer) and a transit AS (the provider). *Peering links* are established between two ASes who have mutual agreement to exchange their own traffic or their customers' traffic at no cost. Two ASes (e.g. operated by the same organisation after merging) may have *sibling links* between them. Some variants of interconnections exist, including partial transit (i.e. limited destination) and hybrid (i.e. two ASes have both transit and peering interconnection at different locations).

Figure 2.1 illustrates simplified interconnection topology in the Internet. Arrows and lines represent c2p and p2p links, respectively. At the core of the Internet are tier-1 transit ISPs: there are only a handful of them but they have a critical role of keeping the Internet connected [12]. A tier-1 ISP only has peering and transit links (p2c). Tier-2 ISPs are customers of tier-1 ISPs and so on. Typically, traffic flows a from a customer via a c2p link, possibly crossing a p2p link and finally a p2c link before reaching the destination. The real-world topology is much more complicated [27] with almost 35K active ASes and 146K interconnections [28] (as of 2011). CAIDA reports 70K ASes as of 2017. ASes tend to keep the details of their interconnections secret as this information is sensitive for the business. Therefore, inferring an accurate Internet topology has always been an active research area [29].

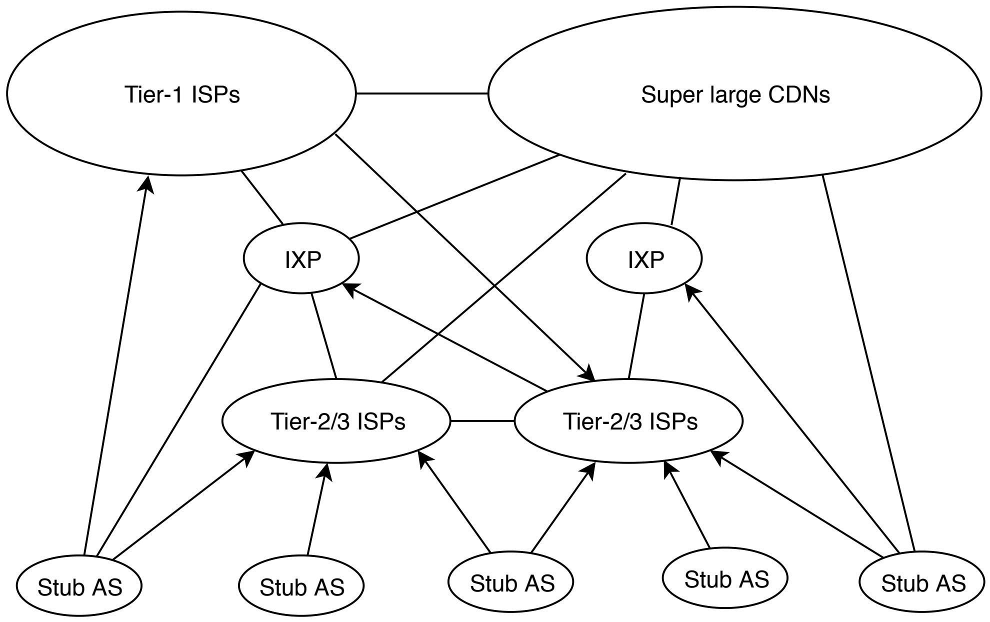


Figure 2.1: A simplified interconnection topology of the Internet

Transit links are commonly associated with a Service Level Agreement (SLA). A SLA is the commitment of the transit provider for the quality of service. SLAs cover aspects of the service including availability, quality (which includes delay, throughput and/or jitter) and responsibilities. However, while an AS has total control of its intradomain routing, it has limited influence on the routing decision made by other ASes, thus the committed SLA by a transit provider is unlikely to be enforced a long the end-to-end path [30].

Many peering links between ISPs are established at colocation facilities and IXP. A study shows that outages at these facilities are common and can last as long as tens of minutes in average [31]; 40% of them even exceed 1 hour. This frequent outages of the peering infrastructure has great impact on remote networks. The same study also finds that the median Round-Trip

Time (RTT) increases more than 100 msec during the outages, with 30% of the paths still having an increased RTT of 40 msec after the outages. The study also notes that outages of peering infrastructures not only have local impact but impact on remote networks as well.

Interdomain routing is policy-driven; end-to-end paths are formed by the collective policy applied by the ASes along the path. Studies have shown that the Internet exhibits path inflation i.e. paths are longer than necessary, affecting quality. The intradomain routing topology and peering policies of ISPs are contributors of the path inflation [32].

Valley-free routing is a universal property of the Internet interconnection. Valley-free routing dictates that traffic between two large ASes should not transit a small (customer) AS [33]. Violation of valley-free routing can significantly degrade performance (i.e. the small AS cannot accommodate the traffic). However, studies found that valley-free violations are pervasive with many valley paths lasting for months [34].

Having said all that, stub ASes (end networks) have limited influence and control over the formation of the end-to-end Internet paths. It is mostly up to the transit providers to decide how their customer routes will be propagated. Moreover, routing changes (i.e. topology, policy) in transit ISPs can significantly contribute to end-to-end packet loss [35]. Meanwhile, the associated SLA of transit services is likely not enforced across the path.

2.1.3 Border Gateway Protocol

BGP is a path vector protocol: it does not route traffic on the shortest path like many other intradomain protocols.

Two BGP-speaking routers establish a TCP session between them to

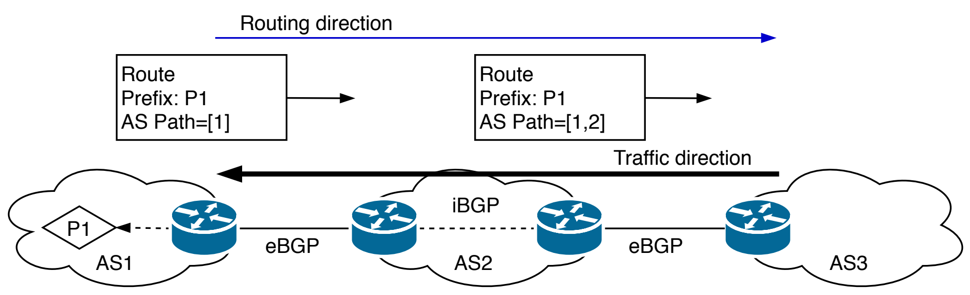


Figure 2.2: BGP route announcement and the established path.

exchange routing update messages. A router sends to its neighbours an announcement message to inform them that it knows the path to the associated prefix(es). A withdrawal message is sent to a neighbour when the router does not want that neighbour to use the path or it actually has lost the path. If two routers are in different ASes, its BGP session is called eBGP otherwise iBGP. Figure 2.2 depicts the high-level operation of BGP. The traffic flows in a reverse direction of routing information. A route announcement contains two key attributes: prefix(es) and AS path along with many other attributes. The list of popular attributes are shown in Table 2.1. Local-preference (called local-pref for short) is the most used attribute for routing policy [36].

A BGP route consists a number of attributes which a router can use for route selection. The operator defines an import policy based on these attributes which dictates which route a router should install into its local routing table. Attribute values may be modified before the update is sent out. Noted that, none of the existing attributes reflects the performance properties of the path rather than the hop count. Thus, operators typically choose routes based on the path length (AS hops). Nevertheless, other attributes can also be used. Study shows that default routes provided by BGP are usually poor in terms of performance metrics such as throughput and delay

Attribute Name	**Description**
Origin	The origin code tells how BGP learned about the specific route (i.e. three sources: IGP, EGP and incomplete (other source))
AS path	The sequence of ASes to reach to the destination. BGP selects shortest AS path by default
Next hop	Provide information about the next hop for traffic forwarding
Multi-Exit Discriminator (MED)	An optional non-transitive attribute, used to hint external neighbours about the preferred path into an AS
Local preference (or local-pref)	Most used attribute for influencing outbound traffic engineering
Community	Widely used to provide additional information about the route and expected handling of the route

Table 2.1: Popular well-known BGP Attributes.

provided that there exist routes with better performance [37].

A BGP router typically receives multiple routes to the same prefix from different neighbours. It only selects a single route (known as the best route) which is used for forwarding traffic. BGP routers use a pair-wise comparison for selecting the best route: the route attributes of two routes are compared and the superior one is selected. Although the BGP specification does not specifically define the route selection algorithm, the industry agrees on a common and simple decision process as described below:

1. Verify if the next hop can be resolved
2. Prefer the path with the highest local preference
3. Prefer the path with the shortest AS path
4. Prefer the path with lowest origin value ($IGP < EGP < incomplete$)
5. Prefer the path with the lowest MED
6. Prefer the path learned from eBGP over iBGP
7. Prefer the path with the lowest IGP metric to the next hop
8. Prefer the oldest path

2.1.4 Interdomain Routing in Transit ISP

This section describes the topology and architecture of interdomain routing within a transit ISP, and the common routing policy used by ISPs. The popular route distribution mechanism (i.e. route reflection) and its limitations are also discussed.

Within an AS, BGP protocol does not allow a router to exchange a route learned via iBGP to other routers of the same AS. Thus, a full-mesh iBGP connectivity between routers is required. However, such full-mesh topology is not scalable for large networks. Route reflection technique is used. A route reflector is a special BGP router which has iBGP connection to all routers and reflects a route received from one router to the others.

Route reflection increases routing convergence, since update messages traverse a longer path before reaching the final iBGP router. Route reflection reduces path diversity. The route reflector itself runs the same path selection

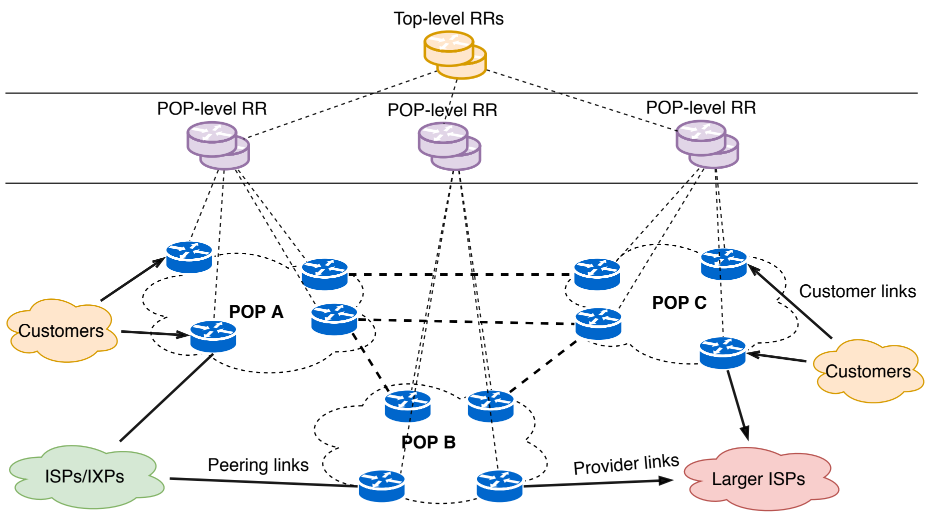


Figure 2.3: A logical routing topology in an ISP.

algorithm as other routers, thus only a single best path being announced. It is most likely that not all the best paths chosen by the reflector would be the best paths for each of all its clients [38].

An incremental solution to the limitation of RR was proposed, called BGP Add-path [39]. The extension allows BGP routers to advertise multiple paths, in order to improve the path diversity [40]. This increases loads on the BGP route computation [41].

Interdomain traffic engineering in transit ASes is a non-trivial task [42]. One of Traffic Engineering (TE) objectives that is common in transit ISPs is to minimise the time interdomain traffic traverses across their network. This is commonly known as "hot-potato" routing. Traffic engineering of interdomain routing in transit ISPs is more challenging than in stub ASes due to the cascade effect.

BGP provides great expressiveness for routing policy. Range of policies can be categorised into four categories: *business relationship*, *traffic engi-*

neering, *security* and *scalability* [36]. Typically, ISPs prefer route learned from a customer, than peer and provider. This is achieved by setting higher local preference for customer routes. The main policy mechanism used by ISPs is local preference.

2.2 Software-Defined Networking

This section provides an overview of the SDN concept and principles, and a brief introduction about different SDN architectures and implementations.

2.2.1 SDN Architecture

Traditionally, network devices are vertically integrated. The control plane which decides how to handle network traffic and the data plane which forwards traffic are bundled in the same hardware platform. This tightly coupled configuration hinders the innovation and evolution of the network [43]. For instance, deploying a new capability or upgrading the forwarding capacity would possibly require a replacement of existing devices.

Noted that, many modern network devices have modular architecture with "hot-swap" capability which allows the control and data plane to be upgraded independently on the fly. This makes the network management easier to some extent. But since the design of the control plane and the communication protocol the two planes are proprietary, the operator has to rely completely on its vendors for innovations. Moreover, network devices are autonomous and make decisions based on limited view (largely based local events) of the network.

SDN breaks this tradition in networking by introducing three key capabilities: separation of the control and data plane, centralisation of the control plane and an open API, as ways to make the network programmable [44]. According to Open Networking Foundation (ONF), a consortium among operators including Facebook and Google, founded in 2011 to promote SDN, in SDN the control plane and the forwarding planes are decoupled (i.e. deployed in separate hardware platforms) and the network intelligence is (logically) centralised in a programmable SDN controller. The SDN controller maintains a global view of the network and makes it appear to applications and policy engines as a single logical device.

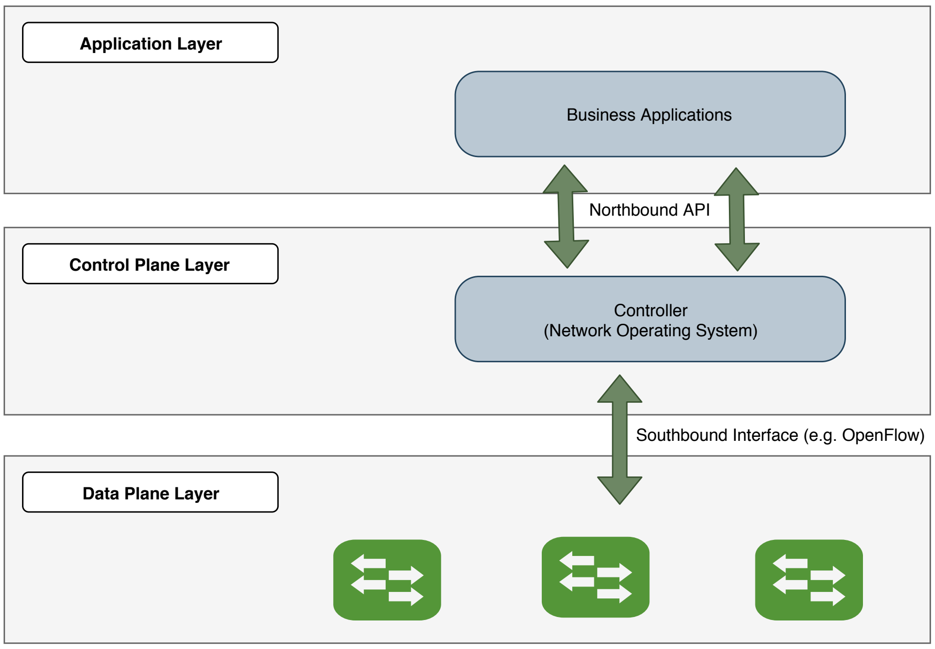


Figure 2.4: A high-level view of SDN architecture.

A simplified view of an SDN architecture is depicted in Figure 2.4. At the top layer are network applications that interact with the network through an open API provided by the controller. The controller manages the network devices and maintains a global view of the network. The data plane consists of network devices whose job is to simply forward packets according to the control plane's decisions. OpenFlow is a popular implementation of SDN architecture.

There have been several proposals for SDN. The next sections presents popular SDN implementations, including OpenFlow, Forwarding and Control Element Separation (ForCES), Path Computation Element (PCE), Interface to Routing System (I2RS) and NetConf.

2.2.2 OpenFlow Architecture

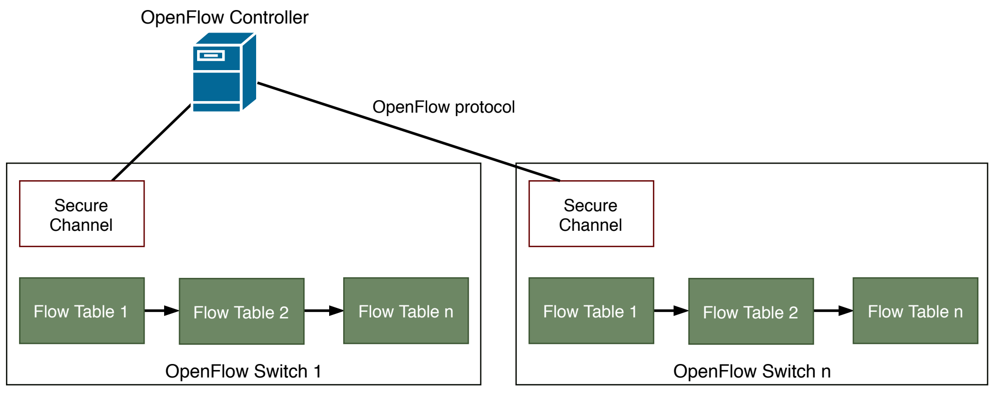


Figure 2.5: Overview of OpenFlow Architecture

An OpenFlow architecture as shown in Figure 2.5, is composed of a controller, a secure channel and one or more OpenFlow-compliant switches [45]. An OpenFlow switch can be configured with one or more flow tables in a sequence (called pipeline), with each containing a prioritised list of flow entries, instructing the switch how packets should be handled. Each flow entry has a match field, a set of instructions and a counter. If a packet matches a flow with highest priority, it will be processed according to the instructions and the counter is incremented. The packet can be forwarded out of a port(s), sent to the next table or dropped. If there is no match, the packet may be encapsulated and sent to the controller.

The controller is a software program responsible for managing the flow tables of switches. The controller can install, remove or modify a flow entry by sending OpenFlow commands to the switch via the secure channel. The OpenFlow protocol specifies the data models and the communication between the controller and the switches. To date, OpenFlow protocol has gone through a number of development, from version 1.0.0 to 1.5.0. However, OpenFlow version 1.3 is currently a widely supported protocol by switch vendors [4]. It has many improvements compared to the previous versions including support for cookies, and flow meters.

Much research into SDN and OpenFlow is dedicated to the development of the OpenFlow controller [46]. As a result, many controller platforms and architectures (both open-source and commercial) have been proposed - NOX, ONOS, Open Daylight, Floodlight, Ryu and POX to name a few. These controllers enable APIs to develop SDN applications in many popular programming languages such as C, Java and Python. Ryu, a Python-based controller was chosen in this thesis for prototyping and experiment, as it is the well-tested, well-documented controller which is accepted in both academics and industry.

OpenFlow switches are available in both hardware and software form [47]. Some hardware OpenFlow switches are distributed as black boxes that exposes only the OpenFlow interface to the external controller. Some are distributed as "white-box" that users can install their own switch control software (e.g. OpenFlow agent). Software switches are commonly available as open-source. Many OpenFlow switches support hybrid mode (i.e. both OpenFlow and legacy mode) to facilitate incremental deployment in an existing network.

OpenFlow is considered to be the enabler for network innovations [48]. To date, it is the most commonly deployed SDN technology in both experiment and production environments [4].

2.2.3 Other SDN Architectures

OpenFlow is not the only SDN architecture. Several architectures have been proposed before OpenFlow. This section gives an overview of these architectures and highlights the key differences with the OpenFlow architecture. A more detail of SDN architecture can be found in two surveys [49] and [50].

ForCES

The Forwarding and Control Element Separation (ForCES) was introduced in 2003 by the IETF ForCES Working Group [51]. The ForCES architecture is composed of two logic entities called the Control Element (CE) and the Forwarding Element (FE). The CE is responsible for control and signalling functions whereas the FE provides per-packet processing according to the instructions of the CE.

The ForCES architecture is based on a building block called Logical Function Block (LFB). The LFB is a well-defined, logically separable functional block that resides in an FE and is controlled by the CE via the ForCES protocol [1]. OpenFlow tables are equivalent to LFBs but with more flexibility. The function of a LFB (e.g. Ethernet processing) and the order of LFBs (i.e. packet flow) are defined in advance, whereas OpenFlow tables can match on arbitrary packet headers and move the processing (i.e. the "Goto" capability) to any subsequent table.

The ForCES architecture is not widely adopted due to the lack of clear language abstraction definition and controller-switch communication [52].

PCE

The Path Computation Element (PCE) architecture was proposed to facilitate the computation of paths (i.e. labelled switch paths or LSPs) in a MPLS-enabled network [53, 54]. The PCE architecture has two key elements: the PCE and the PCC (path computation client). The PCC is usually implemented in a network management system or in a node, initiating a request for a LSP path to the PCE which in turn performs the computation based on its global view of the network (i.e. the traffic engineering database or TED).

In comparison to OpenFlow, PCE is limited to path computation in MPLS networks whereas OpenFlow offers programmability to implement new functionality in the data plane.

I2RS

I2RS is an IETF standard proposed in early 2013, which defines protocols and mechanisms to control the dynamic state in routers and switches [55]. The I2RS architecture consists of the clients and the agents. The agents are implemented in network devices that interact with the RIB table (i.e. the

[1]https://tools.ietf.org/html/rfc6956

routing table) and the routing policy. The client(s) controls the agents and provides interfaces to applications.

Similar to the PCE architecture, I2RS focuses on a specific domain in networking (i.e. the routing control). In contrast, OpenFlow aims to improve the programmability of the data plane. I2RS is being standardised. Router vendors have not yet supported and implemented I2RS.

NetConf/Yang

NetConf is a replacement of the traditional simple network management protocol (SNMP) which provides the functionality for installing, manipulating and deleting configuration of network devices [56]. Yang, a data modelling language, is used to model configuration as well as state of network elements [57]. Yang is designed for both human and machine readability. NetConf/Yang provides a simplified and extensible way for network automation, an alternative to the conventional command line interface (CLI). Yang modules are to be converted into XML format, populated with configuration data and transported to the device by NetConf.

In comparison to OpenFlow, NetConf offers limited programmability. It can tune the behaviour of a functionality implemented in a network device (i.e. configuration of routing) but does not allow implementation of a new functionality. OpenFlow, in contrast, allows full programmability of the data plane, thus making it possible to realise new functionality in the same hardware.

2.2.4 SDN Applications

Most existing works are dedicated for developing SDN-based solutions for enterprise networks, for instance, traffic monitoring and analysis [58, 59], traffic engineering [60], network management and virtualisation [61].

With regard to interdomain routing, the authors in [62] use SDN to develop a new interdomain protocol called multi-dimension link vector (MLV) that allows the exchange of fine-grained virtual network views between ASes. The virtual network views are composed of intra-domain virtual links and inter-domain physical links. The link information includes the multi-dimensional prefixes which are allowed to pass that link, the bandwidth, the utility, version and other features of the link. The network view is exchanged by having each AS to send the link vector message to its neighbours according to their commercial relationships. The link vector message includes a directed link list together with the multi-dimensional prefixes which are allowed to pass through that link list. Exposing the intradomain topology can pose serious security risks and scalability problems.

In other works, the direct programming of the forwarding devices was leveraged in SDX [11], a routing controller supplementary to BGP in IXP environment. SDX consists of a route server, a policy compiler and a switching fabric. The route server receives eBGP routes from member networks and selects best routes based on the standard BGP decision process. The policy compiler takes members' policy definitions and best routes as input, compiles OpenFlow rules and installs to the switch. Since SDX targets IXPs of which the network size in terms of number of devices, number of customers and number of routes is small, SDX assumes a single route server the whole network. In SDX, any member can define arbitrary fine-grained policy respected to routes advertised by or advertised to the member. Thus, this policy does not take into account the routing objective of the ISP itself if SDX is to be deployed as the controller. RouteFlow [10] and SDN-IP [9] have proposed architecture for hybrid networking of BGP and OpenFlow. However, these works are limited to demonstrate how BGP can be deployed in OpenFlow without further investigation into its limitations can be addressed.

2.3 Performance of Interdomain Routing

This section reviews the literature regarding the three performance attributes of interdomain routing: routing programmability, flexibility and scalability.

2.3.1 Routing Programmability

Networks are traditionally managed via the command line interface (CLI) which is designed specifically for human. The CLI makes it extremely difficult to automate management tasks, as CLI does not come with data models and it varies from vendor to vendor.

Moreover, traditional network devices are deployed as appliances with a specific set of built-in functionality, for instance a switch used for Layer 2 forwarding, a router for Layer 3 forwarding and a firewall for Layer 4 packet filtering. Deploying a new functionality means replacing the hardware or installing new software onto the device.

The concept of programmable networks has been proposed as a way to simplify the task of network management and performance tuning and to break the “Internet ossification” [50]. Traditionally, network devices are managed through low-level, vendor-specific commands, therefore it is a difficult and error-prone task to manually transform high-level policies into low-level configuration commands. Moreover, as network devices are *black boxes*, running the vendor's closed-source codes, introducing new functionality to the network requires replacement or upgrade of software and hardware.

A programmable network enables third-party codes to run on the control and/or the data plane [63]. This capability enables the operator to quickly implement and deploy new network services and functionality.

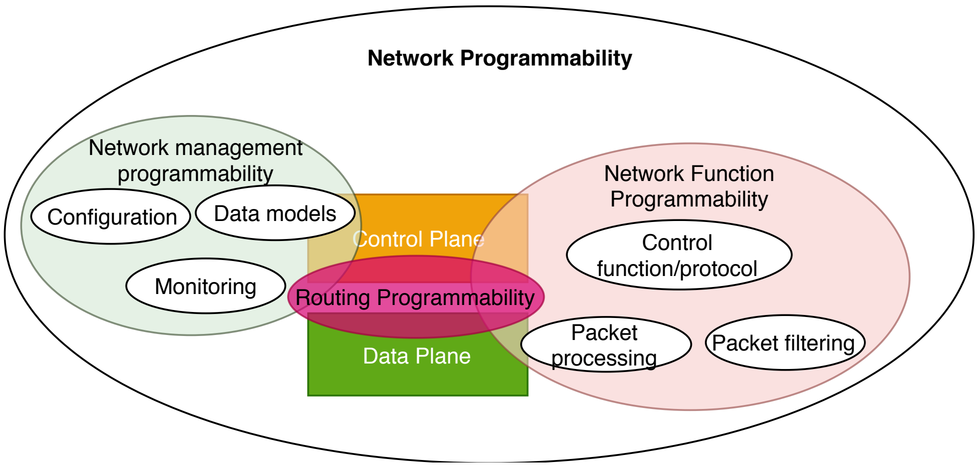


Figure 2.6: Different aspects of network programmability

There are two main aspects regarding programmability in networking: *network management programmability* and *network function programmability*. Figure 2.6 illustrates the components and interaction of the two programmability aspects. Network management programmability mainly interacts with the control plane, whereas Network function programmability works with both the control and the data plane. Routing programmability resides within the scope of network programmability and requires both the control and the data plane.

There are a vast number of research works on programmable networks [64, 49, 50, 5, 44, 50]. In the context of interdomain routing, routing programmability is concerned with functionality of interdomain routing and manipulation of the BGP policy. Currently, programmability of interdomain routing is limited to a static set of attributes and actions provided by BGP, for instance to filter a route if one of its attribute has a certain value and to modify the value of an attribute.

In the literature, the proposed architecture for routing programmability is typically composed of a centralised routing controller which maintains a complete view of the network and provides southbound interfaces to applications where the high-level routing objectives are implemented.

Past effort to improve interdomain routing programmability in traditional networks includes: Routing Control Platform (RCP) [65], Morpheus [14, 1] (academia), Intelligence Route Service Control Platform (IRSCP) [13], Application-Based Network Operation (ABNO) [53], and Juniper Egress Peering Engineering [66] (industry).

RCP [65] is the first and a simple route controller which performs routing computation for all routers in the network in a centralised manner (see Figure 2.7). With a complete network view, RCP can compute optimal routes for each router. However, RCP does not enhance BGP's functionality in any way nor does it expose programmatic interface to external applications. It simply computes routes using the standard BGP selection algorithm. It is to show that centralised routing is technically feasible for large networks such as tier-1 ISPs, even with the controller running on a commodity PC. RCP is the first prototype to demonstrate the possibility and benefits of separating routing from routers.

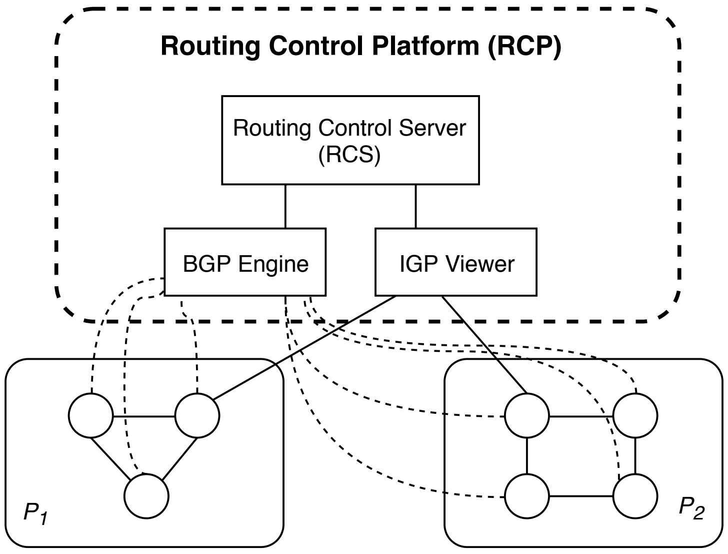


Figure 2.7: RCP architecture.

Morpheus [1] proposes to enhance the functionality of BGP by making the routing aware of external metrics rather than just the built-in attributes. In Morpheus, routes are first classified and tagged and given an appropriate weight for each tag. The algorithm is then to select routes with the highest weight. By having multiple classifiers (e.g. one for latency, bandwidth, security, stability, etc) and adjusting the weights, the operator can select routes that satisfy conflicting policy objectives. Morpheus suggests running multiple decision processes (each with its own weight settings) in order to realise different policies and satisfy different customers. Morpheus proposes to use VRF, MPLS or IP tunnels to realise multiple routes. Morpheus does not expose API to applications. Moreover, its configuration interface requires the human operator to define pair-wise comparisons (per criterion) of all alternative routes. Morpheus is more powerful and flexible than the BGP's standard selection algorithm. However, it does not meet the requirement for dynamic and programmatic network control.

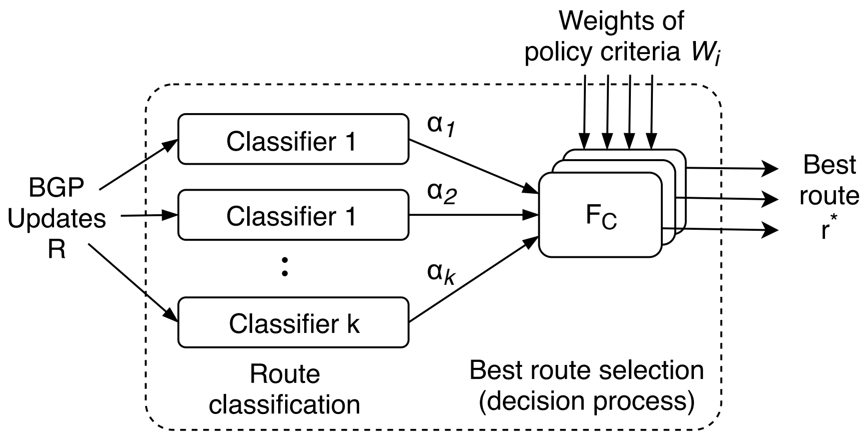


Figure 2.8: Morpheus's route selection process [1].

The industry is also interested in enhanced routing control for BGP. IRSCP [13] is the route controller developed by and deployed in AT&T network. IRSCP supplements the traditional BGP decision process by allowing external applications to override the decision with a per-destination, per-router explicit ranking of traffic egresses. This allows to take into account external information such as available bandwidth of egress links. In IRSCP, the controller performs the standard BGP decision from step 0 (ignore unreachable egress) to step 4 (lowest MED). The next step is based on the ranking provided by applications. The authors argued for overriding the hot-potato policy. Similar to Morpheus, IRSCP has the full visibility of routes and is built upon the notion that the underlying network is capable of destination-based routing and the use of BGP as the communication protocol between the controller and the routers.

AT&T extended the RCP concept and developed a controller system called Intelligent Route Service Control Point (IRSCP) [67] which offers *dynamic connectivity management*. It enables operators to perform selective blackholing, planned maintenance dryout, VPN gateway selection and network-aware load balancing. Similar to RCP, IRSCP communicates with routers via IBGP to learn routes and send the selected routes back to routers. IRSCP is implemented based on Quagga and offers primitive commands (such as addblackhole/delblackhole) to the operator. In a later version, IRSCP is extended to allow control via applications. The decision is opened up to application which can utilize external information into the decision process.

ABNO framework [53] - a PCE-based architecture for application-based network operations. ABNO provides interfaces to network applications (e.g. web, and email) to request for and make reservation of network services (e.g. connectivity, reliability) and resources (e.g. bandwidth). ABNO framework aims to make the network application-driven. Currently, the network is responsive to management commands driven by a human user. Applications should be able to make reservation for connectivity, reliability, and resources. ABNO does not explicitly specify how the TED database is extended to support interdomain routing. ABNO is designed for intradomain routing, particularly for provision and management of LSPs between border routers. ABNO and SDIRO can work together to provide a complete solution for ISPs. Juniper EPE [66] is another industrial proposal by Juniper.

2.3.2 Routing Flexibility

The term *routing flexibility* refers to the granularity the operator has, to control and implement routing policy in the network. There are two aspects of routing flexibility: *control plane flexibility* i.e. the granularity of control in the control plane and *data plane flexibility* i.e. the ability to utilise the diversity of paths in the network.

The control granularity depends on the routing protocol used in the control plane. For instance, OSPF offers limited flexibility. It calculates path based on the shortest path algorithm. The only control mechanism is to change the link weights. BGP allows more control. BGP as a standard does not specify how a router should select routes. However, the common route selection algorithm implemented by vendors can be controlled based on any route attributes or any combination of them.

In the data plane, routing flexibility refers to the ability to utilise multiple paths for different forwarding policies. For instance, traffic loads can be shared between paths.

In the context of interdomain routing, the intradomain network is treated as a single virtual big router with interfaces being border routers. Routing flexibility is the ability of this "single big" router to enforce diverse forwarding policies to control packet forwarding from one virtual interface to the other.

This paper [68] discusses how iBGP policy can be utilised to improve routing flexibility in the traditional network and the risks of doing so.

In the data plane, MPLS and Segment Routing (SR) are two mechanism/architecture for flexible routing in the traditional network.

This work [69] proposes a backward-compatible approach to end-to-end QoS in interdomain routing. It relies on MPLS tunnels. Elaborate more.

2.3.3 Routing Scalability

In the Internet, routing scalability has been an outstanding issue since the early 1990s. Two main main concerns about routing scalability are the increasing number of routes in the routing table (see Figure 2.9) and the routing churn [70]. The growing number of devices and customer networks (i.e. stub ASes) connected to the Internet have contributed to the scalability issue.

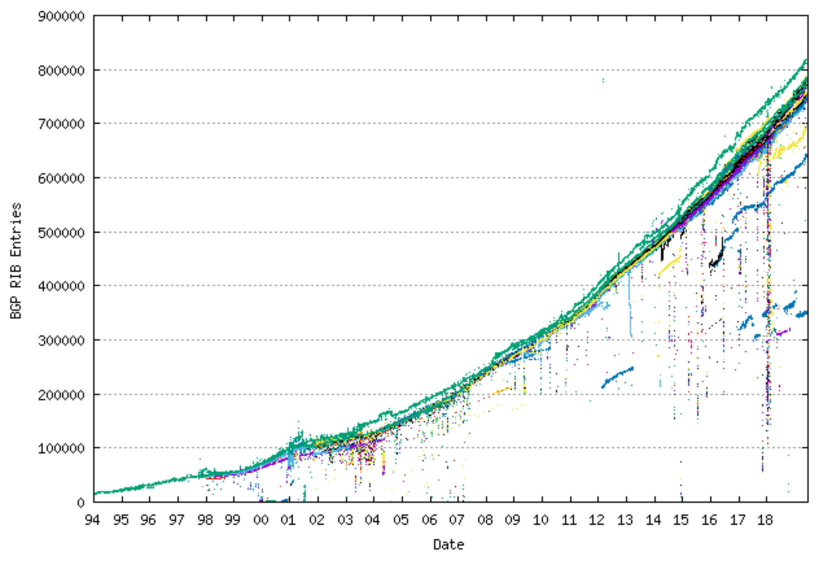


Figure 2.9: BGP routing entries in the Internet core routers (source: https://bgp.potaroo.net)

The Internet's only scalability mechanism, i.e. route aggregation, does not work well mainly due to multi-homing and the adoption of provider-independent addressing. Route aggregation requires a good hierarchical topology to work: a provider aggregates multiple customer routes (prefixes) into a super one before advertising to its upstream providers. However, customer networks often seek for more reliability, availability and control by multihoming to several different providers and by using their own IP prefixes. Now, if one provider tries to aggregate the customer's prefixes, the traffic toward the customer's network would completely traverse the other providers.

The dilemma is that customer networks wants more routing control and they are doing so by advertising a number of small prefixes. ISP providers are struggling (e.g due to economic constraints) to promptly upgrade their networks to the latest technologies to keep up with the demands. The current common practice of ISPs is to limit the number of prefixes a customer can advertise to the Internet (hence limited control for the customers).

In coping with the growth of the routing table, a number of works have been proposed. They can be categorised into classes: clean-slate and evolutionary approach.

Numerous past work has investigated compression techniques to address Internet routing scalability [20]. Compression is also computationally expensive and has limited effectiveness when routes have diverse next-hops. The main drawback of this technique is that the operator loses fine-grained control over routing. For example, it does not support operators' desire to set different QoS for different set of destinations. It can also lead to packets being forwarded even though no actual route exists (i.e. "punching hole" effect) Nonetheless, this method is complementary to ours and can be applied before the placement to achieve a better result.

Another approach is offloading forwarding rules to cheaper hardware (e.g. a software router). In [71], the authors propose a "configuration only" approach in which the conventional FIB table is decomposed using virtual prefixes and the subtables are then installed to one or more routers at carefully chosen locations. Ingress routers are then configured to forward traffic destined to a virtual prefix to the router which is responsible for that prefix. To avoid manual configuration, the work in [72] uses SDN to automate the system. Our work advances this approach by considering more complicated forwarding semantics. Our system shares some commonality with DIFANE

[73] in which the controller partitions the space of rules and distributes to switches. However, DIFANE works with multidimensional rules, and does not scale to hundreds of thousands of rules as in interdomain routing. Although their results do not show, computational overhead is probably high. Offloading forwarding rules can also be done locally in a router by moving rules from expensive memory to inexpensive one [74, 75]. A recent work in [76] considers offload complex SDN rules. However, it is not efficient as FEs are still duplicated in different routers. Moreover, the use of software switching can result in inconsistent packet switching latency [77].

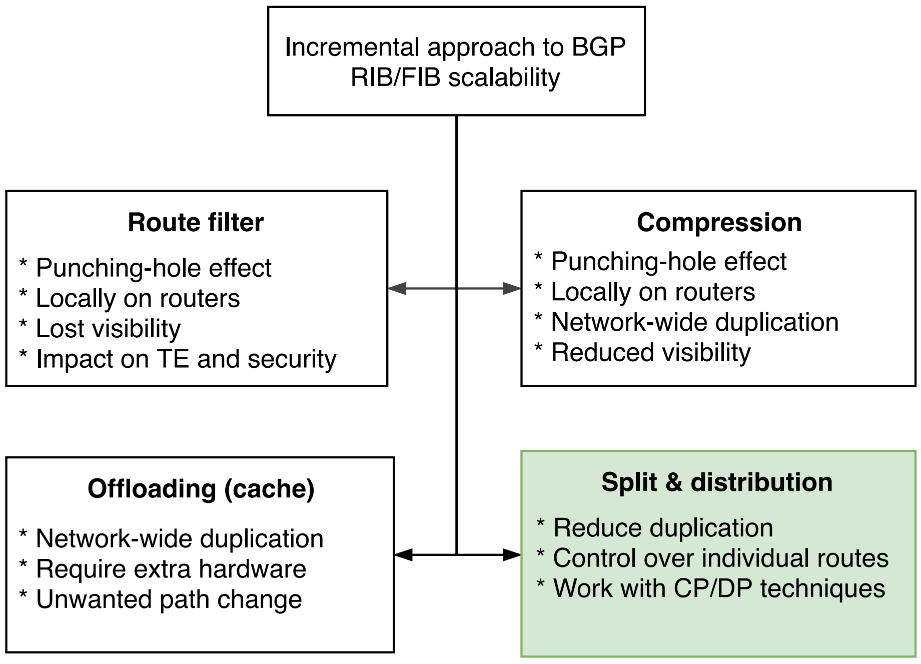


Figure 2.10: Summary of approaches to Internet routing scalability.

Existing works have considered the split and distribution approach to deal with ACL scalability in enterprise networks [78, 79, 80]. For instance, [81] proposes a legacy architecture to deal with routing scalability in ISP networks. Distributed servers select routes on behalf of routers. Each router maintains its share of Internet routes in addition to a cache of routes currently used to forward traffic.

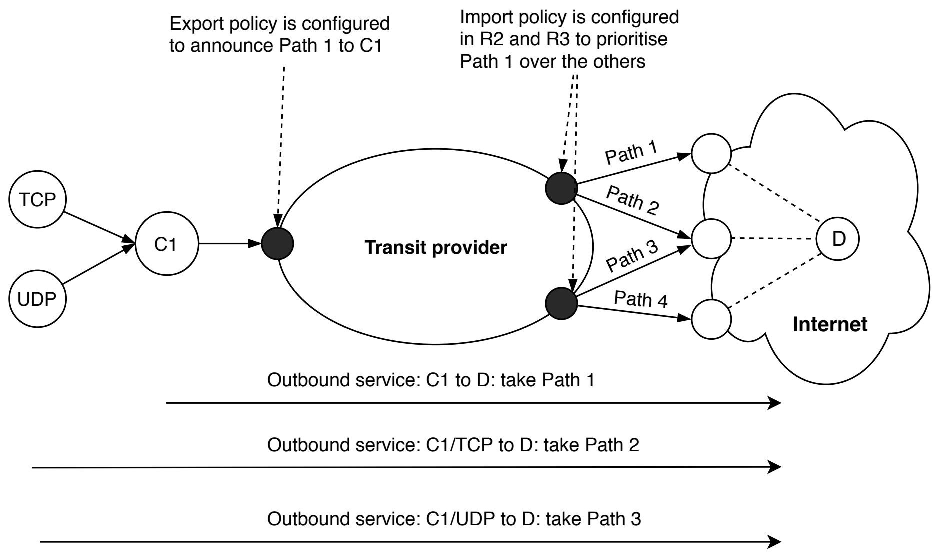
2.4 Summary

This chapter introduced interdomain routing in the Internet and BGP, the de facto protocol for interdomain routing, and several popular SDN architectures. It reviews the recent literature on SDN-based solutions to improve interdomain routing in transit ISPs. Three performance attributes, namely programmability, flexibility and scalability are considered importantly for transit networks. The subsequent chapters leverage SDN to develop a number of SDN-based solutions for addressing these performance attributes.

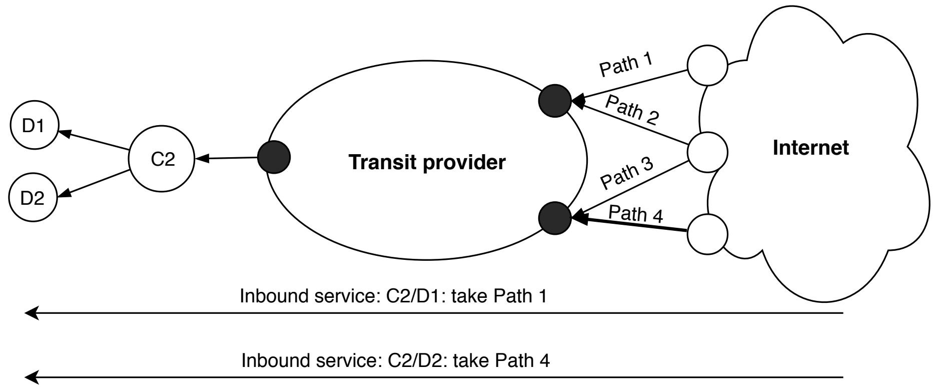
Chapter 3

Design and Implementation of gRCP

This chapter presents the design, implementation and evaluation of an interdomain routing control platform for ISPs which allows the operator to programmatically control and manage its transit services through a well-defined programmatic interface. The design addresses one of the key issues in the controller design: representation of the global network view. The key idea in designing the controller is to use the property graph model to represent the network intelligence and the declarative query language to interact with the network.


Export policy is configured to announce Path 1 to C1
Import policy is configured in R2 and R3 to prioritise Path 1 over the others
TCP
UDP
C1
Transit provider
Path 1
Path 2
Path 3
Path 4
D
Internet
Outbound service: C1 to D: take Path 1
Outbound service: C1/TCP to D: take Path 2
Outbound service: C1/UDP to D: take Path 3

(a) Outbound transit service abstraction


D1
D2
C2
Transit provider
Path 1
Path 2
Path 3
Path 4
Internet
Inbound service: C2/D1: take Path 1
Inbound service: C2/D2: take Path 4

(b) Inbound transit service abstraction

Figure 3.1: Transit service abstractions for inbound and outbound services. $C1$ and $C2$ denote customer routers. $D, D1$ and $D2$ denote destination IP prefixes. The transit service abstraction is defined as a mapping of a source and a destination entity to a path.

3.1 Design for Programmability

3.1.1 Transit Service Abstraction

In the traditional network, provisioning a transit service is a complicated process. Consider a simple ISP in Figure 3.1, to provide a customer $C1$ with a transit service to the destination D first the operator will need to define an import policy on the border routers $R2$ and $R3$ which have learned paths to D. The import policy eliminates paths that the operator do not want to use and prioritise the others according to some policy objectives.

Assume path $P1$ is chosen (i.e. because it satisfies some policy objectives). Next, the operator specifies an export policy which determines whether $P1$ can be announced to the customer. This is often determined based on the agreement between the provider and its customers. When $P1$ is announced to $C1$, the customer will be able to send packets to the destination D. The operator also needs to provision the reverse path so that the customer can receive traffic from the Internet.

In defining the reserve path, the operator imports all prefixes (many enterprise customers usually advertise only a few prefixes) from the customer and exports them to other ISPs. The operator can choose any neighbouring ISP as long as the path is visible to every network in the Internet. It is up to the provider to decide which customer path to be exported to which neighbour. For example, the provider decided to use paths $P1$ and $P4$ for the customer's inbound traffic to $D1$ and $D2$ respectively.

An example of the import and export policy configuration defined in Cisco's route-map language are shown in Listing 3.1. Line 1 to 12 (including comments) define two import policies for routes received from the Peer 1

```
 1  ! policy definition for the provider (ASN 100)
    ip as-path access-list PROVI1 permit ^100
 3  route-map PROVI1_IN permit 1
        match as-path PROVI1
 5      set local-preference 80
        set community 1:123
 7  ! policy definition for the peer (ASN 200)
    ip as-path access-list PEER1 permit ^200
 9  route-map PEER1_IN permit 1
        match as-path PEER1
11      set local-preference 90
        set community 1:123
13  ! export to customer
    route-map CUST_OUT permit 10
15  ! export policy for PEER2
    ip community-list expanded NO_PEER permit 1:123
17  route-map PEER2_OUT deny 10
        match community NO_PEER
19  ! apply policy
    router bgp 1
21      address-family ipv4 unicast
            neighbor 10.0.1.1 route-map PROV1_IN in
23          neighbor 10.0.1.2 route-map PEER1_IN in
            neighbor 10.0.2.1 route-map PEER2_OUT out
25          neighbor 10.0.3.1 route-map CUST_OUT out
            neighbor 10.0.3.2 route-map CUST_OUT out
```

Listing 3.1: Example of import/export policy in Cisco's policy language.

and Provider 1. The policies set higher local-preference for the peer routes than for the provider ones (line 5 and 11) so that they are more preferable. Export policies defined from line 13 to 18 prevent routes from the peer and the provider to be exported to Peer 2 but allow all routes to the customer. Policies are applied in line 19 to 25. Some lines were omitted for clarity. This configuration snippet is typically very long depending on the number of neighbours and particularly the policy granularity (size of peer groups, size of aggregate prefixes, etc,.).

When the policy configuration is specified, it can be applied to the router (through the CLI or an automated tool like Ansible [1]). While it is possible to write applications to generate the policy configuration, it is rather difficult with language like Route-map. Moreover, it is not possible to load balance traffic (e.g. send customer 1 and 2 via the peer 1 and provider link respectively) since BGP does not directly support this capability. If the link peer 1 is congested, the operator will have to determine some prefixes to be moved to the provider link. Then the policy has to be updated at finer granularity.

The transit provision is equivalent to creating two working virtual links (one for each direction) between the customer and Internet destinations while fulfilling its own and its customer routing needs, for instance lower transit cost and maximised utilisation. Thus, this whole process can be abstracted as a list of mappings between customers (or in more general term origin entities) and the Internet paths. Transit service provisioning is then meant to creating such mappings. The granularity of the mapping does not need to be limited to a customer but can be extended to protocols or applications that the customer is using. For instance, the arrows in Figure 3.1 represent the transit service for customer $C1$, and its two TCP and UDP applications. A mapping can be expressed as below:

$$\text{A mapping } m : (O, D) \mapsto P$$

where O is the originator, D is the destination and P is the path. These abstractions will be discussed in the next section.

While the mapping specification of transit service can be specific, defining and managing such mappings per customer and per destination prefix

[1]https://www.ansible.com/

is clearly not scalable, even for a few hundreds of customers and prefixes. Moreover, from the high-level policy's point of view the operator may not need to consider what exact path is being used but the kind of path should be used. For instance, the operator may be interested in forwarding customer's traffic via a path learned from a peer rather than an underutilised path from a provider. Such specification may be applied to not only a single destination but a group of them. Having said that, the operator needs a declarative way for expressing the mapping specification which ensures expressiveness, simplicity and scalability. This chapter describes a query language which can fulfil such requirements.

3.1.2 Global Network View Abstraction

The Global Network View (GNV) is the abstraction of the interdomain routing state. The GNV provides a centralised access to network topology, administrative information and router states, AS relationships, received routes, and traffic statistics.

While the GNV in this context serves the same purpose as in other controller designs such as ONIX [82] and ONOS [83], it is specifically designed for the interdomain routing. The ONIX's GNV contains components that are typically suitable for intradomain applications (e.g. switch ports). The GNV consists of the following abstractions:

- **Router**: A Router is a node in the GNV which represents a Forward Controller in the topology. This is not a typical BGP router; it is a logical data-plane component which is described in the Chapter 4. A router typically has *IntraLinks* links to another router, and *InterIngress* and *InterEgress* links to/from *Neighbors*.

- **Neighbor**: A traditional BGP router that belongs to a neighbour AS (i.e. customers, peers or providers). A neighbour has an *InterIngress* and *InterEgress* link to the same border router, which represents the incoming and outgoing traffic from the standpoint of the border router. Neighbors can be further categorised into Customer, Peer and Provider.

- **Application**: An application represents a type of traffic. An application can be connected to a Neighbor.

- **IntraLink**: An IntraLink represents a logical connection between two routers. It is assumed that only one IntraLink exists between two Routers. That does not imply a single physical connection. The operator decides the value of bandwidth of the IntraLink which at max is the sum of bandwidth of all physical links.

- **InterIngress**: An InterIngress represents an incoming link from a Router and a Neighbor.

- **InterEgress**: An InterEgress represents an outgoing link from a Router and a Neighbor.

- **Route**: A route is a link in the graph which represents a BGP route. A route link a Neighbor and a *Destination*. A Route has a number of attributes including BGP attributes such as AS path, and others such as bandwidth (i.e. amount of traffic using this route).

- **Destination**: A destination is a routable address in the Internet. In the current Internet routing system it is an IP prefix (i.e. IPv4 or

IPv6). Type of the Destination is off concern for the controller.

- **Paths**: A path is a sequence of nodes and relationships which traffic takes. For example, traffic from a router R_1 to prefix p_1 may take a path $(R_1) \rightarrow (R_2) \rightarrow (N_1) \rightarrow (p_1)$. With the purpose of separating the interdomain and intradomain, we define a construct for typical paths as a sequence of (ingress), (intradomain link), (egress), (egress link), (neighbor), (route), (destination). That is, the interdomain function does not care about how the ingress router is connected to the egress. It is concerned with *what* (i.e. the capacity, utilisation and performance). A special case would have the ingress and egress router being the same. Paths can be expanded to (customer), (intradomain ingress), (ingress).

- **Mapping**: A Mapping is a relationship between a Router, Neighbor or Application node (i.e. originator) and a Destination. The existence of a mapping implies that traffic from the originator can be sent to the destination. A mapping has attributes to identify what the actual path traffic will take.

An example of a Global Network View is illustrated in Figure 3.2. The ISP has two Routers which are connected to a customer, a peer, an IXP and a provider. There is one destination prefix in the network. Two routes to that prefix were learned from the provider and IXP. A transit service, described as the *MapTo* link between the customer and the prefix, is created. Its attributes describe the service. As compared to the traditional policy expression, the API greatly reduces the complexity.

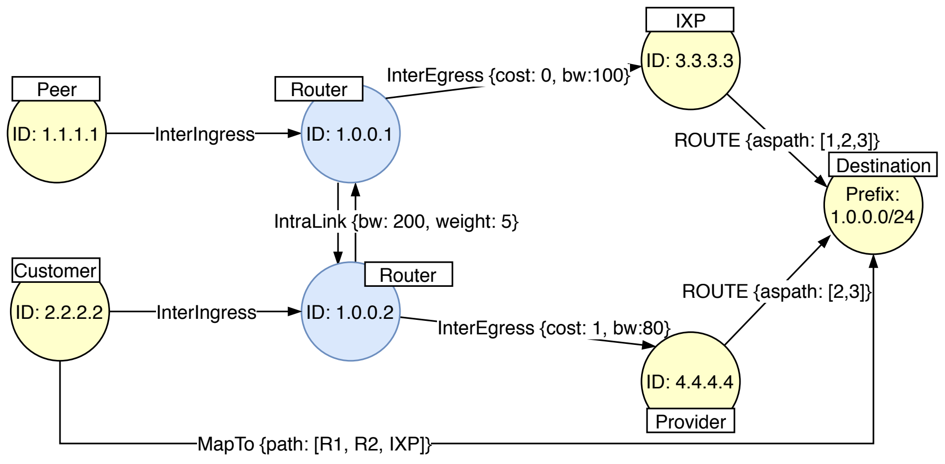


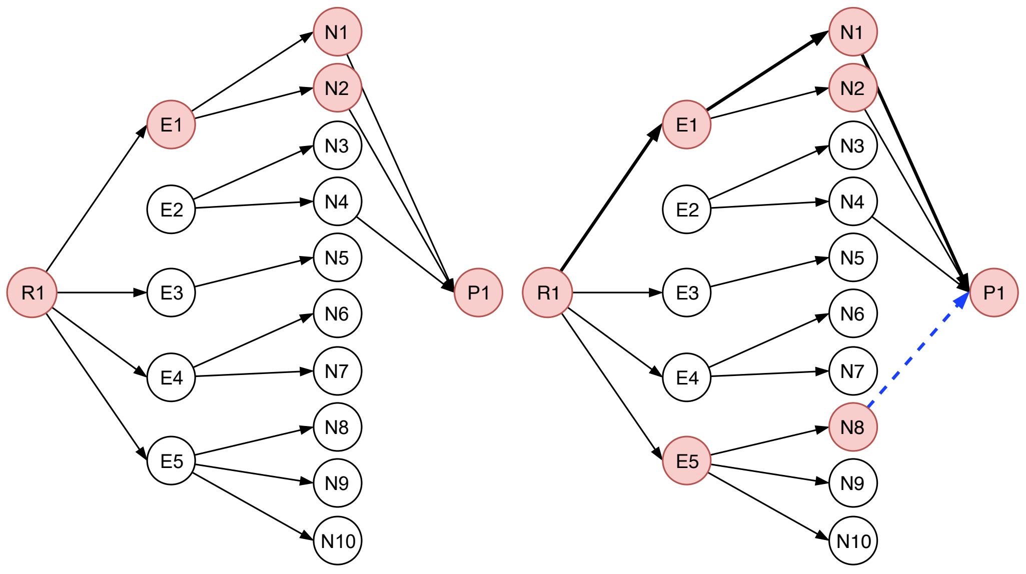
Figure 3.2: Example of Data models in the Global Network View.

3.1.3 Flexible Path Computation

In a traditional network, routers select the best route by performing a pair-wise comparison of route attributes. Thus, to affect the result, the operator tweaks one or more attributes of a route to make it more preferable than the rest.

By modelling routing in a graph, the path computation is turned into a graph traversal between a router and the prefix. By traversing the graph, attributes of nodes and links can be taken into consideration. Figure 3.3 demonstrates this process. The path computation is done for router $R1$ which has intradomain links to four egress routers ($E1, E3, E4$ and $E5$). Those egress routers have interdomain links to neighbours. Links between a neighbour and the prefix $P1$ represent the route announcement received by that neighbour.

In the first scenario which $R1$ does not have route to $P1$ (i.e. fresh computation), the path computation is done by traversing the graph from $R1$ to



(a) Fresh computation (b) Incremental computation

Figure 3.3: Path computation is done by traversing the graph. Shaded circles are nodes that form potential paths that meet the policy requirements. Bold lines are selected path. Dotted line is a new route.

$P1$. Paths that do not meet the requirements will be eliminated. Attributes of the path are computed from attributes of individual links and nodes. Eligible paths are then sorted or compared based some algorithm and the best one is selected. The computation complexity is increasing proportionally to the number of egress routers and neighbours. However, in a typical ISP network the number of paths to a prefix is often less than tens of them. Thus, by reversed traversal (i.e. starting from the prefix), the complexity will be reduced. The path traversal allows for complex computation, for example to consider only paths with a certain capacity.

In the second case, router $R1$ has already had the best path (the bold line) and a newly route is learned from neighbour $N8$. The computation can simply figure out if the new path meet the requirement and perform a comparison with the current one to see if it is more preferable.

3.1.4 Programmatic Interface

This section describes a language for programming transit services.

APIs are built based on declarative language. Each node and relationship in the GNV are associated with a read and write API. *Read* API is used to query for properties, whereas *write* API is used to update, create and delete nodes, relationships and their properties.

For each abstract model, the read API and the write API are expressed as *query()* and *update()* methods respectively. The query method takes parameters to filter specific node or relationship. It is used to define a set of nodes which a certain operation will be applied on. For instance, Router.query() returns all routers in the GNV whereas Router.query(loc="New Work") returns routers located in New York city.

The API can be used against other composite model such as Path which represents a sequence of nodes and relationships. A path has properties including a list nodes and links (that it travels), and other properties such as delay and bandwidth. Paths can be queried using query method. For example, Path.query(src.name=`'R1'`, dst.prefix=`'1.0.0.0/12'`) returns all paths between the node $R1$ and the destination with prefix 1.0.0.0/12. In addition, an *order()* method can be used with query method to sort the result. For example, Path.query().order(cost, aspath_len, weight, bandwidth, delay) will order first cost by ascending, then length of AS path, weight and so on.

These APIs allow for simple expression of routing policy. For example, the code snippet in Listing 3.2 implements a common policy in ISPs which

```
  def gao_import():
2     path = Path.query()
                .filter(Path.bandwidth >= 2)
4               .order(Path.cost, Path.aspath_len,
                       Path.weight, Path.origin,
6                      Path.bandwidth)
      dest = Destination(prefix='1.0.0.0/20')
8     neighs = Neighbor.query(attached_to={'name': 'Router1'})
      create_mapping(src=neighs, dest=dest, path=path)
```

Listing 3.2: API example to define Gao's import/export policy.

prefers customer to peer and provider paths (i.e. Gao-Rexford [84]). It computes routes for all routers in the network. When there are multiple options, it chooses a path based on cost (assume cost of provider links > peer > customer), then AS path length and so on (defined in line 2). The operator first define the path and the preference (line 2 to 6) and then the destination (line 7) and the originator (line 8). The final step is to call the *create_mapping* API. The controller runtime will select the first path in the list and create a mapping between the neighbour and the destination.

3.2 Controller Design

The high-level architecture of the controller is depicted in Figure 3.4. It consists of multiple controller instances, a graph database which stores the global network view and a pub/sub system for message exchange between instances. Control applications implement the routing policy in response to events from the network. Forward Controllers (FCs) are the controlled entities that are responsible for packet forwarding. Routing events generated by FCs are handled by multiple controller instances for scalability. An event can be handled twice by two instances for reliability. Details of how scalability and reliability are achieved are discussed in the next sections.

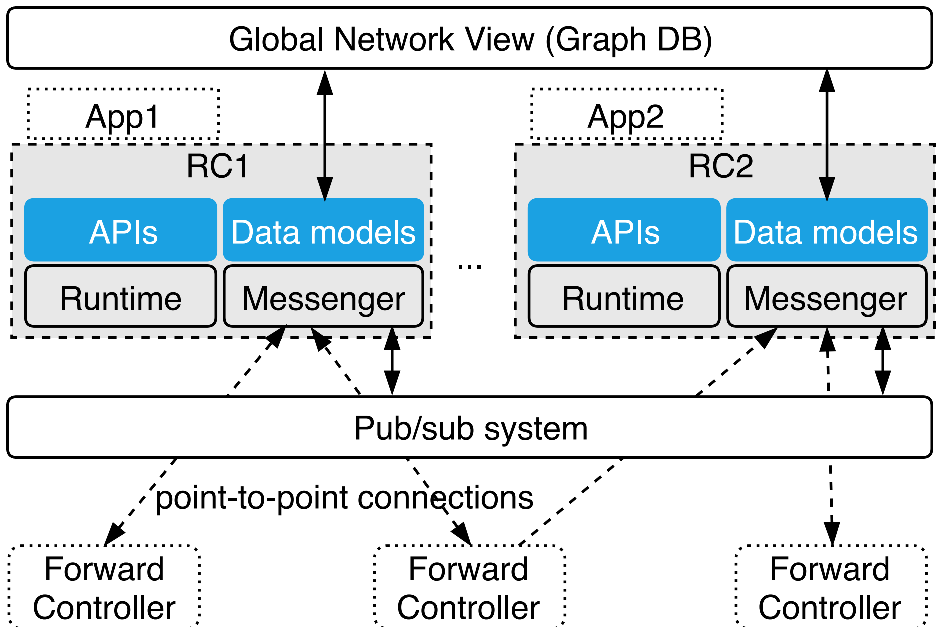


Figure 3.4: Main components of the *gRCP* Controller.

Controller instances communicate through a publish/subscriber subsystem. The pub/sub provides for a reliable and complex point-to-multipoint communication between controller instances. The pub/sub system also enables a loosely coupling between controller instances, allowing for easily launching new instances to take over the increased workload.

All controller instances access the global network view through the same graph database system. An instance will be in charge of updating the GNV upon receiving an event from a FC which it is the master of. This will benefit from many write operations being localised (i.e. affecting a small number of nodes and relationships), thus consistency checking is not necessarily in many cases hence better write speed.

3.2.1 Availability

Availability is of importance for the survival of ISP networks. Controller failures can lead to service disruption which affects the customer traffic.

Controller redundancy approach is taken in order to improve network availability. That is, each FC is configured with a master controller and a list of backup ones. An FC establishes a point-to-point connection to the master controller over a reliable transport protocol. An FC registers itself to the master and periodically sends heartbeats to the controller for liveliness monitoring. If the FC does not hear back from the master for a certain time unit, it automatically switches to the next controller. Similarly, if the master does not receive a certain number of heartbeats from an FC, it considers the FC has become unavailable. The proposed mechanism works similarly to the master/slave configuration in an OpenFlow network.

The master controller's responsibilities include installation and update of forwarding states (i.e. mapping rules) and reception and handling of events generated by the FCs of which it is in charge. Multiple (identical) controllers can be deployed to listen and respond to the same set of events. These controllers do not necessarily have to be the same ones used for FCs. This allows for flexibility in optimising the system.

3.2.2 Scalability

Transit ISPs must be scalable in order to cope with the exponential growth of the Internet. The controller faces two important scaling issues: the rate of routing updates (churn) and the scale of the network (i.e. number of devices, customers and neighbours). In this section, a method based on prefix subnetting is introduced to help overcome the update churn problem.

This method takes into account the fact that most workloads are associated with the number of route announcements and withdrawals the controller receives. The figure for a large ISP can be millions per typical day, depending on the number of neighbours it has. Fortunately, route updates can be partitioned based on the prefix it carries. Figure 3.5 represents a portion of prefix tree for IPv4 (0/1 is a short form of prefix 0.0.0.0/1). A route update for a prefix p will be located to one and only one leaf. By distributing leave prefixes to multiple controller instances, the workload per instance will be reduced. This assumes that prefixes are non-overlapping. Redundancy can be achieved by having two or more instances managing the same prefix. This technique works for IPv6 as well.

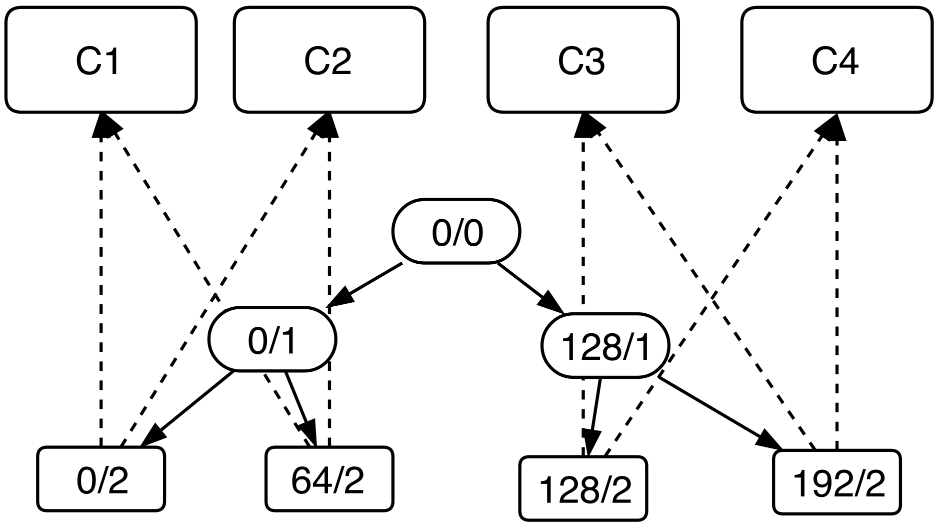


Figure 3.5: Technique for route update distribution to multiple controller instances. An update carries a prefix, e.g. $p \in 0/2$ will be sent to $C1$.

Workloads can vary significantly per prefix. However, it is possible to find a prefix distribution which permits a load balance across instances. Development of a solution for this matter is out of this thesis's scope.

This technique can be easily implemented using, for example an *Update Distributor* (UD) which examines each update's prefix and computes the leave prefix it belongs to to determine the instances to send the update to. The UD can be a centralised or distributed. For instance, if a messaging system such as RabbitMQ is used for communication, the UD can be implemented with the content-based routing feature. With a distributed approach, the FCs are to maintain mapping configuration and to mark updates with a marker corresponding to a prefix group. Then, the update can be distributed via RabbitMQ using simple routing keys.

3.2.3 Event Processing

The *gRCP* architecture is a distributed system where the functionality and workload are split between components. Figure 3.6 shows the process of which a routing event is processed by the system. Events are typically originated by FCs which assign each event with an unique ID throughout the lifetime of the system. Events expire after a period of time, e.g. a few seconds. Expired events are silently discarded.

In this process, an event E_p (e.g. related to a route announcement $r\langle \text{prefix} : p, \text{nexthop} : N, \text{attributes} : A \rangle$ is originated by FC F_x. Its master controller C_x first receives and handles the event. C_x updates the graph database, e.g. by creating a sequence of relationships $(F_x) - [InterEgress] \rightarrow (N) - [Route] \rightarrow (p)$. It, then, re-generates an event with an identical ID of the original one with the content being the newly created sequence. The event is distributed through the pub/sub system to the interested controller

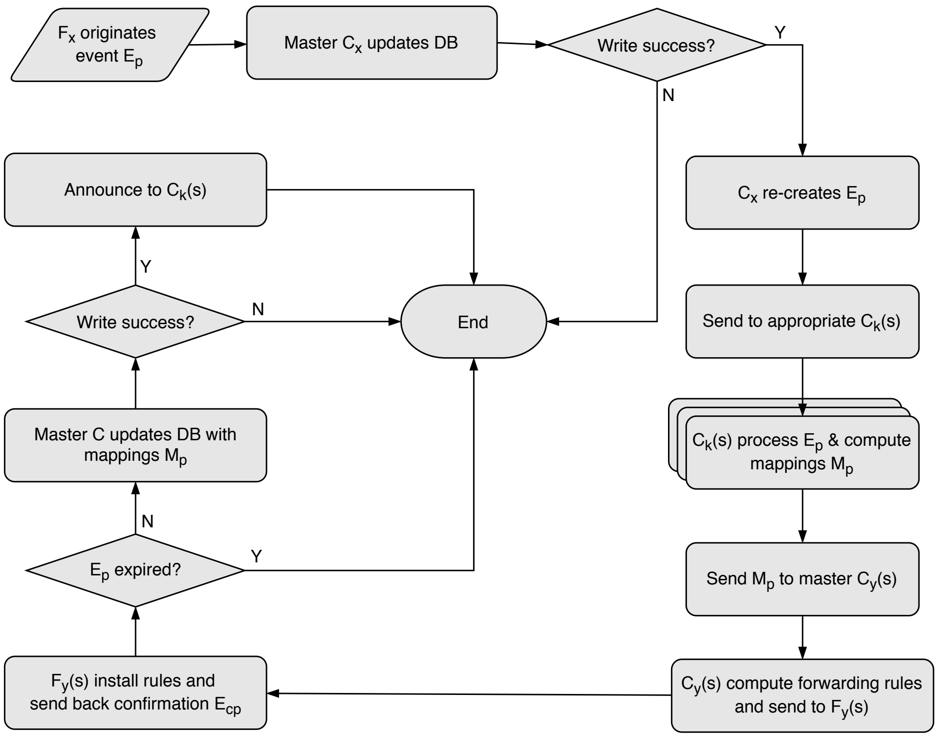

F_x originates event E_p
Master C_x updates DB
Write success?
Y
N
C_x re-creates E_p
Send to appropriate C_k(s)
C_k(s) process E_p & compute mappings M_p
Send M_p to master C_y(s)
C_y(s) compute forwarding rules and send to F_y(s)
F_y(s) install rules and send back confirmation E_cp
E_p expired?
Y
N
Master C updates DB with mappings M_p
Write success?
N
Y
Announce to C_k(s)
End

Figure 3.6: Routing event processing.

instances C_k (those are the instances responsible for a group prefix which p belongs to). C_k(s) compute multiple identical mapping solutions M_p for Forward Controllers F_y(s). M_p are delivered to the responsible controller instances C_y(s) who in turn compute the forwarding rules and program the forward controller F_y(s). Noted that C_y(s) may receive more than one mapping solutions. They simply discards ones that arrive late. F_y(s) inform their masters if the rules have been installed successfully. Upon receiving the confirmation, if the event has not expired, the masters update the database and notify related controllers.

It is the responsibility of control application (i.e. the developer) to handle the loss of confirmation events or half-baked event handling. There are occasions when there exist the disparity between the global network view and the forwarding state. For instance, the master controller may crash during updating the database and after the reception of the confirmation. This disparity may not have impact on the traffic flow. However, applications may take appropriate actions to make sure that the network and global network view are in sync.

Failures of a C_k do not affect the working of the system. If there are multiple C_k instances handling the same event, then the chances that all of them crash during the process are unlikely.

The database can fail too. However the deployment of a cluster of databases can eliminate this single point of failures. Most database systems such as Neo4J support this configuration.

3.3 Evaluation

3.3.1 Language Evaluation

gRCP defines a declarative domain-specific language (DSL) based upon a graph query language to improve the ability of operators to define routing policies programmatically. We evaluate to what extent this aim is achieved using quality characteristics from research into the qualitative assessment of domain-specific languages [85]. Our evaluation is based around these key characteristics: *expressiveness*; *transparency*; and, *usability*.

Expressiveness. The case studies show that three different types of routing policies can be implemented using *gRCP* but expressiveness is more than just functionality. To what degree can a problem-solving strategy can be mapped into a program naturally? In this case path selection is the key strategy and unlike BGP or Morpheus, the language provides a direct mapping because it allows path selection to be expressed directly as a query. It also exposes important domain concepts such as Nodes (or Neighbours) and Paths, allowing direct expression of path selection criteria. Again, BGP and Morpheus require you to restate selection criteria in terms of proxies such as community string or weightings.

Transparency or simplicity. A key feature of a successful DSL is that one does not need to read the whole system to understand a small piece. BGP's implementation of programmable inbound and link congestion resolution requires both BGP configuration directives and a requirement to log into and update multiple routers that are each sent their own version of the configuration file. Understanding the effect of this requires understanding both parts. Morpheus does improve on this by having a single configuration file specifying preferences and mappings in one place. However, the interaction between these is not directly relatable to the selection of paths and requires definitions of special categories used for mapping such as *lat1* and *lat2* in the example. On the other hand, *gRCP* exposes the path computation in a single place with node and path attributes explicitly identified in the query. There is no need to update other parts of the system allowing an operator to read only part of the system in order to understand the overall effect.

Usability. Usability includes characteristics such as appropriateness for the specific applications of the domain, number of activities required to achieve a goal and whether the language has properties that aid producing reliable programs. In terms of appropriateness, the graph database repre-

Metric	***gRCP***	**BGP**	**Morpheus**
Expressiveness	medium	low	high
Readability	easy	hard	easy
Structured programming	flexible	limited	n/a
Error-proneness	low	high	low

Table 3.1: Comparison of rogramming aspects with BGP and Morpheus.

sentation is chosen because it is a natural fit for modelling networks where relationships should be as important in the data model as the data itself. With regard to number of activities, implementation of a new routing function can be done easily in *gRCP* with a few lines of codes. The operator would create an application per function or customer. Typically, a new routing requirement is implemented by simply specifying the *Path.query()* and *Path.order()* statements. In contrast, BGP requires to define policy configuration for different scenarios and to switch between configurations at the right time. On the other hand Morpheus's configuration is much more minimal than either *gRCP* or BGP but it does not support other routing requirements. Finally, reliability is supported by *gRCP* because of the advantage of error checking in the underlying graph query language and the ability to test the routing policy interactively against the database before applying the policy. In contrast, neither BGP nor Morpheus allow testing of path selections prior to deployment on the actual system.

3.3.2 Performance Evaluation

The performance evaluation focuses on routing convergence because it is the most important performance indicator for a routing design. The convergence latency is commonly measured by the elapsed time since a routing event occurs until all routers in the network agree on a single routing solution. A routing event can be a link or router failure or a routing update caused by a policy change. In the context of interdomain routing, routing updates (i.e. BGP route announcement or withdrawal) are more frequent and are in great numbers than other events, therefore the convergence latency is measured against the update types.

Metrics

Typically, convergence latency T_c is composed of following components:

$$T_c = T_{detect} + T_{distribute} + T_{compute} + T_{install} \tag{3.1}$$

where T_{detect} denotes time taken to discover an event by the first router; $T_{distribute}$ denotes time required to distribute the event to all other routers; $T_{compute}$ denotes time taken to compute a new routing solution; and $T_{install}$ is time required to install rules to the forwarding table.

In a centralised control system like the one we propose a single controller performs route computation for all routers, thus $T_{compute}$ is measured by the time the controller receives the event and until a solution is computed. $T_{distribute}$ is identical to a conventional BGP network. $T_{install}$ is likely to be higher than traditional routers since it involves conversion of routing decision into OpenFlow rules and communication of rules to hardware switches (and different vendors perform differently). This experiment evaluates $T_{compute}$, see Chapter 4 for evaluation on $T_{install}$.

Experiment design

The controller relies on the operations on a graph database for computing routing. Therefore, the evaluation is designed to get a deep insight into how the scale of the network affects the controller performance and how well query-based route computation performs in different configurations. The hypothesis is that query-based computation can scale to very large ISPs with hundreds of routers, thousands of neighbours (i.e. BGP peers) and hundreds of thousands of destination networks (i.e. prefixes) with an insignificant additional latency compared to that of a conventional BGP network. The faster the network converges, the better quality of service is. A measurement study of two large ISPs concludes that most events converge in between 1 second to 30 seconds [86].

The evaluation focuses on $T_{compute}$ and measures it as the function of scales (number of routers, neighbours and prefixes), and types of updates (route announcement/withdrawal). Experiments were done on a single computer with Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz, 8GB of memory and an 512GB SSD. A graph database used is Neo4J version 3.4 running on a docker container with all default configuration. The measurement script which executes the controller APIs and models running on the host machine (thus, $T_{distribute}$ latency is assumed to be 0 or insignificant. The controller interacts with the database via an official Neo4J driver written in Python (i.e. bolt driver over TCP). Each experiment is repeated 100 times with an empty database to eliminate the caching effect of Neo4J. $T_{compute}$ does not take into account the time needed to populate the database. $T_{compute}$ is measured as time between an event occurrence and when the query is completed.

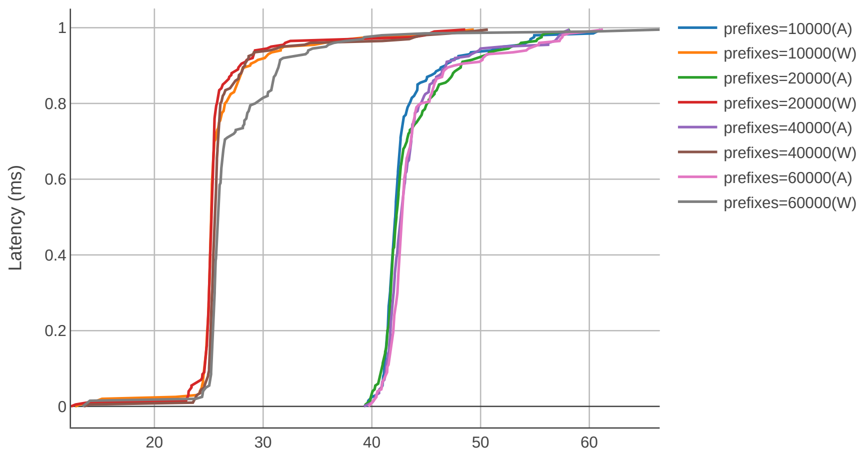


Figure 3.7: Path computation latency as a function of #prefixes and update types. Path query is run for a single router with 10 neighbors. A and W stand for route announcement and withdrawal respectively.

For each test, the router topology is generated randomly and a subset of routers (the number is randomly generated from uniform distribution between 0 and 10) are connected to all neighbours. Prefixes are generated randomly such that each can have at most 5 different paths. For a given experiment (blackbox/whitebox) and a topology, two tests are executed (100 times each): route computation in response to a routing announcement and a routing withdrawal. For all topologies, the route computation is implemented using the following query, which computes routes to a specific prefix for all routers using the same policy (prefer routes by local preference, then AS path length, and MED).

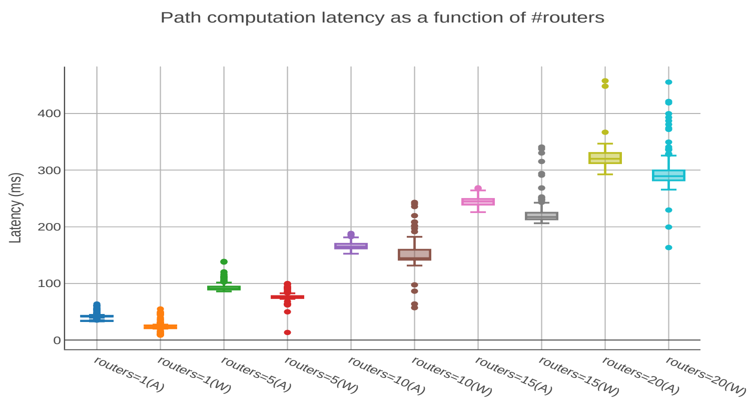
Results: The solution scales

In Figure 3.7, we observe that the route withdrawal test performs better than the route announcement test (the path query was run for a single router and each route update affects a single prefix). Specifically, 90% of route withdrawal updates converge less than 28 ms compared to 47ms of announcements in all configurations. Moreover, the size of the topology (i.e. number of prefixes exists in the network) has minimal impact on performance.

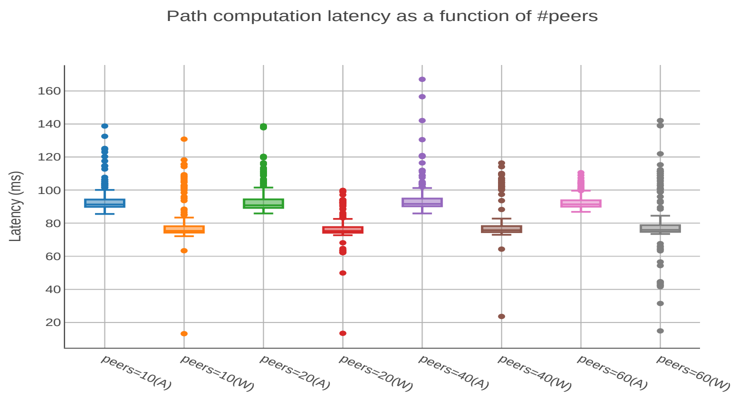
The Figure 3.8a shows that latency increases linearly with the number of routers in the network but at a slightly slower pace. This is anticipated as we simply repeat the query for each router. Some routers may use the computed path of the other from the cache of Neo4J, explaining why the latency increases more slowly than the topology size.

An interesting result shown in Figure 3.8b is that increasing the number of neighbours (i.e. BGP peers) has no impact on performance. Overall, the result shows that the solution can scale well to a very large network. The major scalability concern is the number of routers that the controller has to computes routes for each route update. Since the route computation is basically a query from the graph database (an example of query is shown below), we need to specify the router (either neighbouring router or a border router) as the start of the path and a prefix as the end. That means a single query cannot be used to compute routes for all routers. Since each query is run sequentially in different transactions, it increases the latency. However, these queries are unrelated, it is possible to scale horizontally by splitting the task for multiple controllers.

To gain better understanding of what contributes to the high latency,



(a) Path computation latency as a function of #routers



(b) Path computation latency as a function of #peers

Figure 3.8: Computation latency on different network sizes (A=announcement, W=withdrawal).

we decompose the $T_{compute}$ (according to the implementation) and evaluate performance of each components. When the controller receives an update, it first updates the database (by running one or more read and write queries in a single database transaction), then an event is created and distributed internally to applications. An application runs a read query to compute paths

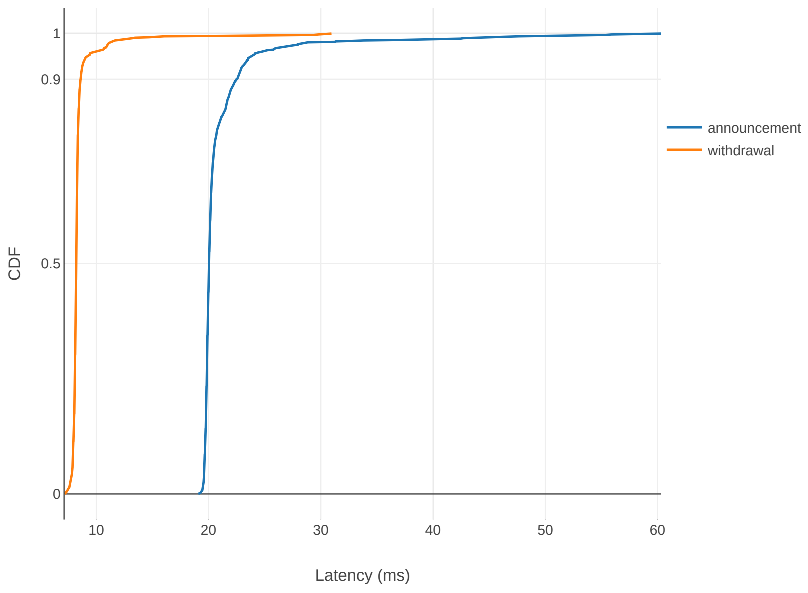


Figure 3.9: Route update latency with regard to types of updates.

and selects the desired one (i.e. the query(), sort() and fetch() methods are run in another transaction). Thus, $T_{compute} = T_{update} + T_{path}$, where T_{update} is time to update the database and T_{path} is the time to query for paths. For example, a route announcement requires a query for the neighbor and prefix (read) and update the relationship between them (write) (refer to the model design for better understanding). If necessary, a Prefix node has to be created (require one more write). For a route withdrawal, one read and one write is needed. Noted that Neo4J performs consistency checking for every write before writing to disk. Meanwhile, reads can be fetched from cache or disk without consistency checking.

The Figure 3.9 shows results for performing database update upon the recipient of a BGP update. Route announcement performance is 3 times slower than that of a route withdrawal. The announcement latency can

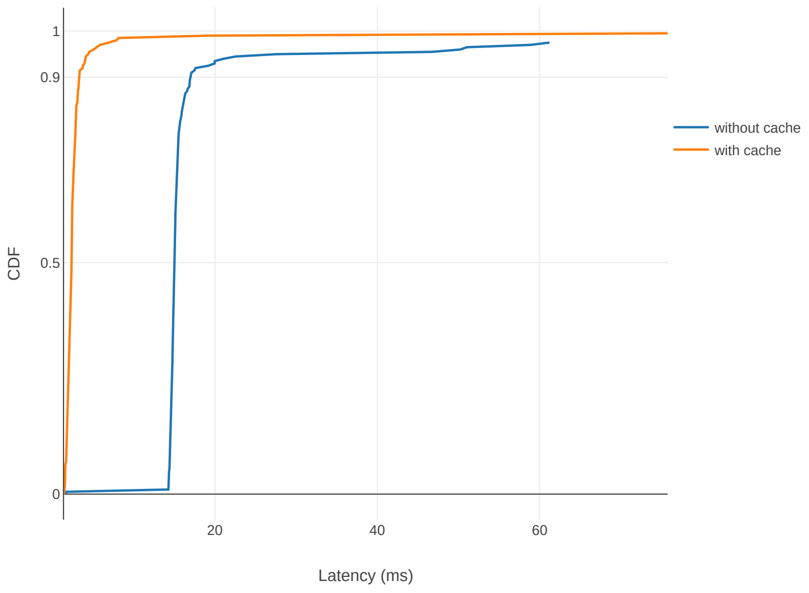


Figure 3.10: Performance can be significantly improved by Neo4J caching.

potentially be reduced by having Prefix nodes created in advance, this reduces the number of writes to the database. This is possible since most popular prefixes in the routing are known beforehand. Moreover, the performance can be improved with caching technique in Neo4J, as shown in Figure 3.10. Path query which has been cached performs 4 times faster than non-cached queries.

3.4 Summary

This chapter provided an overview of a design of a routing control platform to transit ISPs. The design utilises graph models and graph database to represent network routing and graph query language for computing routes as an alternative to the conventional BGP's pair-wise comparison. As shown in the evaluation, *gRCP* improves routing flexibility and expressiveness while introducing a small penalty for routing convergence. Chapter 6 will provide more analysis of *gRCP*'s capabilities in comparison with the traditional approach.

Chapter 4

Design and Implementation of fBGP

This chapter presents the design, implementation and evaluation of *fBGP*, a SDN controller that leverages capabilities of OpenFlow to enable routing flexibility in conventional BGP routing. *fBGP* equipes the border routers in ISPs with the multipath routing capability.

There are four main sections in this chapter: (1) Section 4.1 provides an overview of the main elements of the *fBGP* controller; (2) Section 4.2 provides an overview of *fBGP*'s implementation; (3) Section 4.3 describes a deployment of *fBGP* controllers in a physical testbed representing a small ISP POP; (4) the final section 4.4 evaluates *fBGP* with regards to performance and functionality.

4.1 Design of fBGP

This section presents the design of *fBGP* routers, focusing on the multipath routing capability. The capability improves path diversity in transit ISPs and allows multiple routing paths to be used simultaneously. Multipath capability can be used for better traffic engineering and resolving conflicting routing policies. The quality of experience and throughput can also be improved by forwarding traffic over multiple paths (e.g. high bandwidth paths used for bulk data transfer applications whereas low latency paths are for real-time traffic) [87, 88].

For demonstration, let take a look at an example ISP shown in Figure 4.1. The ISP network learns five different paths (depicted by numbered circles) for a particular prefix. In a conventional network, the router $ASBR_1$ selects one path out of the five for all traffic from the customers according to the assigned local preference of the paths (and the IGP distance to the exit points, depicted by numbered squares). For instance, both $ASBR_1$ and $ASBR_2$ will select path 3 if it has the highest local preference. If the local preferences are equal, $ASBR_1$ selects either path 1 or 2 while $ASBR_2$ chooses $ASBR_1$ as its exit point. Thus, routing policy such as "C1 to use path 1, C2 path 3 and C3 path 4" is difficult to realise in the current network. Techniques such as Equal-Cost Multiple Path (ECMP) can be used for load balancing across multiple paths but they do not allow flexible mapping of traffic to a path.

In this conventional network, two workarounds namely PBR and VRF may be employed for implementing the multipath capability. PBR allows routing to be based on various packet headers rather than just the destination address. Therefore, appropriate PBR rules can be defined and applied

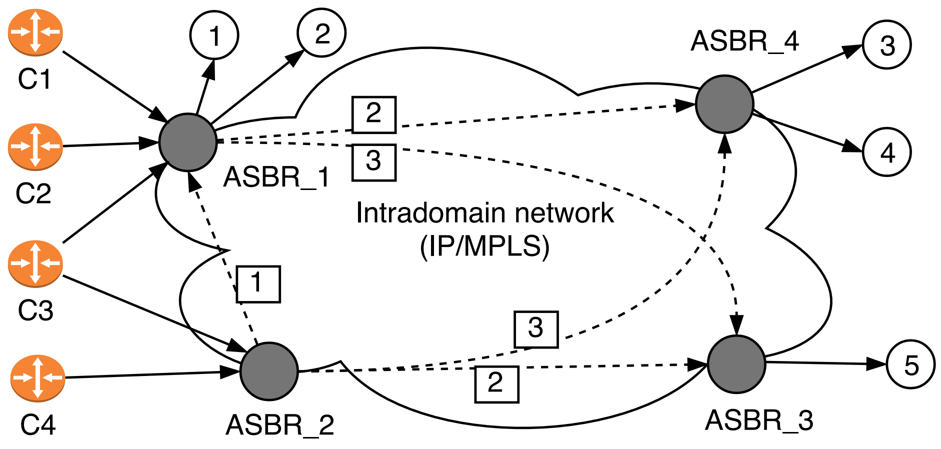


Figure 4.1: The case for multi-path BGP policy. The numbered circles represent candidate paths to a prefix p. Numbered squares are the IGP distance between $ASBRs$. In the current network, policy is prefix-based making it difficult to relocate customer traffic over different paths.

on both the ingress and egress border routers so that customer's traffic is forwarded to a different path rather than one chosen by BGP. However, this incurs significant management overhead because PBR rules need to be kept in sync with BGP routing. As BGP routes are withdrawn PBR rules will need to be updated or deleted quickly to avoid long black-holing.

VRF is a technology that allows additional routing tables to coexist alongside the default table (i.e. the global routing table). Therefore, a VRF can be defined for a customer to handle its traffic differently from the default routing. Since VRFs are isolated, each must hold all the routes otherwise the customers can be black-holed for some prefixes, resulting in a huge duplication of routes. This can be addressed by the route leaking technique which allows VRFs to share routes. However, implementing routing policy with VRF and route leaking is troublesome; it needs to determine the shared routes and define the configuration for route import and export between VRFs [1].

fBGP is designed to overcome the challenges facing PBR and VRF. It is based on OpenFlow to allow conventional routing and flexible forwarding policy by allowing utilisation of all available paths concurrently in the same platform. One of the design goals of *fBGP* is to ensure backward-compatibility with the existing network. This is accomplished by extending BGP functionality by utilising the flexibility of OpenFlow, rather than re-designing the BGP protocol. One of the design challenges is routing scalability. This will be addressed by a design of a distributed forwarding table. The following section describes *fBGP*'s architecture.

4.1.1 High-level Architecture

The design of *fBGP* adopts a hierarchical and modular architecture. It consists of three different layers: *Routing*, *Forwarding* and *Datapath* as shown in Figure 4.2. The modularisation is achieved by leveraging three distinct functionalities of a router: routing functionality, forwarding functionality and the forwarding hardware (i.e. Ternary Content Addressable Memory (TCAM)).

The Routing layer is comprised of BGP routing stack. The BGP stack runs in a separate process (and probably in separate hardware) from the Flow controllers (FCs) and is responsible for the BGP finite state machine, managing BGP speaker peers, performing route selection and executing routing policy. BGP stack keeps track of the peers and routes but is not necessarily concerned with how peers are connected to the router. BGP sessions are

[1]https://cumulusnetworks.com/blog/vrf-route-leaking/

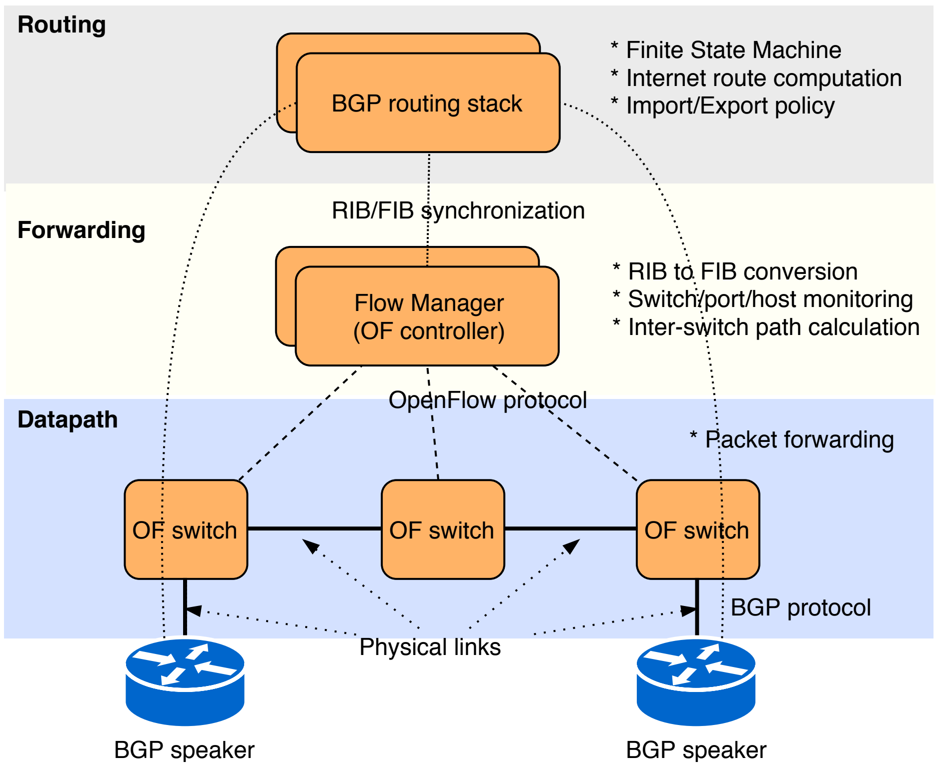


Figure 4.2: An OpenFlow-based architecture for interdomain routing in ISP. BGP routers are replaced by OpenFlow switches and routing is centrally performed by BGP stack which replaces route reflectors.

established directly between the BGP stack and the external speakers. The BGP stack communicates its decisions to the Flow Manager which translates them into appropriate forwarding rules for the switches.

In the Forwarding layer there are OpenFlow controllers responsible for managing switches and abstracting the distributed switch as a single forwarding table. These controllers manage connectivity between switches and employ routing algorithms for path computation within the distributed switch, oblivious to the BGP controller. Only important events, such as a failure which disconnects traffic flows from one neighbour to another, should be notified to the Routing layer.

The Physical layer is composed of programmable OpenFlow switches. These switches are interconnected via high speed links. Some ports are used to connect to neighbours' routers (called external ports), while others are for interconnecting switches (called internal ports).

This architecture offers several advantages including scalability and flexibility as compared to the conventional BGP network. For example, it is now easier to scale the system to support more BGP peers since the Routing stack is running on separate hardware and is independent from the data plane construction. As more capacity (e.g. bandwidth or number of ports) is required, new switches can be added to the data plane without modification to the Routing stack. The reliability in the data plane can be improved by for example Link Aggregation, or connecting a customer to multiple switches. The Routing stack simply sees a single session with the customer. The operator is free to implement any possible mechanism to load balance or switch traffic between links.

4.1.2 Multipath Routing Capability

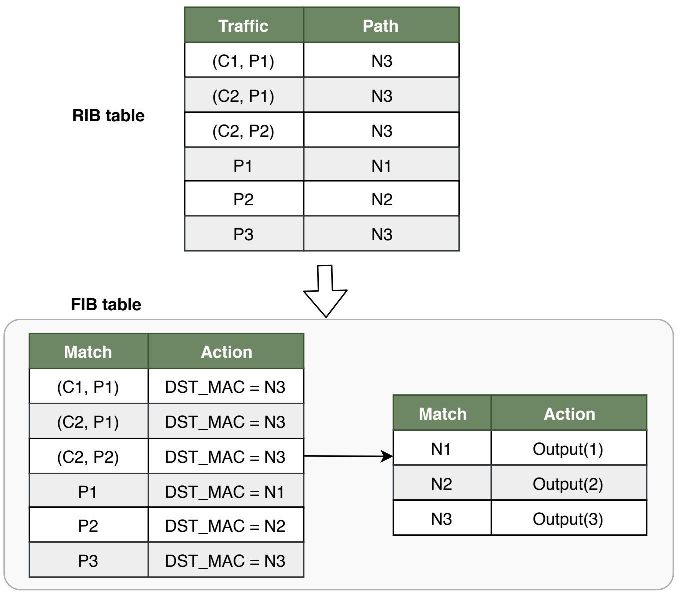
In conventional routers, Routing Information Base (RIB) entries are translated into forwarding rules and installed into the FIB table. In an OpenFlow switch, these FIB entries are OpenFlow forwarding rules (hereby called FIB rules for short). A FIB rule matches on the destination prefix and has actions to modify packets appropriately before outputting to the port connected to the next-hop.

In implementing the FIB table, the multi-table support in OpenFlow version 1.3 onward is utilised to decompose the FIB table into two sequential tables. The first table contains rules that match on the prefix and have necessary modification actions. Modified packets are forwarded to the second table which contains rules that match on the destination Media Access Control (MAC) address and send packets to the next-hop's port. This hierarchical FIB table simplifies rule management in the controller. If the next-hop moves to a new port, only a single rule needs to be updated. Moreover, FIB rules (which change frequently in interdomain routing) can be added, deleted or modified independently from the resolution of the next-hop into MAC address.

In order to support flexible routing policy, an extension of the FIB table to allow matching on finer-granularity of packet headers is required (e.g. match on the source IP or MAC address). For instance, to support customer-specific routing, the FIB table is extended to support matching on customer's identity along with the destination prefix. A customer can identified by its MAC address or the OpenFlow *inport*. Thus, a normal FIB rule e.g. $match(P) \rightarrow forward(N)$ will be transformed to a multipath FIB rule $match(C, P) \rightarrow forward(N)$, where C can be the source MAC address. These multipath rules could be installed to the same FIB table with the normal rules with higher priority. This, however, would end up in a significantly large table with $N \times D + P$ entries, where N is the number of customers having multipath rules to D of all P prefixes (see an example in Figure 4.3a).

By leveraging the number of customers which is typically larger than the number of possible paths and several customers may share the same path, the size of the FIB table can be reduced as per following technique.

Each path is assigned a unique tag and (virtual) MAC address (called

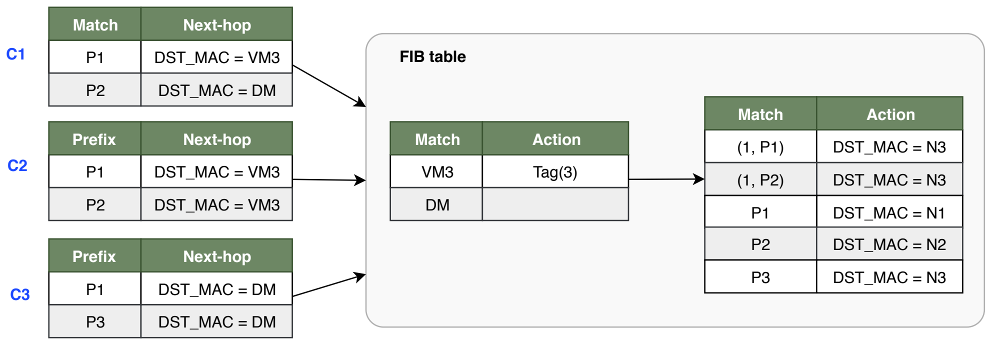


Traffic	Path
(C1, P1)	N3
(C2, P1)	N3
(C2, P2)	N3
P1	N1
P2	N2
P3	N3

Match	Action
(C1, P1)	DST_MAC = N3
(C2, P1)	DST_MAC = N3
(C2, P2)	DST_MAC = N3
P1	DST_MAC = N1
P2	DST_MAC = N2
P3	DST_MAC = N3

Match	Action
N1	Output(1)
N2	Output(2)
N3	Output(3)

(a) Rules are flattened and installed into a single table.



Match	Next-hop
P1	DST_MAC = VM3
P2	DST_MAC = DM

Prefix	Next-hop
P1	DST_MAC = VM3
P2	DST_MAC = VM3

Prefix	Next-hop
P1	DST_MAC = DM
P3	DST_MAC = DM

Match	Action
VM3	Tag(3)
DM	

Match	Action
(1, P1)	DST_MAC = N3
(1, P2)	DST_MAC = N3
P1	DST_MAC = N1
P2	DST_MAC = N2
P3	DST_MAC = N3

(b) Rules are decomposed and installed into multiple tables (Priorities are not shown due to space).

Figure 4.3: Two approaches to flexible forwarding policy using OpenFlow. In (a) RIB entries are translated into PBR-like rules. In (b) the FIB rules are decomposed into 3 different tables.

VM for short). The first table matches the prefix and set the destination MAC to the *VM*. The second table matches on *VM* and assign the tag to packets. Default MAC address does not have associated tag. The third table matches packets on both the prefix and the tag and set the destination MAC to that of the next-hop before sending to the *Output* table (see Figure 4.3 for an example).

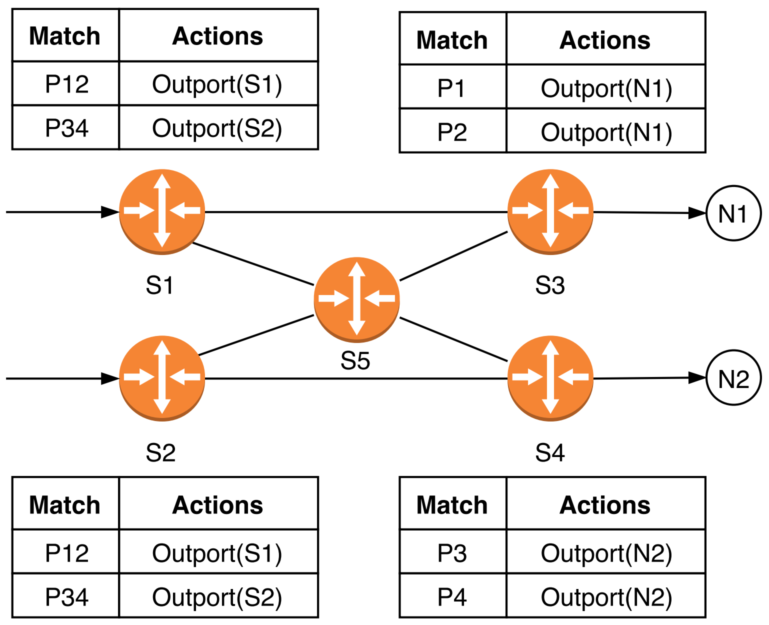
This results in more rules in total but less rules in each table. However, since the first table already exists in the customers' router, *fBGP* does not need to store it. To form a complete forwarding pipeline, *fBGP* will need to alter the first table stored in customers. This can be done by tweaking the next-hop resolution process. As a customer router sends a resolution request for a next-hop, *fBGP* examines the mapping table and returns appropriate *VM* which identifies a path. Thus, each customer would perform the packet tagging on behalf of *fBGP*.

This technique results in a smaller number of rules, equal to $|N| + |K| \times |D| + |P|$, where K is the set of available paths ($K << N$). In a typical setting, an ISP can have tens of paths per prefix which is much smaller than the set S. Moreover, among those paths, a smaller set may be enough to satisfy a large number of customers.

4.1.3 Distributed Forwarding Plane

An approach to ensure routing scalability is to enable horizontal scaling by implementing the FIB table on top of a distributed forwarding plane which is composed of multiple OpenFlow switches. This section describes how the implementation can be achieved.

In the forwarding plane, egress switches are directly connected to the next-hops whereas ingress switches are connected to the incoming traffic. Egress and ingress switches are directly connected or interconnected via transit switches. The FIB table then can be decomposed and distributed



Match	Actions
P12	Outport(S1)
P34	Outport(S2)

Match	Actions
P1	Outport(N1)
P2	Outport(N1)

Match	Actions
P12	Outport(S1)
P34	Outport(S2)

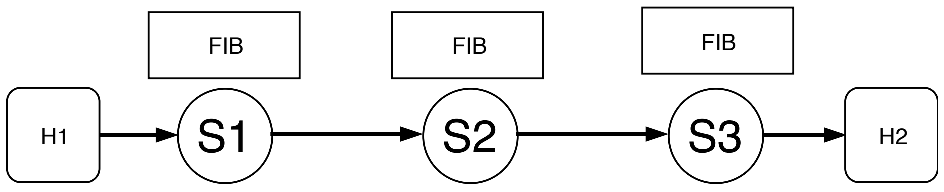
Match	Actions
P3	Outport(N2)
P4	Outport(N2)

Figure 4.4: Translation of RIB into *dFIB* table of a given topology.

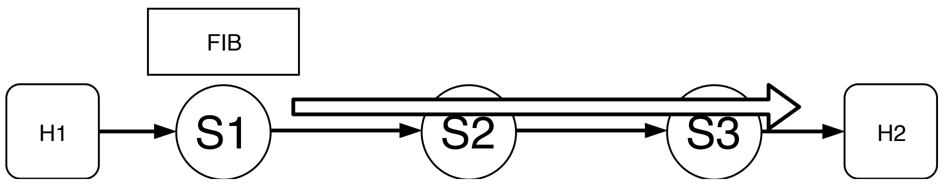
to switches by a simple approach demonstrated in Figure 4.4 as follows.

This approach replicates the FIB at all switches with the next-hop being the next switch along the path to the external next-hop, as illustrated in Figure 4.5a. This result in unnecessary redundant rules. To address this, source routing technique is used. Source routing (as shown in Figure 4.5b) requires a single FIB entry and tunnel rules to drive packets to the desired switches. Thus, only ingress switches need to store the FIB table. This technique eliminates the unnecessary duplication but does not utilise efficiently the available space.

To improve space utilisation efficiency, FIB rules can be decomposed and installed to multiple switches. One decomposition technique is to base on the next-hop. For instance, all rules with next-hops $N1$ will be installed to the egress switch $S1$ since $N1$ is connected to $S1$. Incoming traffic to an ingress



(a) Hop-by-hop forwarding. FIB entries are installed at every hop along the path.



(b) FIB entries are installed at the ingress node only. Subsequent nodes tunnel packets to the egress node.

Figure 4.5: Two approaches to establishing end-to-end forwarding path: Hop-by-hop forwarding and source-routing

switch will need to be forwarded to the correct egress switch. This can be achieved by computing aggregate FIB rules by grouping prefixes that are numerically aggregated, for instance two prefixes 1.0.0.0/24 and 1.0.0.1/24 can be aggregated to 1.0.0.0/23. The aggregate rules are to be installed in the ingress switches to direct packets to the right egress switch where packets are examined by the actual FIB rules. This process spreads FIB rules to multiple switches but the unbalanced distribution may still exist because some switches have to hold more rules than the others (e.g. aggregate rules are much less in number compared to the actual rules).

The unbalanced distribution causes inefficient use of FIB space in switches. This unbalance can be overcome by migrating some FIB rules of egress switches to the ingress and transit switches. The rule migration is decoupled from the routing and is to be performed by the FC controller. The determination of which FIB rules to be installed on which switch in order to achieve efficient distribution will be investigated in Chapter 5.

4.2 Implementation of fBGP

This section provides an overview of the implementation of a fully functional prototype of *fBGP*. The prototype was implemented fully in Python. It implements the most critical functionalities of the design including the conventional BGP selection process, the ability to manipulate the path mappings and the interaction with the OpenFlow data path. Its main components and interactions with other parts of the system are shown in Figure 4.6.

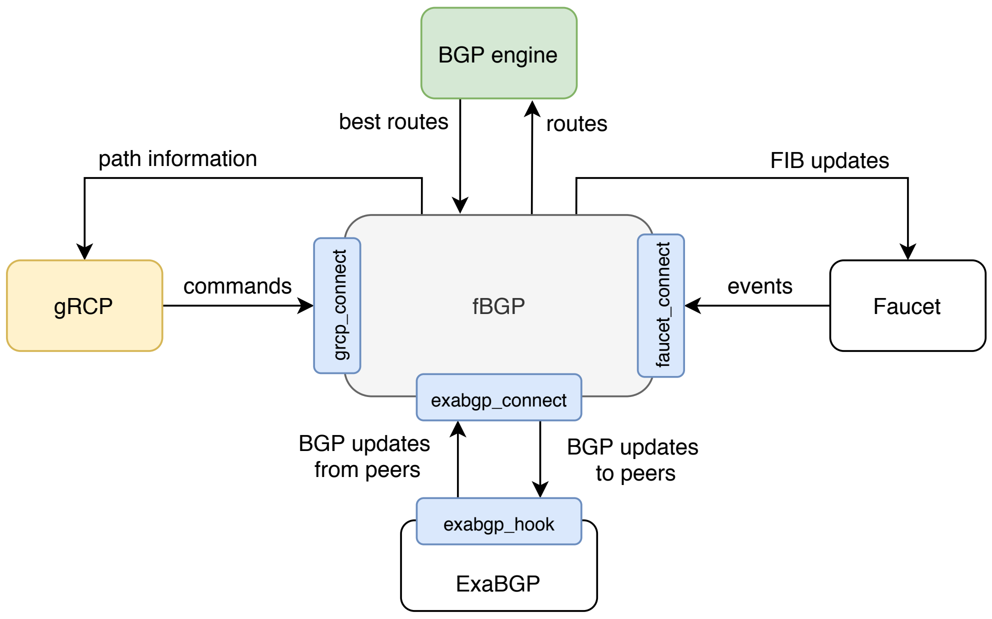


Figure 4.6: fBGP components and interactions within the system. The shaded boxes were implemented in Python as part of the thesis. The others are existing open-source codes.

ExaBGP [2] is used to learn and exchange routing updates with other BGP routers. ExaBGP known as BGP Swiss army knife of networking, provides a simple and reliable platform to develop BGP applications. ExaBGP maintains the BGP sessions with the BGP peers and handles the BGP state machine. The *exabgp_connect* module is used for communicating BGP updates with ExaBGP. New routes received from ExaBGP are passed to the *BGP engine* module for updating the routing table and selecting the best routes.

The *fBGP* prototype relies on Faucet controller [3] for implementing the forwarding plane and the FIB table. Faucet is an open-source OpenFlow controller that provides L2 and L3 routing functionality. Faucet runs on Ryu controller. Faucet is selected because it is written in Python, very lightweight, well documented and easy to use. However, it is possible to port *fBGP* to other controllers such as ONOS or Open Daylight.

The *fBGP* controller similar to Faucet is built as a Ryu application. It runs alongside the Faucet on the same Ryu controller. The module *faucet_connect* interacts with Faucet via Faucet APIs. At the time of the prototype development, the Faucet APIs required for updating the FIB table were implemented as part of the prototyping. In addition, several additional APIs and extensions were implemented in Faucet since it did not currently have the multipath routing capability.

Within *fBGP* and *gRCP*, pathIDs are used to identify interdomain paths in the network. Each pathID is associated with a virtual gateway (VIP) in Faucet. When a path is advertised to a peer, the VIP is used as its next-hop.

[2]https://github.com/Exa-Networks/exabgp
[3]https://faucet.nz/

As the peer resolves the next-hop an associated MAC address is returned. Faucet installs a rule into the mapping table. Packets from the peer that uses the path will be added with an OpenFlow metadata (i.e. pathID). The packets are then pipe-lined through the FIB table. Faucet's FIB table was extended to match on the metadata and the destination prefix. OpenFlow metadata differentiates rules and has an important attribute. If the rule with the metadata does not exist, the (default) FIB rule without metadata is used. This allows continuous forwarding when failures occur and while the *gRCP* or *fBGP* controller are computing alternative path.

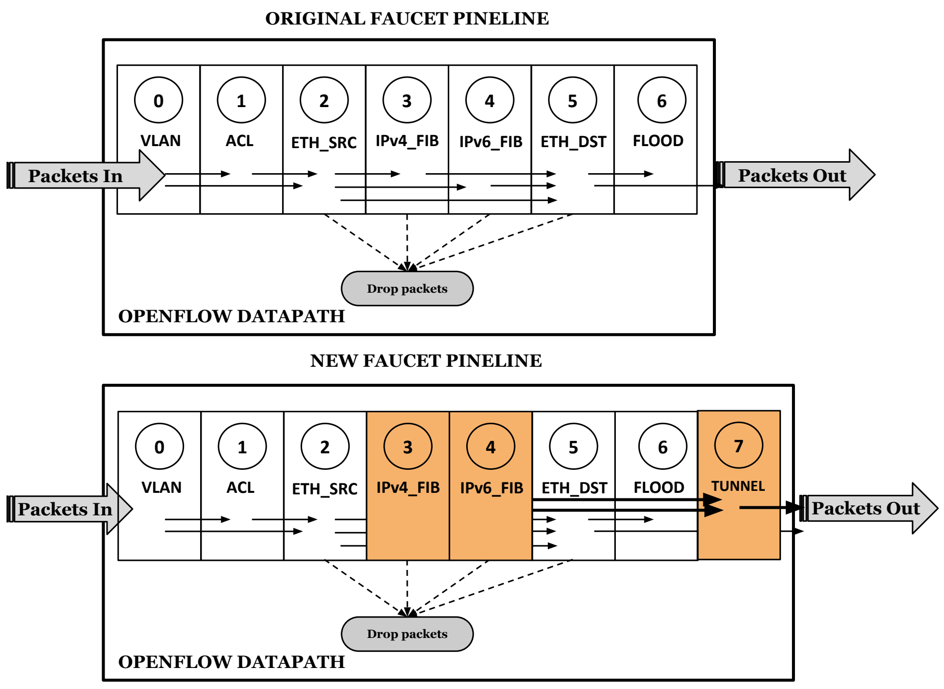


Figure 4.7: Forwarding pipeline of the original Faucet (top) and multipath routing-enabled Faucet (bottom). The IPv4_FIB and IPv6_FIB were extended with Metadata to support multipath.

The modified Faucet's forwarding pipeline is shown in Figure 4.7. IP forwarding entries are stored in two FIB tables: *IPV4_FIB* and *IPV6_FIB* table for IPv4 and IPv6 prefixes respectively. Traffic classifier is implemented by adding rules that match MAC addresses associated with the virtual IP addresses in the *ETH_SRC* table. The *ETH_SRC* table keeps track of learned hosts and determines whether traffic will be handled by L2 or L3 routing. For tunnelling traffic between border routers, a *tunnel* table is added. In the new pipeline, the FIB tables determine whether packets will be forwarded straight to output port or are handled by the *tunnel* table. The *tunnel* table encapsulates packets in Multiprotocol Label Switching (MPLS) headers which carry the path identifier and tunnelling information used by the core network to forward the packets to the correct egress border router.

The *fBGP* controller updates the *gRCP* controller with path information as it learns from the neighbours. The *gRCP* controller can send (mapping) commands that override the decisions made by *fBGP*. Two types of commands are implemented: the *Add* commands that add a path mapping between *fBGP* itself or a peer and a route for a particular prefix, the *Delete* commands that delete existing mappings. The module *grcp_connect* that handles the communication between *fBGP* and *gRCP* is implemented using the Python Twisted library [4] for event-driven network programming. The prototype is implemented with mechanism to fallback to the local routing decision when the connection to *gRCP* is lost.

[4]https://twistedmatrix.com/trac/

4.3 Testbed Deployment

A physical testbed was deployed for the purpose of this research. The testbed is composed of three hardware switches from Allied-Telesis, one *gRCP* controller and three *fBGP* controller representing three ISP POPs. It is fully functional and compatible to BGP (i.e. Quagga).

4.3.1 Physical Topology

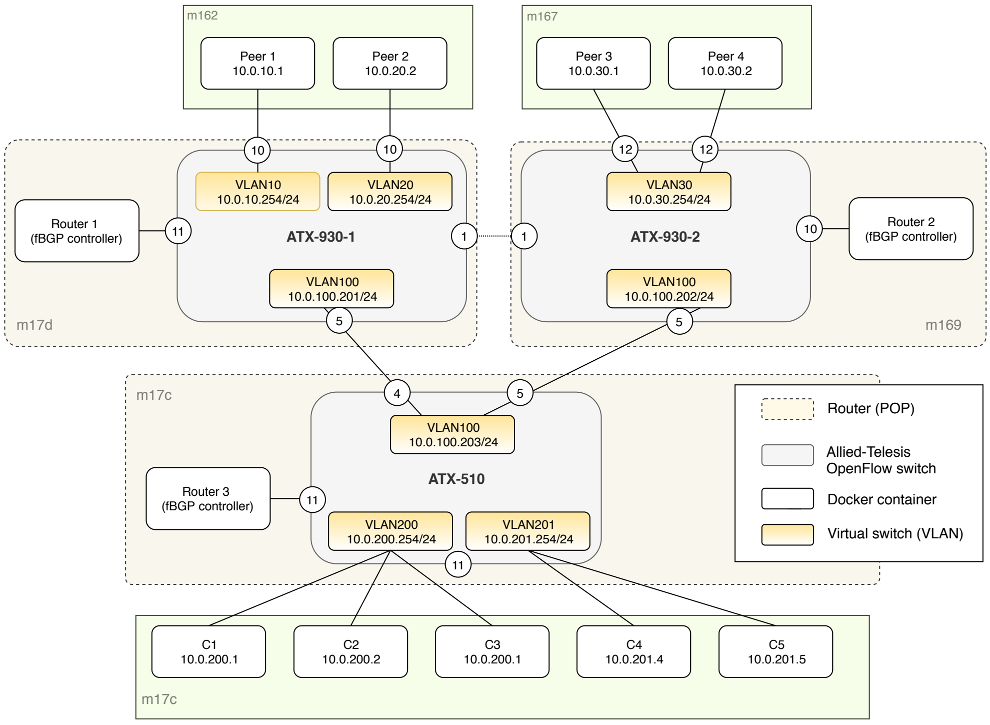


Figure 4.8: Physical topology of the testbed. It is made up of three Allied-Telesis switches and six Linux PCs hosting controllers and quagga routers.

A physical SDN testebed, called SDN@VUW, has been deployed for evaluation purposes of the research. The topology is shown in Figure 4.8.

The SDN@VUW testbed uses three physical switches from Allied-Telesis, i.e. series ATX-930 and ATX-510. Those are high-performance 24-gigabit port switches designed for enterprises. They can run in traditional mode or in OpenFlow v1.3. ATX-930 and ATX-510 switches have been officially verified to be supported by the Faucet controller. A number of Linux computers are also used to simulate the peer routers and to host the controllers. They run Ubuntu 16.04, having Intel Core i7-4790 @ 3.6 GHZ CPU and 8 GB of memory.

Docker containers are used to deploy the peer/customer routers. A docker container is used for each peer. Quagga is used for BGP routing. A physical link between a computer and a switch is configured in trunk mode. Docker networking MACVLAN is used to simulate separation between peers.

In the physical switches, VLANs (configured and managed by Faucet controllers) are used to simulate IP routed interfaces. From the peers' perspective, each VLAN is a logical IP interface.

The SDN@VUW testbed is managed and automated totally by Ansible. A separate management network (using a Cisco 2960 switch) (not shown) is used for communication between the controllers and the controllers and the switches. The control plane packets between a *fBGP* controller and a peer are traversing completely in the data plane.

4.3.2 Logical Topology

The SDN@VUW logical topology is shown in Figure 4.9. Three routers Router1, Router2 and Router3, corresponding to three hardware switches, represent border routers of the three ISP POPs. A route reflector (quagga) is used to facilitate the exchange of routes between border routers.

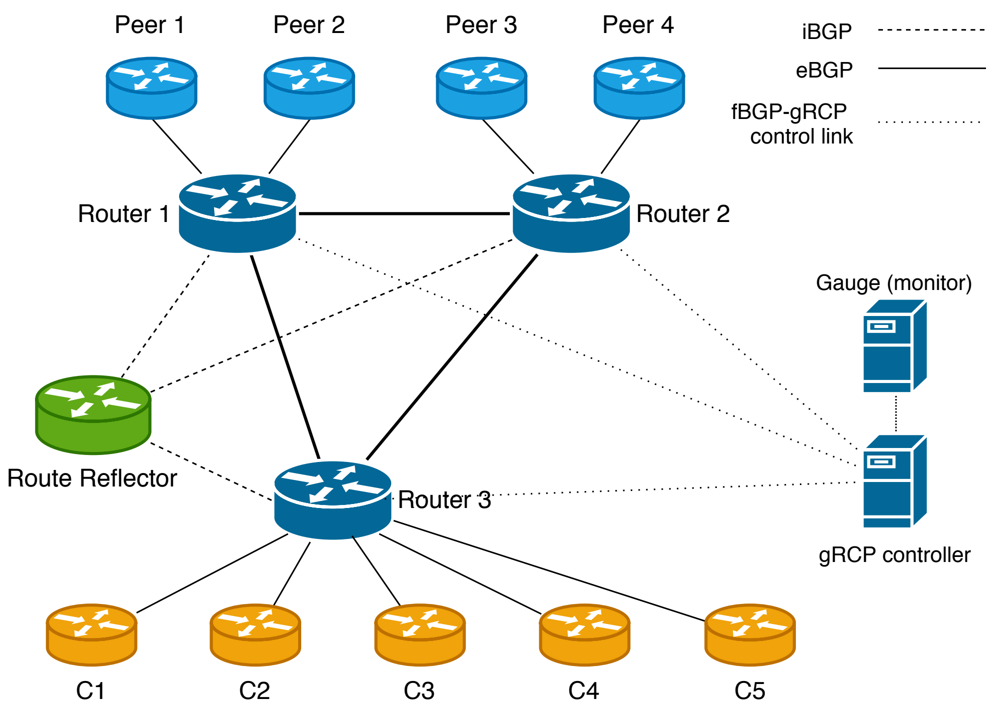


Figure 4.9: Logical topology of the testbed. It represents a small ISP with three border routers.

For evaluation purposes, peer routers are configured to advertise four prefixes $1.0.0.0/24, 2.0.0.0/24, 3.0.0.0/24$ and $4.0.0.0/24$. Policy routing is applied such that prefix 1.0.0.0/24 is preferable via Peer 1 and so on. Thus, by the default policy routing, Customer C1 to C5 would use Peer 1 to reach to prefix 1.0.0.0/24, Peer 2 for 2.0.0.0/24 and so on.

The *gRCP* controller is deployed with a simple application which switches C2 traffic to 2.0.0.0/24 to Peer 2 if the link Router1-Peer1 is 60% saturation. It also splits C4 and C5 traffic to prefix 4.0.0.0/24 to Peer2 and Peer4 (preferable) or Peer3 and Peer4. If Peer3 is not available for 4.0.0.0/24, then the default routing is used.

Currently, the monitoring (by the Gauge controller of Faucet) is configured to port monitoring only. Faucet does support flow monitoring but it does not support exporting the metrics to Prometheus database which *gRCP* relies on the collect network statistics.

A traffic generation tool, iperf3 [5] is used to generate traffic from a customer router to a prefix. To simulate reachability, a logical sub-interface is configured on each Peer router and ping and traceroute utilities are used to validate the connectivity.

A few issues were encountered during the deployment, that are not described in the official manual from Allied-Telesis. When the OpenFlow controller programs rules that specifies VLAN for a port but the VLAN is not currently configured on the switch, it can cause unexpected behaviour. The traffic traverses that port experience extreme delay and packet loss. Most packets are lost, except the first packets. The same behaviour is observed if the VLAN is configured but at the same time the port is assigned to the VLAN as if the switch is in normal mode.

4.4 Evaluation

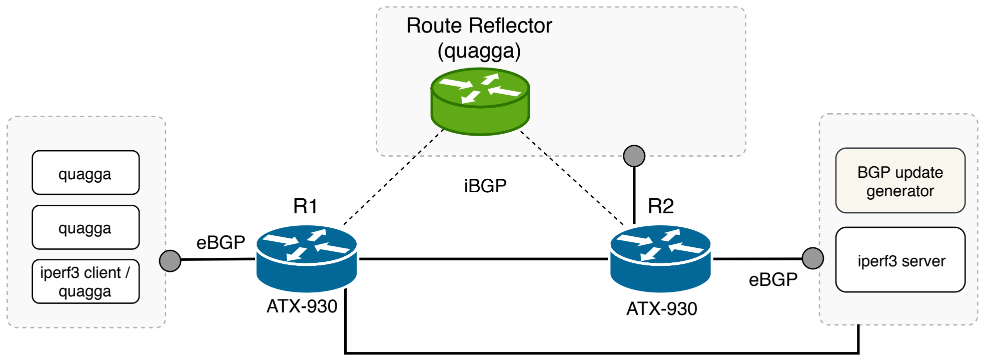
Two sets of experiments were conducted. The first one focuses on the convergence performance in the control plane. The convergence performance is measured as the latency between the arrival of an BGP update arriving at the controller and the completion of installing the corresponding FIB rules on the switches. This evaluates the impact of using OpenFlow on implementing the BGP functionality.

The second set of experiments evaluates the forwarding performance of

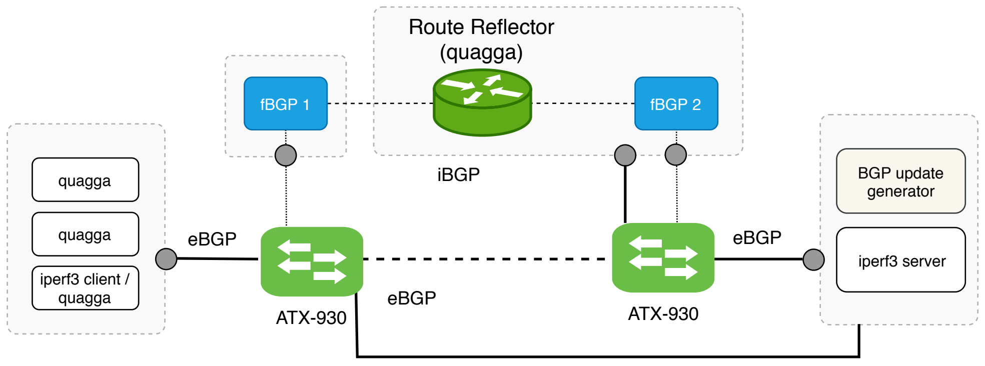
[5] https://iperf.fr/

the data plane. The main focus is the impact of using multiple tables and of the switch performance on updating the forwarding rules on traffic.

4.4.1 Experiment Setup



(a) Traditional mode setup



(b) OpenFlow mode setup

Figure 4.10: Experiment setup. Filled circles are data-collecting points.

The experiment setup is shown in Figure 4.10. The switches support OpenFlow mode and traditional IP mode with BGP capability. The setup simulate a simple ISP topology with two POPs. The iperf3 server represents the upstream provider whereas the iperf3 client represents the customer. The physical connections are all 1Gbps. iperf3 is used to measure bandwidth, jitters and packet loss. The traffic tool may not provide a highly accurate result as it is software-based. However, due to its accessibility and easiness to use and the lack of accessibility to a hardware-based tool, iperf3 was selected. The iperf3 client sends UDP traffic at full-rate (1Gbps) to the iperf3 server. Packet loss and jitters were recorded at the server. Network Time Protocol (NTP) was used to synchronise the clock of those machines.

Experiments vary in terms of rates (number of updates per second), number of prefixes per update and types of updates (announcement, withdrawal, or mix). Each experiment was carried for BGP and *fBGP* configuration.

To understand the effect of the dynamics in the control plane (i.e. control plane convergence) on the data traffic, iperf tests were used for the evaluation. The iperf client sends continuously UDP traffic at 100Mb/s to the server (There were attempts to send traffic at higher rates, but the traffic was randomly dropped at the docker containers). The iperf server announces its prefixes to the border router R2. The update generator was configured to send updates that causes the disturbance in the control plane, hence the data plane. The iperf tests provide a number of measurements including bandwidth, packet loss and jitters. Two performance metrics being examined are jitters and throughput.

4.4.2 Results

The results on convergence performance and forwarding performance are reported and discussed as follows:

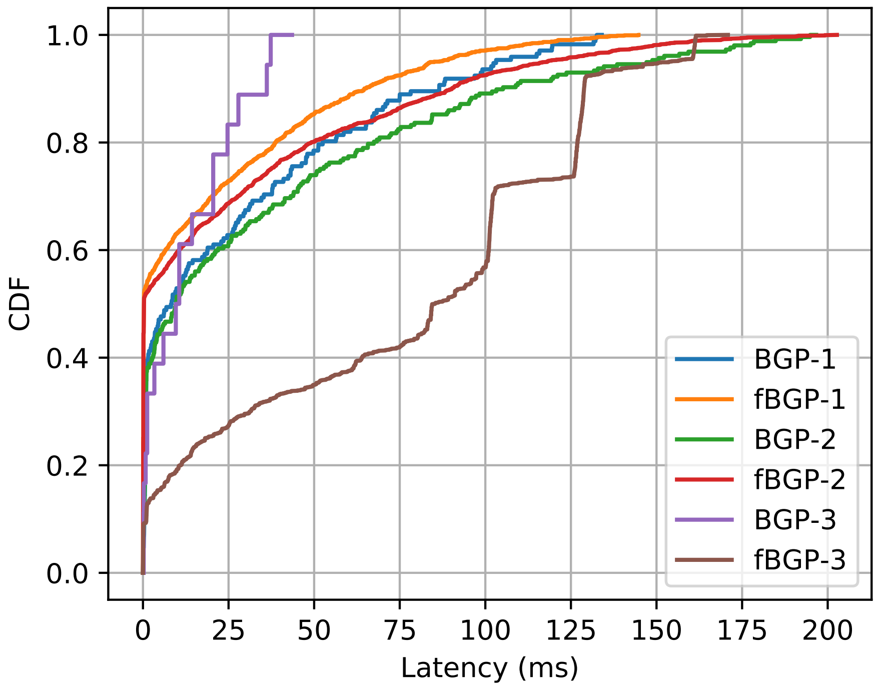


Figure 4.11: Latency comparison between BGP and fBGP in three experiments. In all cases, BGP outperforms fBGP.

Convergence Performance

Three different experiments, called Exp-1, Exp-2 and Exp-3 were used. In Exp-1, updates are generated randomly as announcements or withdrawals at the rate of 10 updates/second. Each update carries a random number of prefixes up to five. Exp-2 is similar to the first one, but the number of prefixes is fixed at one per update. Exp-3 uses realistic BGP data trace obtained from RouteViews.

Figure 4.11 compares the convergence performance of BGP and *fBGP* in terms of processing power of the controller (let call it control plane convergence). As can be seen, BGP performs slightly better than *fBGP* in Exp-1 and Exp-2 (about 20ms faster in average). In Exp-3, due to unknown technical issues tcpdump did not fully capture all the updates. However, the captured data shows that BGP outperforms *fBGP* significantly. A closer look at the data trace shows most of the updates contain five or more prefixes and many of them have both announcements and withdrawals. The current *fBGP* implementation processes prefixes sequentially. Performance would be improved if parallel processing is used thanks to the independence of prefixes.

Figure 4.12 analyses the convergence performance with regard to the flow installation in the data plane (let call it data plane convergence). Similar to the control plane convergence, *fBGP* performs poorly with realistic data. This is explained by the fact that an update carrying multiple prefixes results in the same number of OpenFlow modification messages.

Forwarding Performance

Impact on jitters were tested in two scenarios. In the first scenario, the FIB rule used by the test traffic was updated and reprogrammed and in the second scenario, the other FIB rules were being updated instead of the test traffic carried one. In both cases, jitters are almost identical for both BGP and OpenFlow modes. Noted that the results are indicative only. This is because iperf, as a software traffic generator, may not be able to generate reliable and consistent traffic.

The result as shown in Figure 4.13 shows that the ATX-930 switches performs slightly better with regard to jitters in OpenFlow mode as compared to the BGP mode. Thus, the multi-table pipeline used by *fBGP* seemingly

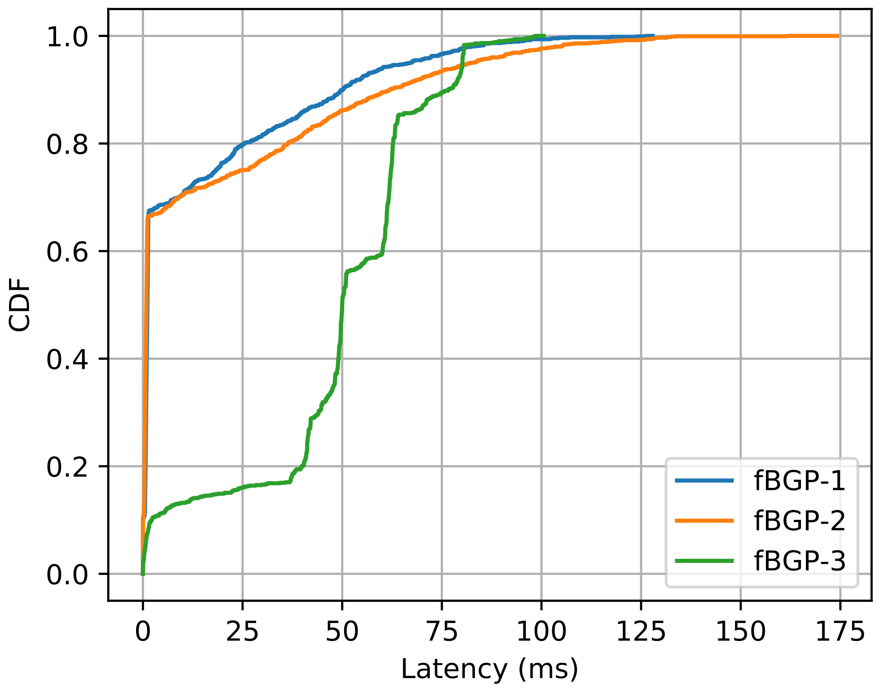


Figure 4.12: Latency of OpenFlow. fBGP-1 and fBGP-2 are randomly generated updates at different rates, fBGP-3 is realistic updates from RouteViews.

does not have any negative effect on data traffic. A closer look at the switches using two CLI commands, *show openflow rules* and *show openflow flows* gives some explanations. The OpenFlow rules sent by the controller are stored in the switch's memory and are not actually used until there is a matching packet. The switch then creates a flow by collapsing the whole pipeline into a single entry and installs into the ASIC. If no packet is observed for a specific period (i.e. one second or so), the flow is removed. However, the first packet of a flow usually observes several tens of millisecond delay.

The impact on throughput was analysed based on the reported band-

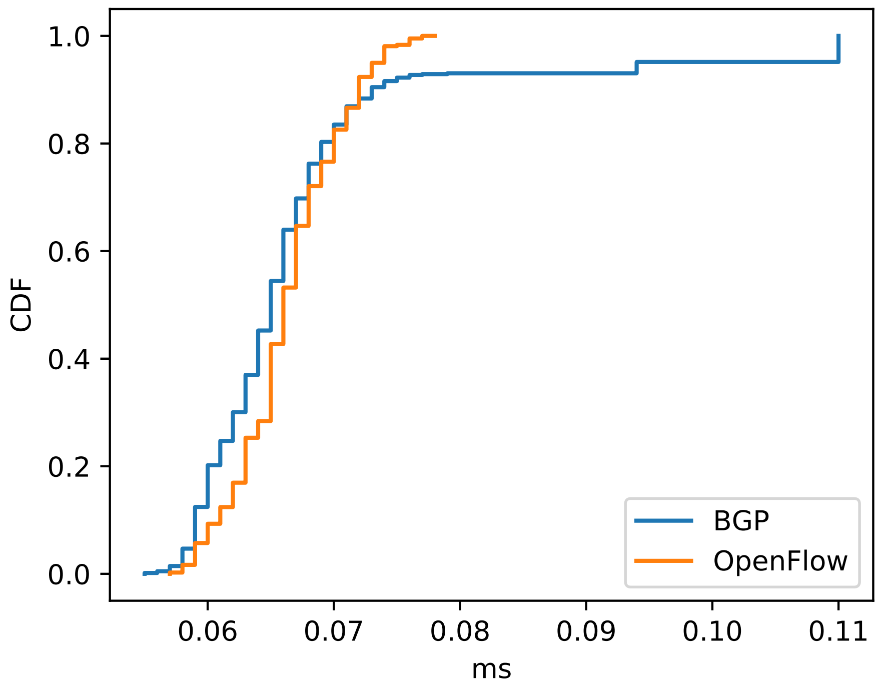


Figure 4.13: Allied-Telesis switches perform slightly better in terms of jitters in OpenFlow mode compared to BGP mode.

width reported by iperf. For the throughput tests, the update generator was configured to regularly announce and withdraw a superior update to router R1 that cause it to switch traffic to the iperf server from the indirect link via R2 to the direct link. However, there were technical issues with iperf setup and BGP configuration that caused abnormality in the iperf results. Iperf reported a long period of lost connectivity (i.e. *nan%*) when the superior update was sent. The issues were not resolved before the BGP and OpenFlow licenses expired. Nonetheless, the result is still reported in Figure 4.14. It shows that BGP experiences a longer period of lost connectivity as compared

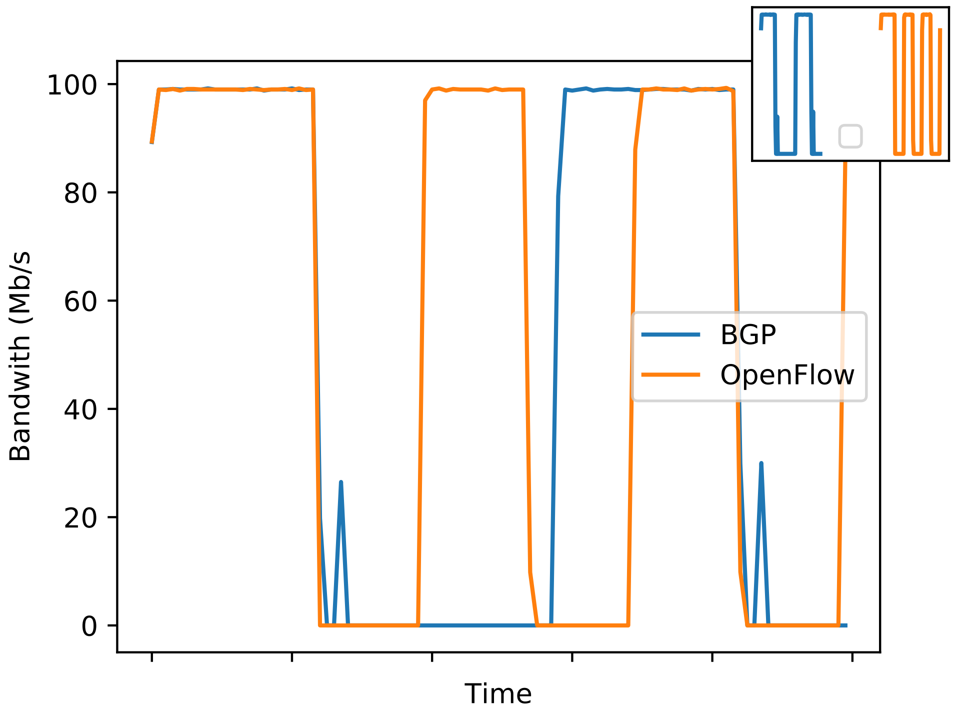


Figure 4.14: Throughput test. Technical issues with the experiment cause long periods of lost connectivity as reported by iperf for both BGP and OpenFlow mode.

to OpenFlow. It is probably because of BGP route dampening algorithm. Further investigation would be needed for a proper explanation. It was observed in the test bed that reducing MRAI timer (it is the other timer which can add more delay to BGP updates) of quagga to zero causes the high CPU utilisation. For that reason, MRAI timer was set to one second. This, however, still does not explain the lost of connectivity.

4.5 Summary

This chapter presented an OpenFlow-based design for border routers to enable multipath interdomain routing in transit ISP networks. The design enables the operators to achieve more flexible routing policies than the traditional single-path-capable BGP routers. A prototype was implemented based on Faucet, an open-source production-ready SDN controller and deployed in a physical testbed. This deployment proves the design is functional and practical. Although *fBGP* does introduce a small penalty in control-plane convergence performance, there is no impact on forwarding performance in respect of both throughput and jitter metrics. The true benefits of *fBGP*, thanks to OpenFlow are the multipath-capability and the operational flexibility by decoupling the software (controller) from the hardware (forwarding plane).

Chapter 5

Design of and Implementation of FIBO

This chapter provides an overview of the design, implementation and evaluation of a FIB distribution mechanism, called FIBO. *FIBO* is a part of the *SDIRO* architecture. *FIBO* emulates a distributed data plane consisting of multiple switches as a single logical forwarding table. *FIBO* efficiently utilises the TCAM spaces in switches while preserving the forwarding behaviour.

The chapter has four main sections: (1) Section 5.1 provides an overview of the problem and the approach; (2) Section 5.2 formally defines the problem as a linear programming problem; (3) Section 5.3 presents a heuristic algorithm that trades efficiency for computation time; (4) Section 5.4 provides an evaluation in terms of efficiency and complexity, in comparison with the traditional approach.

5.1 Overview of the Problem

High-speed routers rely on TCAM memory to look up the next-hop for packets at the hardware line rate [89]. TCAMs are expensive, power-hungry, and sizeable [90]. In a router, the TCAM space is usually shared between routing and other applications such as firewall and packet classification.

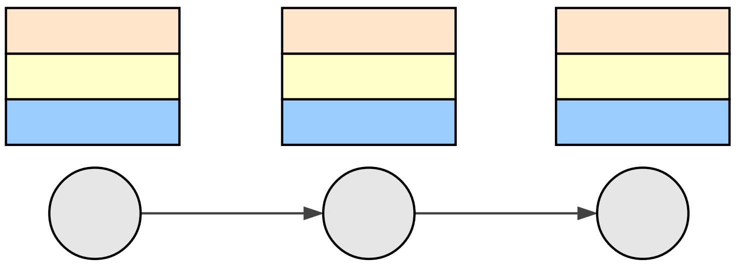
In a typical transit network, border routers require over 700K TCAM entries for BGP routes. The figure is doubling every 5 to 6 years [1]. In 2014, the BGP routing table exceeded the memory capacity of many Internet routers, causing massive service disruption [2]. Deployment of the multipath routing capability presented in Chapter 4 exacerbates this scalability problem.

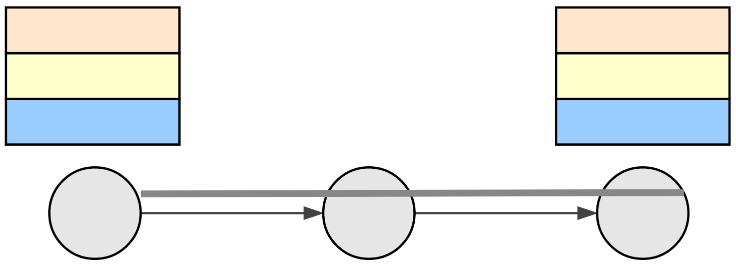
In a conventional ISP POP with multiple border routers, BGP routes learned from external peers are replicated at each hop along the forwarding path (as demonstrated in Figure 5.1a). In modern ISP networks (i.e. BGP-free core), routes are installed only at the ingress and egress routers (demonstrated in Figure 5.1b. Tunnelling techniques (e.g. MPLS) are used to tunnel packets from the ingress to egress routers. These networks poorly utilise the available TCAM spaces.

To reduce sizes of the FIB table, the common technique used by ISP operators is route aggregation (i.e numerically grouping prefixes into a super prefix). The performance of route aggregation varies from router to router and network to network, depending on the appearance of prefixes in the route table. For example, if most of the routes have the same next-hop the table size can be greatly reduced using route aggregation.

[1]https://bgp.potaroo.net/

[2]The forwarding table exceeded 512K entries causing memory overflow in core routers. https://bgpmon.net/what-caused-todays-internet-hiccup/


(a) Hop-by-hop forwarding where the FIB is replicated at hops.


(b) Tunnelling routing where the FIB is installed at the edges.

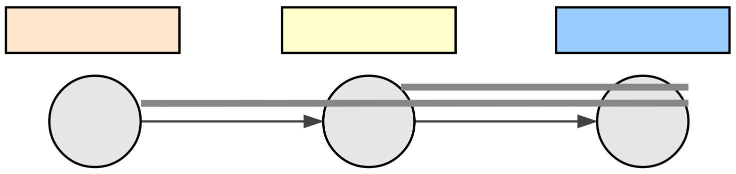
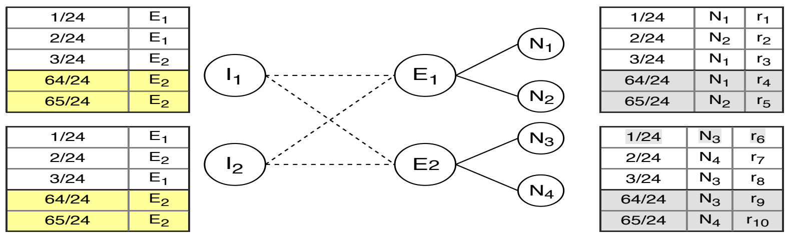

(c) The concept of FIB distribution.

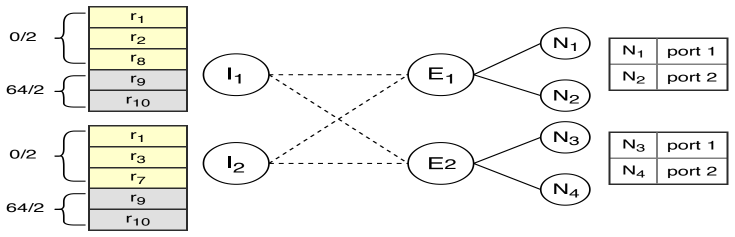
Figure 5.1: Routing techniques and the approach to FIB distribution.

The basic concept of *FIBO* is demonstrated in Figure 5.1c. The FIB table is decomposed and installed into all routers along the path. Tunnels are established between switches so that the forwarding behaviour is maintained.

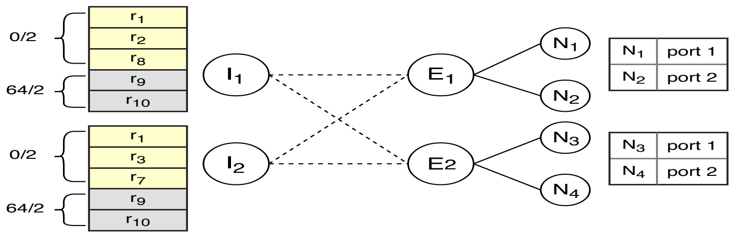
To show how *FIBO* works, let consider an example in Figure 5.2. Five unique prefixes are reachable via two routers E_1 and E_2. The corresponding FIB tables are shown in Figure 5.2a. FIB entries are routing prefix written in a format of the first IP address in a network followed by a slash (/) and ending with the bit-length of the prefix. For instance, 1.0.0.0/24 (or 1/24 for



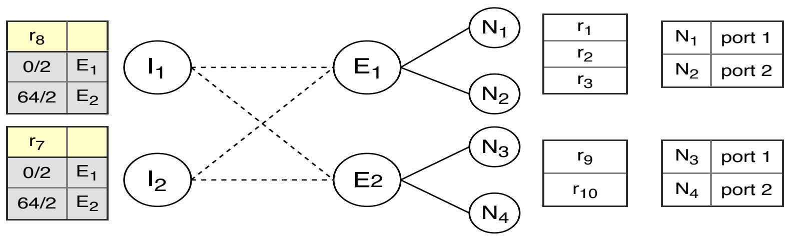
(a) The original FIB tables in each router. The first col. is the destination prefixes and the second col. is the next-hops. The third is the rule identity used for demonstration purpose only. The yellow shaded entries can be aggregated and grey shaded ones are redundant



(b) FIB entries can be installed at PE routers. Processed packets can be tagged and tunnelled to the border router. The right-hand side tables contains the mapping between tags and out ports



(c) Selection of cover prefixes which are used to decompose the FIB tables



(d) The final FIB tables in routers are shown, assuming a special rule placement strategy has been applied.

Figure 5.2: Operations to reduce duplication and minimise the FIB size.

short) is an IPv4 prefix of having 24 bits for network and 8 bits for hosts.

By using tunnelling technique, FIB entries in E_1 and E_2 can be eliminated. Then, FIB tables in I_1 and I_2 are to be decomposed with a set of cover prefixes (Figure 5.2b). Intuitively, two cover prefixes that can be used are 0/2 and 64/2. The final step is to distribute FIB entries. In this example, entries belong to the two aggregate routes can be placed into E_1 and E_2. The aggregate routes are installed to the ingress I_1 and I_2 to direct packets towards E_1 and E_2. The final FIB tables in routers (Figure 5.2d) are smaller than the original ones.

The key problem to be solved here is to find a set of cover prefixes which can be used to decompose the rule tables into multiple smaller tables (hereafter called subtables) and to find the locations for those subtables in order to reduce the duplication while satisfying the table space constraint in switches and the routing policy requirement. The next sections present the formation of a linear programming problem for this FIB distribution problem.

5.2 Problem Definition

This section describes a mathematical model of the FIB distribution problem.

5.2.1 Overview

Let denote $G = \langle S, L \rangle$ a network composed of a set of switches S and a set of links L between them. Each switch $s \in S$ has finite C_s number of rule entries it can hold. Let I and E be the set of edge switches which are ingress (i.e. customer-facing) switches, and egress (i.e. provider/peer-facing) switches respectively.

The router learns routing update from its neighbours and computes a global FIB table $\mathcal{F} = \{r_1, ..., r_n\}$ for a set of unique prefixes $P = \{p_1, ..., p_n\}$. Each entry r is unique and identified by a tuple (*prefix*, *next hop*); $r = \langle p, n \rangle$. A next hop can only be connected to one egress switch $e \in E$. As in a traditional router, the identical FIB table $\mathcal{F}$ will need to be installed into all ingress switches in order to enforce the forwarding correctness.

Notation	Description
S	A set of switches
$E \subset S$	Set of egress points
$I \subset S$	Set of ingress points
Q	Set of cover prefixes
$i, j \in I$	An ingress switches
$e \in E$	An egress switch
$q \in Q$	Represent a portion of FIB entries (hereby called a subtable)
$s \in S$	A switch in network G
$y^e_q(i, j)$	No. of rules shared b/w i and j of given egress e & cover prefix q
$x^i_s(q, e)$	Placement of rule group e of q of a given ingress i into a switch s
$l^i_e(s)$	If s lies on the shortest path $i - e$
$k^i_q(e)$	If q to be assigned to the shortest path $i - e$

Table 5.1: Summary of notation used in the model

When a packet arrives at an ingress switch i and hits a rule r it will be delivered to the corresponding egress switch which is connected to the rule's next-hop. At the egress switch, the packet will then be forwarded to the next-hop. Each switch has an implicit rule which drops packets that hit no actual rule.

In order to scale out the FIB table $\mathcal{F}$, it will be decomposed into multiple small tables (hereby called subtables) thanks to the fact that IP prefixes are numerical aggregated. Given two prefixes p and q, the notion $p < q$ denotes that prefix q overlaps prefix p or p is encompassed by q. Based on that, the table $\mathcal{F}$ can be decomposed into subtables by a set of prefixes Q (i.e. each $q \in Q$ encompasses several prefixes in P). Thus, for a subtable F_q with a corresponding $q \in Q$, it can be specified as $F_q = \{r_i \in F | r_i.p \leq q\}$.

The FIB distribution problem is defined as to find the assignment $f : I \times Q \times E \mapsto S$ such that the switch space constraint and the intradomain policy is satisfied. The assignment f determines that rules having a particular egress switch e will be installed on a switch s for a given ingress switch i.

Optimizing the FIB distribution would mean to find the assignment f such that the total number of rules to be installed into switches are minimized without breaking the forwarding correctness. The optimisation problem is composed of two components. The first component, *decomposition* decomposes the FIB table into subtables by determining the set Q. The second component, *rule placement*, computes the assignment f. The rule placement component is discussed in the next section as it is considered to be critical for the distribution performance.

5.2.2 Rule Placement Problem

For a given ingress i, a cover prefix q is assigned to the path between i and an egress switch e which was chosen by the intradomain control plane. By computing a (cover) rule that matches on q, whose next-hop is the subsequent switch along the path, the *forwarding path* for q can be formed. Then, all rules belonging to q can be distributed to the switches lying on the forwarding path. A packet which does not match an actual rule will be forwarded to the next switch following the forwarding path. The final switch drop packets that hit no actual rules by default.

Since a rule of a subtable F_q may have an egress switch which is different from that of its forwarding path, it must be placed into the switch that lies on a valid path (chosen by the intradomain) from the ingress switch to its egress switch. Thus, the problem is to allocate rules to the right switch which does not violate the intradomain policy and the switch space constraint.

This placement problem can be solved for every single rule. However, it is unnecessarily expensive with hundreds of thousands of rule and not scalable. Note that, all rules belonging to q that have the same egress switch can be placed to the same switch. Thus, the placement problem would be faster if rules are considered in groups of rules having the same egress (hereby called egress groups or rule groups, denoted as $F_q(e)$) for each subtable. The following constraints have been identified.

Forwarding semantic constraints:

$$\forall i, q \in I, Q : \sum_{e \in E} k_q^i(e) = 1 \tag{5.1}$$

$$\forall i, q, e \in I, Q, E : \sum_{s \in S} x_s^i(q, e) = 1 \tag{5.2}$$

$$\forall i, e, q \in I, Q, E : \sum_{s \in S} \sum_{e' \in E} x_s^i(q, e) \cdot k_q^i(e') \cdot l_e^i(s) = 1 \tag{5.3}$$

Constraint (5.1) ensures that a cover prefix can only be assigned to a single forwarding path. Constraint (5.2) limits the placement of a rule group to a switch, while constraint (5.3) ensures that the assigned switch has to lie on both the forwarding path, and the chosen path between the ingress and the egress of the group.

In addition, the placement must satisfy the switch space constraint below.

Space constraint:

$$\forall s \in S : \sum_{q \in Q} \sum_{e \in E} |F_q^s(e)| \leq C_s \tag{5.4}$$

where $F_q^s(e)$ denotes the set of rules $F_q(e)$ to be installed to switch s.

For a given switch s and ingress i if $x_s^i(q,e) = 1$, then the rule set $F_q(e)$ will be installed into the switch. Since a single copy of $F_q(e)$ is required in switch s to satisfy the routing for all ingress i, therefore $F_q^s(e) = F_q(e)$ if $\sum_{i \in I} x_s^i(q,e) \geq 1$, otherwise $F_q^s(e) = \emptyset$. Having said that, the constraint in 5.4 can be rewritten by the following two constraints.

$$\forall s \in S; q \in Q; e \in E : K_q^s(e) \geq x_s^i(q,e)|F_q(e)| \tag{5.5}$$

$$\forall s \in S : \sum_{q \in Q} \sum_{e \in E} K_q^s(e) \leq C_s \tag{5.6}$$

where $K_q^s(e)$ is the minimum free space a switch s is required to have for the rules in a subtable $F_q(e)$.

The main goal to minimise the maximisation of space utilisation i.e to equally split the table into all switches. Therefore, the objective function is defined as below:

$$\min \quad p_s(q,e) \qquad \forall q \in Q; e \in E; s \in S \tag{5.7}$$

This LP problem can be solved using a standard ILP solver. Since ILP is NP-hard, the solving time may not be practical to deal with the rate of changes occurring in the FIB table (as BGP updates often come in burst and there may be hundreds of update per second). Therefore, a lengthy solution time can cause backlog and result in traffic disruption.The next section proposes a simple heuristic approximation algorithm which trades optimality for time.

5.3 Algorithm

This section discusses the rule placement problem and present the formation of a linear programming model for optimisation.

A popular solver SCIP [3](version 4.0.1) was used to solve the above linear optimisation problem. However, a satisfied result was not achieved although efforts made to improve the performance by customising multiple settings of the solver. This leads to development of an heuristic algorithm aiming at obtaining a faster computation performance.

5.3.1 A heuristic algorithm

This section describes a heuristic algorithm for the forwarding table placement in a distributed network aiming at improving the computation latency. The algorithm is presented in Algorithm 1 and Algorithm 2.

At high-level, the heuristic algorithm is composed of three components. The first component computes the forwarding paths between the ingress and egress switches that satisfy the intradomain routing policy. The second component computes the distribution of a subtable over a forwarding path and for a particular ingress switch. The main component computes the mapping between the subtables and forwarding paths for all ingress switches.

The algorithm works as follows. It sorts the ingress switches based on their number of rules. It then starts with the ingress switch having the largest

[3]https://www.scipopt.org/

Algorithm 1 Heuristic algorithm for the placement problem

```
Input: $G, I, Q$
Output: a map $f : (I, Q, E) \rightarrow S$
 1: create an empty map $f$
 2: sort $I$ in descending by table sizes {i.e. $|F^i|$}
 3: for $i \in I$ do
 4:     sort $Q$ in descending by table size {i.e. $|F^i(q)|$}
 5:     for $q \in Q$ do
 6:         $u^* \leftarrow \infty$ {the maximum switch utilisation}
 7:         $f_i^* \leftarrow \emptyset$ {the best mapping for the given $i, q$}
 8:         for $e \in E$ do
 9:             $p \leftarrow$ shortest_path_between($G, i, e$)
10:             $u, f' \leftarrow$ distribute($i, q, p$)
11:             if $u < u^*$ then
12:                 $u^* \leftarrow u$
13:                 $f_i^* \leftarrow f'$
14:             end if
15:         end for
16:         update the map $f$ with items in $f_i^*$
17:     end for
18: end for
```

number of rules. For each ingress switch, the algorithm tries to allocate the largest subtable first. For a particular subtable, the algorithm examines the potential distribution over all possible allowed paths between the ingress and the egress switch. The distribution with the least maximum switch utilisation is selected for that table. This process is repeated until all subtables are allocated.

The main component is presented in Algorithm 1. It takes a network topology G, a set of ingress switches I and a set of cover prefixes Q as input. The output is a matrix that maps each element in I, Q and E to the set switch S. It assumes the intradomain policy is the shortest path (line 9). An appropriate function that implements the intradomain policy should replace the shortest path.

The distribution component is presented in Algorithm 2. The algorithm ensures that rules in a table are placed to switch in group based on their egress and that switch utilisation is minimised. The algorithm uses a matrix to keep any valid placement i.e. satisfying constraint (5.3) (line 1 to 9). It then looks for the switch with the least valid placement (line 11), and tries to allocate as many groups of rules as possible (i.e. satisfying the constraint (5.4) (line 13 to 16). Then, the allocated egress is removed from the computation (line 17 to 18). The process keeps going until no egress left.

5.3.2 Decomposition

The quality of the decomposition of the FIB table can affect the feasibility and quality of the distribution. The smaller and the less diverse the subtables are, the better space utilisation the distribution will produce. If subtables are too big the distribution may not be able to find paths for them and will fail. However, large number of subtables increases the computation complexity. In addition, the high egress diversity will affect the ability to find a path for a subtable. The best case scenario is when egress diversity is zero. The subtable then can be placed to a switch which lies on the optimal path between any ingress switch and the egress switch. The decomposition algorithm is not considered in this work. In the evaluation, the decomposition set Q is determined manually in advance.

5.3.3 Incremental Updates

FIB entries are not static. They may change as the routing policy changes or new routing updates are received. For instance, a new rule may be added or

Algorithm 2 Distribute a table over a selected path

```
Input: $i, q$
Input: a forwarding path $p = \{S_1, .., S_n\}$
Output: mapping $f : (i, q, E) \rightarrow S$
 1: create matrix $m$ of zeros with $E$ rows and $n$ columns {$m[e, s] = \{0, 1\}$ denotes
    whether a group of rules with $e$ as egress can be placed into switch $s$}
 2: for $e \in E$ do
 3:   for $s \in p$ do
 4:     $p' \leftarrow$ shortest path between $i, e$
 5:     if $s \in p'$ then
 6:       $m[e, s] = 1$
 7:     end if
 8:   end for
 9: end for
10: while $E \neq \emptyset$ do
11:   $s^* \leftarrow$ get column with smallest sum in $m$
12:   for $e \in E$ do
13:     if $m[e, s^*] = 1$ then
14:       additional rule set $K \leftarrow F^i_(q, e) - F_{s^*}$ {$F_s$: set of rules present in switch
          $s$}
15:       if available space of switch $s^* > |K|$ then
16:         update map with $(i, q, e)$ as key, $s^*$ as value
17:         $E \leftarrow E \setminus \{e\}$
18:         reset row $e$ i.e. $m[e, :] \leftarrow 0$ {i.e. mark it as done}
19:       end if
20:     end if
21:   end for
22: end while
```

an existing rule is updated or removed upon the arrival of a BGP update. The new FIB table may invalidate the current allocation hence a recalculation of the allocation.

However, this brute-force approach may not be efficient and practical enough to cope with the high frequency of BGP updates in the Internet. A lengthy computation can cause backlogs and delay important updates to be processed. Moreover, the new allocation may shuffle a large number of rules,

Algorithm 3 A simple incremental algorithm for BGP updates

```
Input: BGP update $U$
  while TRUE do
    $U \leftarrow newupdate$
    $Q' \leftarrow \emptyset$
    for announcement in $U$ do
      if announcement is withdraw then
        update rule tables and continue
      else if announcement is advertisement then
        for new route $r$ do
          for $q \in Q$ do
            if $r.p < q$ then
              add $q$ to $Q'$
            end if
          end for
          if add to the current allocated switch fails then
            run algorithm 1 on set $Q'$
          end if
        end for
      end if
    end for
  end while
```

potentially affecting significant number of traffic flows.

Instead, an incremental algorithm was introduced. It is based on the observation that if an update is a withdrawal that does not change the FIB table, then no action is needed. Otherwise, for each new route introduced the cover rule for it will be identified. These new rules are then installed to the associated switch. If there is no space left in the switch, allocation for those cover rules will be required. The algorithm avoids recalculation for a large number of updates and limits the calculation to smaller number rules, hence reduced latency.

5.3.4 Handling Failures

Failures are inevitable in any network. A switch failure can cause temporary packet loss for traffic handled by the portion of the FIB table on that switch. Link or switch failure causes topological changes. One way to deal with this is to re-run the algorithm with the new topology as input. However, it is difficult to guarantee fast failover and may result in significant shuffle of traffic. For fast failover, the portion in each switch should be cloned to a backup switch. The controller then can react to topological changes by reprogramming the ingress switches. Due to time constraint, this research does not investigate failure scenarios and solutions.

5.3.5 Algorithm Analysis

This section provides a time complexity analysis of the heuristic algorithm. The space complexity is not considered as memory is not really a concern when running the algorithm in a general-purpose server. The analysis focuses on two components: the distribution component and the mapping component. The intradomain component is not taken into account as it is not part of the interdomain controller and should be delegated to the intradomain controller.

The time complexity of the distribution algorithm (Algorithm 2) will be discussed first, followed by the mapping algorithm (Algorithm 1). The distribution algorithm runs twice the loops over two sets E (egress switches) and p (switches in a forwarding path). The first run gives a runtime $\mathcal{O}(E \times p)$ (Lines 2 - 9). The second run is a double nested loop which gives a runtime $\mathcal{O}(\frac{E\times(E-1)}{2})$ which is equivalent to $\mathcal{O}(E^2)$. Thus, in total, the runtime complexity would be $\mathcal{O}(E^2)$ (Lines 10 - 22).

The mapping algorithm runs a nested loop over three sets I (ingress switches), E (egress switches) and Q (subtables). The last loop (Lines 8 - 14) produces the runtime $\mathcal{O}(E)$ in the worst case. This gives a runtime $\mathcal{O}(I \times E \times Q)$. Thus, this gives the total runtime complexity of the algorithm $\mathcal{O}(I \times Q \times E^3)$ in the worst case scenario.

As can be seen, the algorithm does not scale very well with the egress switches. It is not, however, affected by the number of transit switches in the network. Thus, in a practical deployment, scalability can be achieved by adding more transit switches. With regard to decomposition of the FIB table, there is a trade-off between the distribution efficiency (it means a finer decomposition or more subtables) and runtime.

5.4 Evaluation

The heuristic algorithm was implemented in Python for the purpose of evaluation. An ILP solver was also implemented using the SCIP library (Solving Constraint Integer Programs) with Python API (PySCIPOpt). The evaluation focuses on two aspects: *efficiency*, *latency* and *scalability*.

The efficiency metric is measured based on the utilisation of table space in the network and is computed as follows: $e = \frac{N}{\sum_{s \in S} R_s}$, where R_s is the number of rules in switch s and N is the total number of unique rules. The lower the value e is, the higher the duplication will be. It is ideal when no duplication of rules exists, i.e. $e = 100\%$.

The latency metric is concerned with the processing latency, i.e. time taken to process a BGP update and is measured using real BGP data traces.

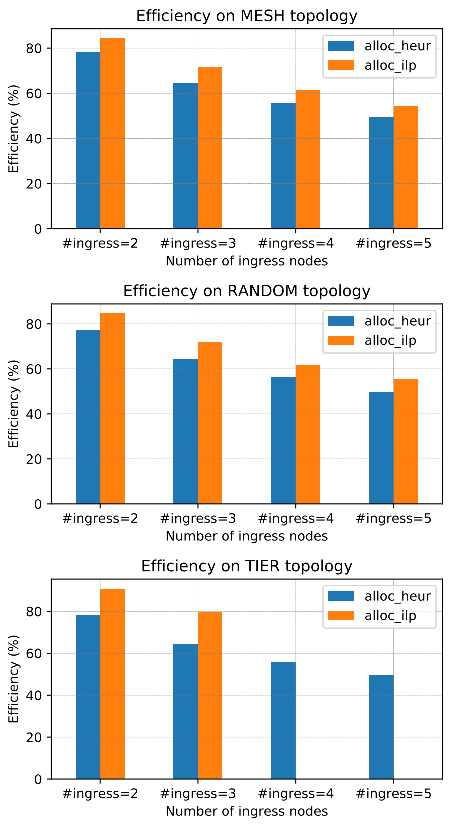
The solution is assumed to be practical if the processing latency is less than a budget of 50 ms which is a common requirement in ISP networks [91]. The BGP datasets were obtained from RouteViews for a week in November 2017. Among the update messages in the datasets, more than 135K causes actual changes to the FIB table, of which 128K updates are advertisements.

The scalability metric measures how well the system scales with regard to the size of the FIB table. The scalability metric is computed as the function of the number of destination prefixes and the largest FIB table size in the network. Based on this definition, in a conventional BGP network or other works such as RouteFlow [10] and SDN-IP [9], the scalability metric will be identical to number of prefixes.

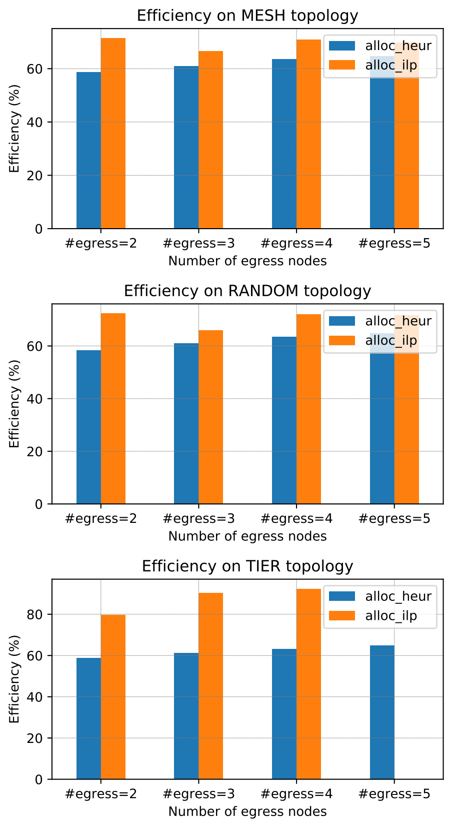
The topology of the data plane used for evaluation is synthetic. Three different kinds of topology were used, including full-mesh, 3-tiered and random kind. In the full-mesh topology, all switches are directly connected to one another. In the 3-tiered topology, the first tier contains ingress switches and the third tier contains egress switches interconnected to the first tier via a middle tier. This 3-tier topology is recommended for POP design by large vendors such as Cisco.

It is not possible to generate a realistic FIB table for a particular ISP due to lack of information about their routing policy and interconnections with the neighbours. Therefore, the FIB table was generated by randomly assigning prefixes to egress switches. For each pair of destination prefix and an ingress switch, the next-hop is randomly selected from the set of egress switches. It is noted that the resulting FIB table has an equal distribution of prefixes per egress switch and majority of adjacent prefixes do not share the same next-hop.

5.4.1 Efficiency



(a) Efficiency with various number of ingress switches.



(b) Efficiency with various number of egress switches.

Figure 5.3: Efficiency on different kinds of topology. The missing data for ILP is because a time limit is set for the solver.

Figure 5.4.1 shows the result for efficiency metric. Overall, the ILP solver performs better than the heuristic algorithm, but the difference is not significant.

In 3-tiered topology, the two have the same performance. As expected, the efficiency decreases as the number of ingress switches increases. This is because more rules are duplicated at ingress switches. An interesting observation is that increasing the number of egress switches i.e. diversity, increases the performance. This is contrary to the property of the FIB aggregation technique. The heuristic algorithm works the best in 3-tiered topology among other kinds.

5.4.2 Processing overhead

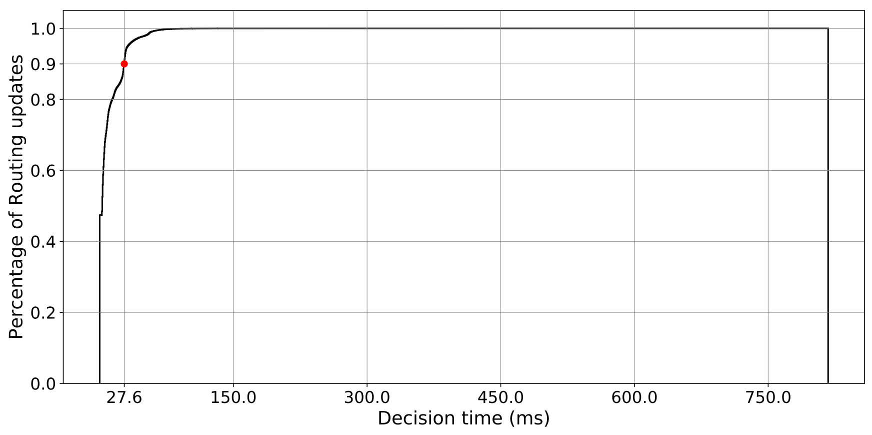


Figure 5.4: Processing latency with real BGP updates

Figure 5.4.2 plots CDF of the processing delay. Most of the updates (99%) can be processed under 60ms and 90% under 30ms. Noted that, the current implementation does not utilise parallelism for improved performance as it was based on a single Python process. If parallelism and multiprocessing were introduced, the processing latency would be reduced. However, a risk of disordered updates should be considered.

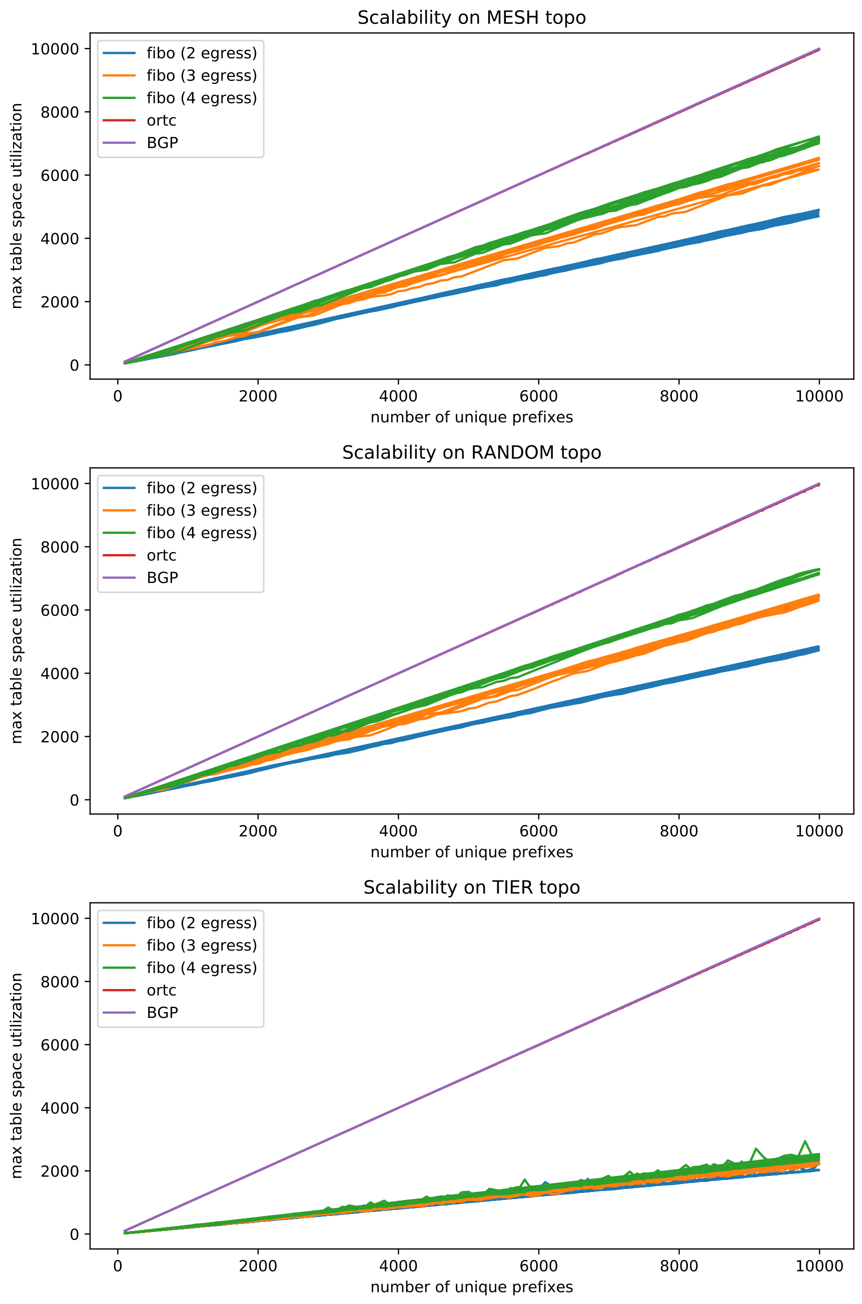


Figure 5.5: Data plane scalability w.r.t number of unique prefixes. Conventional BGP and ORTC are almost lined up with randomly generated routes.

5.4.3 Scalability

In evaluation of scalability, three different types of topologies: MESH, RANDOM and TIER were used, each with different number of egress switches from 2 to 5. The FIB table increases in size at 100 prefixes per step. At each step, prefixes are generated randomly and a egress node is chosen randomly for each prefix. The test is repeated five times for each setting.

As shown in Figure 5.4.2, the heuristic algorithm works well in all settings. It means that a large number of prefixes can be supported by a network comprising of very small switches. It is interesting that increasing the number of egress, i.e. path diversity, reduces the table size. In contrast, FIB aggregation algorithms perform well with less path diversity. It is also observed that, Optimal Routing Table Constructor (ORTC) a popular FIB aggregation algorithm [92] and BGP are very close in term of FIB reduction with the randomly generated FIB table. ORTC works by storing the FIB in a binary tree and traverses the tree three times to generate an aggregated FIB.

5.5 Summary

This chapter discussed the FIB distribution as the approach to routing scalability in transit ISP networks. It presented a formulation of the problem as a linear programming problem and a heuristic algorithm that trades efficiency for time. The heuristic algorithm can handles 90% of realistic BGP updates within 30ms while achieving a close efficiency to that of the ILP solver. The evaluation showed that the proposed mechanism improves the TCAM utilisation in various types of network settings.

Chapter 6

SDN-enabled Interdomain Routing: A Comparison

This chapter provides a detailed comparison and evaluation of two controller architectures, namely *legacy SDN* and *OpenFlow-based SDN*, with regard to the *complexity* of implementing *advanced routing capabilities* in a transit ISP. The legacy SDN refers to a network that is built using legacy routers (i.e. closed-box devices with a proprietary control plane that exposes limited programmability to an external controller), whereas OpenFlow-based SDN, as its name suggests, refers to a network that uses OpenFlow switches.

In this chapter, section 6.1 introduces the legacy SDN architecture and OpenFlow-based SDN architecture. In section 6.2, the method used for comparison and evaluation is presented. Of the subsequent sections (6.3, 6.4, and 6.5), each presents the implementation of a different advanced routing capability. Section 6.6 provides a discussion on the key advantages of an OpenFlow-based SDN, followed by a conclusion section.

6.1 Introduction

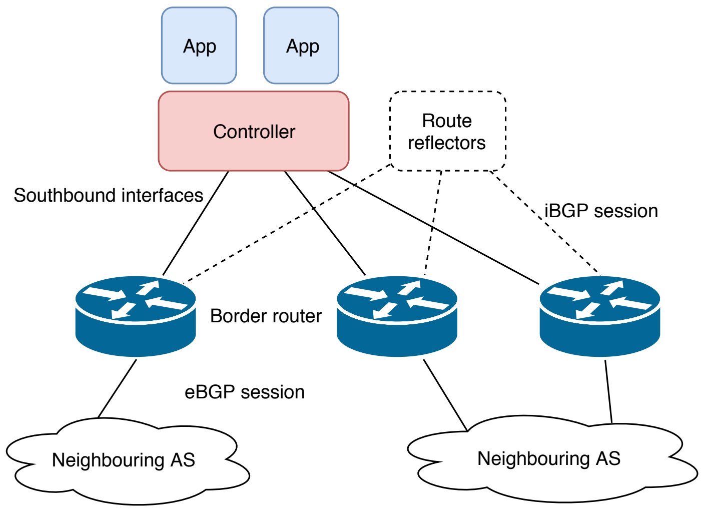


Figure 6.1: High-level architecture of SDN-enabled interdomain routing.

In a transit ISP, the interdomain routing network consists of border routers, BGP sessions to neighbouring ASes and a route distribution system (i.e. route reflectors). Traditionally, interdomain routing is controlled by applying distributed import and export policies in border routers. Conversely, in a network with SDN-enabled interdomain routing a centralised controller will be used to programmatically control the routing [10]. The high-level architecture of SDN-enabled interdomain routing is shown in Figure 6.1. The controller manages border routers, maintains the global view of the network and provides a platform to write routing control applications. It uses the provided southbound interfaces to program the routers.

SDN-enabled interdomain routing can be accomplished using various southbound mechanisms including I2RS, ForCES, NetConf and OpenFlow [93].

Currently, NetConf and OpenFlow are the most popular choices for SDN [1]. NetConf and OpenFlow can be considered as two extremes of the SDN spectrum: the later relies on a centralised controller and allows programmability of the forwarding table while the former only provides interfaces for configuring the control plane. Many published research works are based on OpenFlow, whereas NetConf is popular among industrial discussions [2]. Thus, it would be interesting to have an indepth comparison between the two approaches. The term *legacy* is used to describe NetConf and the related technologies.

The next section provides an overview of two architectural approaches to SDN-enabled interdomain routing: legacy SDN and OpenFlow-based SDN.

6.1.1 Legacy SDN and OpenFlow-based SDN

The term *OpenFlow-based SDN architecture* refers to a network whose border routers are OpenFlow-compliant and capable of flow-based forwarding. In contrast, a legacy SDN-enabled network uses legacy routers which are managed via NetConf. The term *legacy router* describes BGP routers with a destination-based forwarding capability.

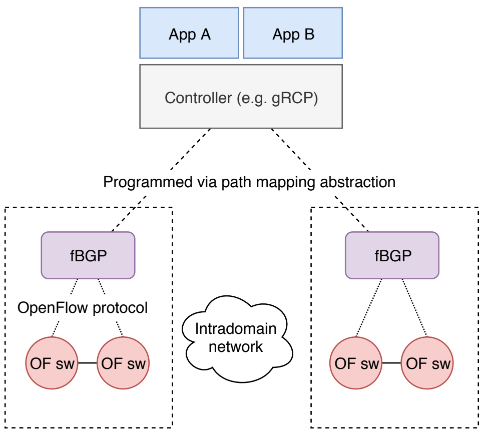
Examples of legacy SDN includes the famous Routing Control Platform (RCP) [65] and Intelligence Route Service Control Point (IRSCP) [67, 13] deployed in AT&T network. Network vendors also offer their own SDN solutions. The Application Centric Infrastructure (ACI) is offered by Cisco for policy-driven networks based on Cisco Nexus 9000 family of switches [3].

[1]https://www.sdxcentral.com/networking/sdn/definitions/southbound-interface-api/
[2]https://archive.nanog.org/sites/default/files/1_Moore_Network_Automation_And_Programmability.pdf
[3]https://www.linkedin.com/pulse/software-defined-networking-sdn-simplified-kevin

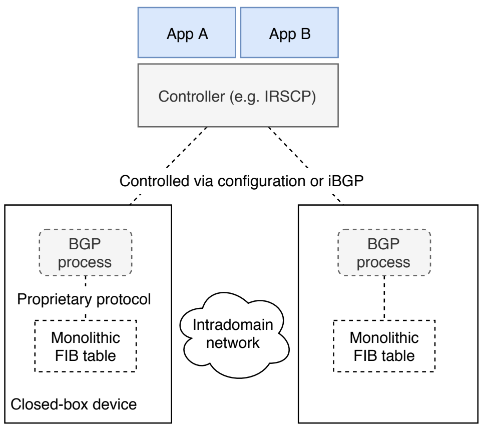
While Cisco ACI uses its proprietary southbound interface (called OpFlex), there are standardised interfaces including BGP, NetConf and SNMP.

Examples of OpenFlow-based SDN includes Software-Defined Exchange (SDX) [11], SDN-IP [9], RouteFlow controller [8] and its variant Cardigan [94].

For the purpose of this research, the focus of the comparison is placed particularly on the proposed *SDIRO* architecture and general legacy SDN. Figure 6.2 depicts high-level overview of the two architectures. Overall, they are architecturally similar with a centralised controller managing distributed border routers. A key difference is that legacy border routers are monolithic whereas that of *SDIRO* are made up of open software controllers and OpenFlow-compliant switches.



(a) SDIRO architecture



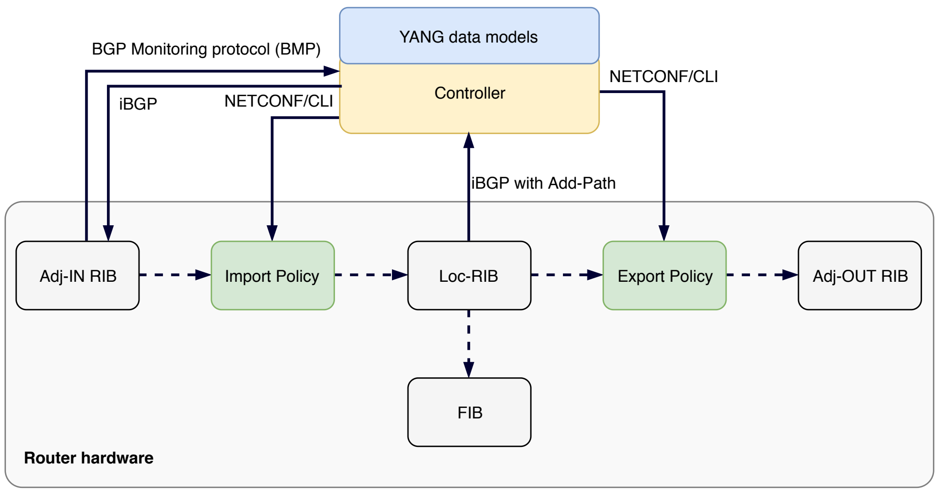
(b) Legacy SDN-enabled architecture

Figure 6.2: High-level overview of SDIRO and legacy SDN-enabled architecture (Dashed boxes depict proprietary components).

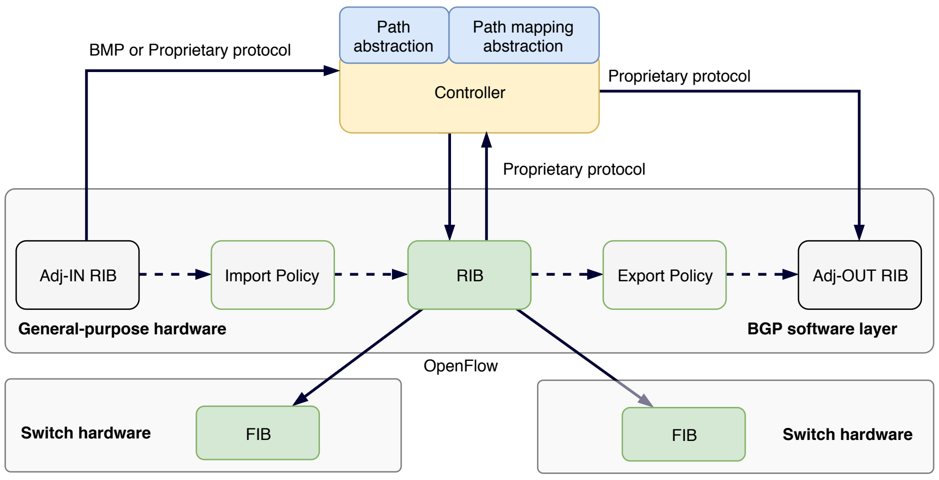
The next section discusses in more details the framework for programming routing control applications, of the legacy and *SDIRO* architecture.

6.1.2 Legacy Programming and SDIRO Platform

The main components of the legacy and *SDIRO* controller platforms are shown in Figure 6.3.



(a) Legacy programming platform



(b) SDIRO programming platform

Figure 6.3: Routing programmability in legacy and SDIRO network

In a legacy network, a router's forwarding table (FIB) can only be programmed indirectly by manipulating the BGP import and export policies.

The import policy contain rules specifying which incoming routes the router can install into its RIB table and then FIB table. The export policy specifies which routes can be announced to other peers. The controller can program the import and export policy using NetConf, a network management framework that defines protocol and mechanism for managing configuration and state of networking devices. To learn what routes are available in the network, several mechanisms can be used. The controller can utilise BGP Monitoring Protocol (BMP) to learn all unprocessed incoming routes (i.e. before the import policy). It can also learn routes from the Loc-RIB table via iBGP (with Add-path enabled to allow multiple routes to be sent to the controller) or NetConf if supported. In association with NetConf, the controller can be programmed using YANG data models. For each particular programming task, a suitable YANG module is required. While YANG is a standardised data modelling language, development of YANG modules and support for them vary across organisations and vendors. Thus, to program the import policy, different YANG modules may be required for different router vendors and there is no guaranteed that they are available.

The *SDIRO* architecture, in contrast, advocates for simpler abstractions designed specifically for interdomain routing. It proposes two abstractions: *Path abstraction* and *Path mapping abstraction*. The path abstraction represents an interdomain route as an end-to-end path between an ingress PE router (or a customer connected to it) and a neighbour. It does not specify the details of the path across the internal network. The path mapping abstraction specifies which path should be used for which customer or ingress router. In a *SDIRO* network, the data plane is made up of two layers: *the control layer* which runs on a separate general purpose hardware and *the* *forwarding plane* which is made up of distributed OpenFlow switches. The RIB and FIB tables in a *SDIRO* network can be programmed directly, allowing more control over the routing behaviour of the network. By relying on a standardised OpenFlow, potential problems due to the differences between vendor implementations could be reduced in a *SDIRO* network. A prototype developed in this thesis, the implementation of the controller is called *gRCP* , while *fBGP* implements the data plane layer.

6.2 Method of Evaluation

This section describes the metrics used for evaluating the programming complexity in legacy SDN and *SDIRO*, and the scenarios used in the evaluation.

6.2.1 Complexity Metrics

Typically, the complexity of a software system can be measured using code metrics (e.g. lines of codes, McCabe's Cyclomatic complexity number) or structure metrics (e.g. McClure's Control Flow Metric) [95]. These common metrics either require a fully implemented system (code metrics) or detailed design of interaction between components (structure metrics). To maintain the generality and due to the time constraint, these metrics are not used for evaluation.

Instead, for simplicity, the complexity of a system is qualitatively measured based on the number of protocols (as well as technologies) one would have to work with in order to implement the desired functionality.

6.2.2 Evaluation Scenarios

The evaluation is based on implementations of advanced routing capabilities. Three different scenarios: *Traffic engineering*, *Customer-defined SLA* and *Customer-controlled Routing* used for evaluation, are described below.

Traffic Engineering

Network congestion is common in the Internet, particularly during peak times or big events such as sport events. Rather than relying on a human operator to resolve congestion, a controller that automates the TE activities would be essential to deal with such events. The controller monitors the links and program routers to resolve congestion/violation which occurs while ensuring stability and predictability. For simplicity, only outbound traffic engineering (i.e. traffic towards other peer and provider ISPs) is considered. In practice, inbound and outbound TE can be performed independently.

Customer-defined SLA

Commonly, the transit service is associated with a service-level agreement (SLA) i.e. a technical specification of the service. A SLA defines the characteristics of the service including bandwidth, delay, loss and availability. In practice, a transit provider may have one or more SLAs which are shared by multiple customers. However, SLAs are typically not enforced in interdomain routing because of the limited control the provider has over the end-to-end paths. Therefore, SLAs typically serve the guideline for compensation when the service does not meet the specifications.

A provider can improve its service quality and experience by allowing each customer to define their own SLA and by having a controller actively monitors the interdomain paths and re-programs the network so that the customer's traffic will be forwarded via the path that conforms to the SLA. Regardless of how and where customers are connected, they can independently set SLA for their traffic. Previous study [96, 18] show benefits of customer-defined SLA.

Customer-controlled Routing

Traditionally, it is up to the transit provider how they handle and process routes learned from customers and other networks. The customers have limited ability to influence the operator's decisions. BGP has a built-in mechanism (i.e. the community attribute or MED) which allows a customer to signal the provider about their intention. For example, some standardised community values can be used to tell the provider not to export a route to some ASes.

However, finer-grained controls such as *"export/no-export given some conditions of the network or links"*, cannot be implemented. The ability for the customers to customise the processing of their routes and to make the processing subject to the network status would give the customers an improved quality of service.

Next sections present a detailed implementation of these routing capabilities using the legacy SDN platform and *SDIRO* platform.

6.3 Scenario 1: Traffic Engineering

The operator in this scenario wishes to build a TE controller which can automatically resolve congestion of its transit and peering links with the upstream providers and neighbouring networks.

The TE controller regularly monitors all interdomain links (e.g. in 5 minute period) to collect measurements for the purpose of prediction of link utilisation. Once a congested link is detected, it identifies one or more traffic flows that need to be shifted to the other links. It assumes the desired granularity of a flow is defined as tuple of an ingress PE and a destination prefix. The controller's algorithm is to pick the smallest flow and shift to a cheaper or equal-cost link that can accommodate the flow. If multiple options exist, the controller chooses the closest exit point in terms of IGP distance between the PE and the gateway router.

The next sections presents the design and implementation of the TE controller in the legacy and *SDIRO* network, highlighting major design issues.

6.3.1 Legacy BGP Implementation

This section presents the design of the monitoring component and the programming model of the controller in a legacy network.

How links are monitored?

In the legacy network, link utilisation can be monitored using the simple network management protocol (SNMP). Legacy routers collect link statistics such as number of packets and bytes sent and received by an interface. The controller can poll these statistics by sending a SNMP request with a specific

Object Identifier (OID) to the router. PySNMP [4] is a popular python library for SNMP which can be used to implement the link monitoring component. Since SNMP is a standardised protocol, the same implementation can work seamlessly for all vendors. Listing 6.1 shows an example of SNMP implementation based on the PySNMP library. While the code snippet looks simple enough, the challenge lies in the fact that OIDs are vendor-specific.

```python
from pysnmp.entity.rfc3413.oneliner import cmdgen
def snmp_query(oid, router, secret='public'):
  cmdGen = cmdgen.CommandGenerator()
  _, _, _, varBinds = cmdGen.getCmd(
          cmdgen.CommunityData(secret),
          cmdgen.UdpTransportTarget((router, 161)),
          oid)
  return varBinds
#get number of bytes received by interface Ge0/0 of a router
ge_0_0_in_oct_oid = "1.3.6.1.2.1.2.2.1.17.1"
rcv_bytes = snmp_query(ge_0_0_in_oct_oid, '172.16.1.1')
```

Listing 6.1: Implementation of SNMP query in Python

How flows can be measured?

The controller will need to measure the bandwidth of each flow individually. A set of flows can be decided in advance, e.g. a matrix of ingress PE routers and a set of well-known, popular prefixes. The number of flows in practice can be as large as tens of thousands, depending on the size of the network.

There is no existing mechanism in legacy BGP routers that supports bandwidth measurement of a TE flow. There are two flow-based monitoring mechanisms that can be used for this purposes are *NetFlow* and *sFlow*. NetFlow is available in Cisco platforms whereas sFlow is available

[4]http://snmplabs.com/pysnmp/index.html

in other vendors' devices. These mechanisms collect statistics of flows defined as 7-tuple header fields (i.e. IP address source/destination, transport protocol, source/dest port, TOS and input interface index). This flow definition is too fine-grained for the TE controller. Thus, an aggregation of multiple NetFlow flows is needed in order to compute the bandwidth for a TE flow. This requires a correlation with the routing table and it can be computationally expensive as a prefix can contain tens of thousands of hosts (e.g. a prefix 1.0.0.0/19 has 8192 hosts). For example, if a router produces two NetFlow flows: $f_1 = \{src : 1.0.0.1, dst : 2.0.0.1\}$ and $f_2 = \{src : 1.0.0.2, dst : 2.0.0.2\}$ and its routing table contains two entries: 1.0.0.0/20 $\rightarrow$ 172.16.0.1 (PE) and 2.0.0.0/20 $\rightarrow$ 12.0.0.1 (external), the figure for the flow (172.16.0.1 $\rightarrow$ 2.0.0.0/20 will be the sum of f_1 and f_2.

In implementing the flow measurement component, a flow collector/analyser is required. ntopng [5] offers an open-source solution for NetFlow/sFlow analyser. It provides APIs for external application which can be utilised to implement the TE flow measurement.

Programming Model

In the event of link congestion, the controller first identifies and sorts the flows on the link. It will need to maintain an up-to-date mapping of flows to links (with link as key). In order to determine the alternative link for a flow, the controller needs the routing table of all gateway routers and selects ones that have active route to the prefix. Through a pair-wise comparison, it then can determine the legitimate egress (i.e. the cheapest, closest policy).

To program a router's forwarding table with a new route, the controller

[5]https://www.ntop.org/products/traffic-analysis/ntop/

can either manipulate the import policy or update the router with a superior route update. The former is CPU-intensive and can potentially impact the router's performance because changing the import policy causes the router to recalculate all the affected routes. The later is a lightweight option. By establishing an iBGP session to the router, the controller can program individual routes by sending a BGP update with e.g. higher local-preference. Implementation of this programmer component can be done using ExaBGP, a popular Python library and runtime for BGP.

```
# executed upon each congestion event
def congestion_resolver(link):
  flows = get_all_flows_in_link(link)
  flows = select_and_sort_flows(flows)
  for flow in flows:
    routes = select_available_routes(flow)
    cheaper = select_cheaper_routes(flow.route, routes)
    closest = select_closest(flow.ingress, cheaper)
    if closest is not None and closest != flow.route:
      program_router(flow.ingress, closest)
      if get_congestion_level(link) < THRESHOLD:
        return
```

Listing 6.2: Pseudocode implementing the TE logic of the legacy controller

Listing 6.2 provides a pseudo-code implementing the TE logic for demonstration. It loops through a list of sorted flows on a congested link, and for each flow, it computes the closest route which is cheaper or equal. If such route is found, the route will be programmed to the ingress PE, otherwise the next flow will be considered.

A visual description of the controller's component and architecture are shown in Fig. 6.4. The controller is centralised which can be deployed in a server at a desired POP. The deployment requires with minimal changes

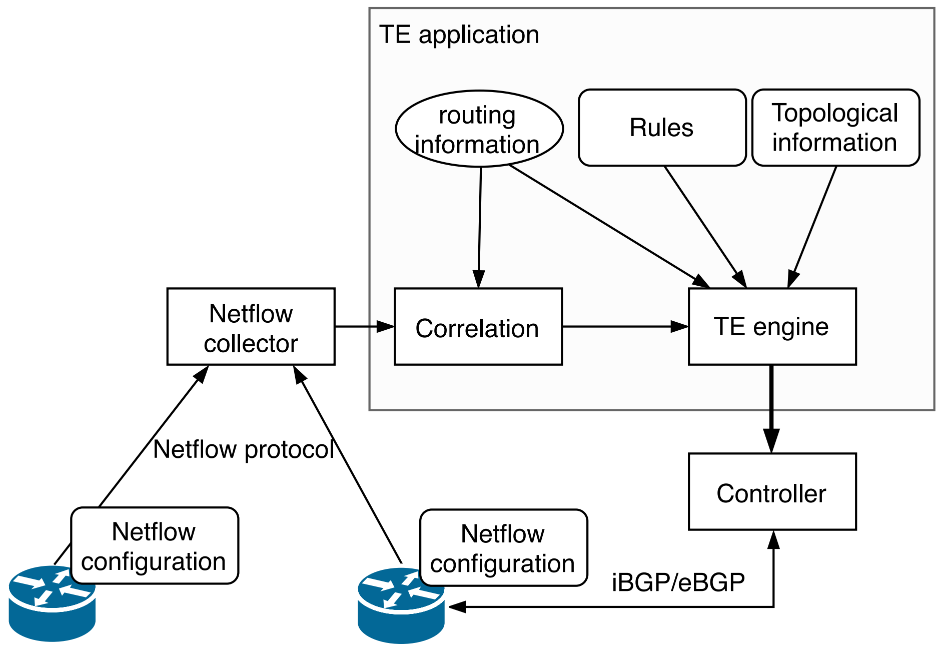


Figure 6.4: The architecture of the TE controller for legacy network.

to the network. NetFlow will need to be enabled on some interfaces (i.e. interface connecting the ingress PE to the gateway routers).

6.3.2 SDIRO Implementation

This section presents the design for the TE controller in a *SDIRO* network.

How links and flows are monitored?

OpenFlow provides a vendor-independent mechanism for network monitoring. OpenFlow protocol specifies that OpenFlow switches offer various types of statistics that the controller can query for by sending appropriate control messages to the switch. The controller can collect *Port statistics* to determine the utilisation and congestion level. It can rely on the same mechanism

```
def is_congested(link):
    if (link.util > THRESHOLD:
        return True
    return False

@listen_ev([LinkStateChange])
def congestion_resolver(self, ev):
    link = ev.link
    if type(link) != InterEgress or not is_congested(link):
        return
    qry = (Mapping.query(Mapping.pathid = link.pathid)
                  .order(Mapping.bandwidth))
    for mapping in qry.fetch(limit=100):
        cur_path = Path.query(Path.pathid = mapping.pathid)
        qry = Path.query(Path.routerid = cur_path.routerid,
                         Path.prefix = cur_path.prefix,
                         Path.inter_util < THRESHOLD)
        new_path = (qry.filter(Path.cost <= cur_path.cost)
                       .order(Path.cost, Path.igp_cost)
                       .fetch(limit=1))
        if new_path:
            topo.create_mapping(routerid=mapping.routerid,
                                prefix=mapping.prefix,
                                path=new_path, update=True)
            if not is_congested(link):
                return
```

Listing 6.3: Implementation of the TE controller in SDIRO

to collect *Flow statistics* to measure bandwidth of a TE flow (A TE flow is represented as an entry in the forwarding table of the data plane). All flows in an OpenFlow switch are associated with several counters for packets and bytes transferred.

In the *SDIRO* architecture, the *gRCP* controller is responsible for maintaining the global network view and representing it in a graph data structure. The TE controller uses the API provided by *gRCP* to discover available paths and their attributes as well as to listen to network events.

Functionality	**Legacy SDN**	**SDIRO**
Link monitoring	SNMP	OpenFlow
Flow monitoring	NetFlow/sFlow	
Programming model	Yang models	Path mapping abstraction

Table 6.1: Programming complexity of Traffic engineering.

Programming Model

The TE controller can be implemented as a control application running on the *gRCP* controller. A fully working implementation of the TE application is shown in Listing 6.3. The application listens for link events (i.e. *LinkStateChange*) and determine if such an event causing network congestion. In case of congestion, it queries for the current flows on a link (lines 11 - 13). Then, for the selected flow to be shifted, it queries for the current path (line 14) and alternative paths considering the policy (lines 15-20) and update the mapping table (line 22).

The programming abstractions provided by *SDIRO* eliminates the programmer from concerns about the underlying mechanisms that are required to make it work. It is noted that, it is possible to realise these abstractions using the mechanisms of the legacy network described above but any maintenance change of underlying hardware could require reprogramming.

6.3.3 Evaluation

Overall, the complexity of the legacy network lies in the fact that various protocols and mechanisms have to be employed for monitoring and control. In contrast, an OpenFlow network provides a uniform way for network monitoring. Moreover, some legacy protocols have vendor-specific components making it hard to switch between vendors. Table 6.1 summarises the protocols and mechanisms used for implementation in the legacy and *SDIRO* network.

6.4 Scenario 2: Customer-Defined SLA

The scenario considered here is that some customers want special treatment for their outgoing traffic. Specifically, the two customers $C1$ and $C2$ want to define their own SLA. It assumes that these customers wish for different paths for traffic to the same set of destination prefixes (e.g. the one who is sending real-time traffic, is desired for low-latency path while the other is interested in high-bandwidth for sending large amount of data).

The high-level abstraction of the system can be depicted as in Fig. 6.5 which shows the process for provisioning customer-defined routing. The SLA specification defined by the customer and the provider's own routing goals are combined as the input to determine the best path for a particular customer. The path is then provisioned in the network.

The next sections discuss the major technical challenges facing the design and implementation of the customer-defined SLA (CDS) controller in legacy and *SDIRO* network.

6.4.1 Legacy BGP Implementation

How paths are monitored?

The support for path performance monitoring varies between vendors. For instance, Cisco provides its own mechanism called *IP SLA*, whereas Juniper

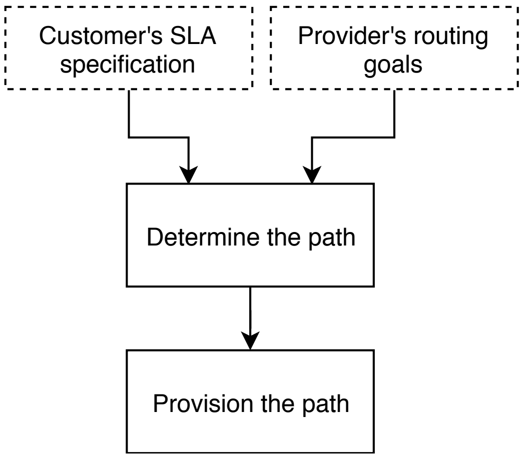


Figure 6.5: The provisioning process for customer-defined routing.

has *RPM* (Realtime Performance Monitor). By enabling IP SLA or RPM, the router periodically sends out probing packets to the configured destination IP addresses. It is required to decide on the IP addresses in advance. For example, to monitor the path to a prefix, one or more hosts belonging to the prefix can be selected and used for enabling SLA. Listing 6.4 shows the configuration to be applied on a router to enable IP SLA. The configuration varies between vendors. On an experiment in a virtualised environment, the Cisco router can only send probing packets via the active route, rendering it useless for monitoring redundant paths.

```
ip sla 6
 icmp-echo 1.0.0.1 source-ip 172.16.0.1
 frequency 300
 request-data-size 28
 tos 160
 timeout 2000
 tag prefix_1.0.0.0_probe
ip sla schedule 6 life forever start-time now
```

Listing 6.4: Configuration snippet for IP SLA in Cisco routers

How customer traffic is routed?

Once a path which meets the customer's requirement is selected, the forwarding plane needs to be programmed appropriately in order to drive the customer traffic to follow the path. Specifically, the gateway router will be programmed so that the selected path becomes active and gets installed into the forwarding table. The ingress router which is connected to the customer is also programmed to make the selected gateway as its exit point.

A challenge arises when two customers are connected to the same ingress PE. Since the PE can maintain a single best path for each prefix, it cannot satisfy both customers who want different paths concurrently. A practical solution to this limitation is to use VRF, a technique that allows multiple routing and forwarding tables to exist concurrently in a router. The controller can be designed such that the default routing table (i.e. the global table) is used for the ordinary transit service, whereas each additional VRF will be used for a different class of service (e.g. low-delay SLA or high-bandwidth SLA). A customer who wants a certain SLA for their traffic will be directed to the appropriate VRF. The high-level design is shown in Fig. 6.6. VRFs are created in the ingress router where the customers are connected and in all selected egress routers. An experiment with Cisco router shows that, either the whole interface is allocated to an VRF (i.e. all traffic) or selected traffic is allocated using policy-based routing. Since customer's requirement is per prefix, PBR is required.

In the same experiment, it appears that Cisco and Juniper routers allow only active routes (from the global table) to be imported to a VRF. This renders routes computed by the local control plane on a router unusable for

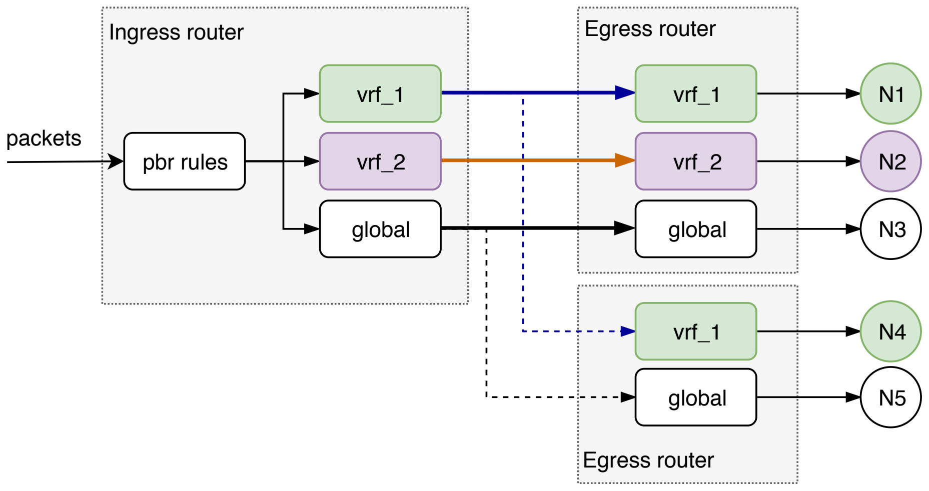


Figure 6.6: Implementation of customer-defined routing in the data plane using VRF and PBR. Each VRF is associated with a class of service offered by the provider. PBR rules used to direct traffic from a customer to a VRF.

this application. To overcome this, the controller will have to program routes in a VRF manually. Specifically, if an egress router learns three routes via BGP (e.g. $r_1 \rightarrow N_1$, $r_2 \rightarrow N_2$ and $r_3 \rightarrow N_3$ as shown in Fig. 6.6) and BGP selects r_3. Then, to install r_1 into vrf_1 the controller crafts a static route and maintains it in according to the updates of the BGP table (i.e. if r_1 is withdrawn by the peer, the controller will need to compute a new static route and update the VRF).

In addition, the experiment also shows that a PBR rule can only match on L3 or L4 headers. This limits the definition of customers to source IP address only. Thus, a knowledge of and maintenance of prefixes belong to a customer will be needed. Implementation of the controller can be achieved using NETCONF and YANG. Listing 6.5 shows a template which may be used for programming static routes in a VRF.

```
<config>
  <routing xmlns="urn:ietf:params:xml:ns:yang:ietf-routing">
    <routing-instance>
      <name>{{ vrf_name }}</name>
      <interfaces/>
      <routing-protocols>
        <routing-protocol>
          <type>static</type>
            <name>1</name>
            <static-routes>
             <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing">
              {% for route in route_list %}
              <route>
               <destination-prefix>{{ route['prefix'] }}</destination-prefix>
               <next-hop><next-hop-address>{{ route['nexthop'] }}</next-hop-address></next-hop>
              </route>
              {% endfor %}
             </ipv4>
            </static-routes>
        </routing-protocol>
      </routing-protocols>
    </routing-instance>
   </routing>
</config>
```

Listing 6.5: Template for static route based on Yang model

Since the current VRF implementation in legacy routers does not allow non-active routes to be imported to a VRF, programming a VRF must be done via static routing. The controller selects a route with the desired characteristics, constructs a static route and installs into the VRF. NETCONF can be used for remote installation of static routes. A YANG model for static routing is needed (a template YANG model is shown in Listing 6.5) which is parsed and transported to the router via NETCONF. An open-source Python library for NETCONF called *ncclient* [6] can be used to implement the static routing component.

Programming Model

```
SLA_SPEC = {
        'SLA1': {'spec': SLA(maxlatency=20), 'vrf': 'VRF1'},
        'SLA2': {'spec': SLA(minbw=100), 'vrf': 'VRF2'},
        'SLA3': {'spec': SLA(maxcost=10), 'vrf':'VRF3'}}

def compute_sla(custid, prefix, sla):
  routes = get_available_routes(prefix)
  sla_spec = SLA_SPEC[sla]
  spec = sla_spec['spec']
  vrf = sla_spec['vrf']
  valid_routes = get_valid_routes(routes, spec)
  if valid_routes:
    best_route = select_cheapest(valid_routes)
    program_network(cust, best_route, vrf)
# Example
compute_sla('10.0.0.1', '1.0.0.0/20', 'SLA1')
compute_sla('10.0.0.2', '1.0.0.0/20', 'SLA2')
```

Listing 6.6: Example of CDS implementation in legacy network

The controller maintains a routing table which maps a prefix to a set of external routes. An additional table is required which associates a route with its SLA characteristics. The controller computes the best route for a customer and a given prefix (i.e. a flow) by looping through all available routes and comparing against the SLA requirement. If multiple options exist, it can choose the best one according to the operator' own policy. A example of the implementation in Python is shown in Listing 6.6. It implements the policy which favours the cheapest route.

[6] https://github.com/ncclient/ncclient

6.4.2 SDIRO Implementation

How paths are monitored?

In OpenFlow, the PacketOut capability which allows the controller to send out arbitrary packets out of an interface, can be utilised to implement the path monitor component. This capability enables the operator to implement various algorithms. This component is not currently implemented in the *SDIRO* prototype.

How customer traffic is routed?

OpenFlow allows a flexible matching scheme. Many different combinations of packet headers can be used. In *SDIRO*, the destination MAC address is used to identify the SLA class a customer wishes for. Matching packets are tagged with a metadata which is then used for matching together with the destination prefix. The (metadata, prefix) implementation is equivalent to VRF in legacy network. By using the destination MAC address, the scalability is improved, as compared to PBR rules in the legacy implementation (i.e. only one rule is used for all customers having the same SLA, compared to multiple rules per customer in legacy network).

Programming Model

The controller's logic is implemented simply by defining path queries that reflect the SLA requirement. Each query is for a class of SLAs. For instance, a filter(Path.bandwidth, Path.delay) assures SLA with minimal bandwidth of 10Mbps and maximum delay of 20ms. The *order()* path is where the operator applies its own objectives. Specifically, the order(Path.cost, Path.distance)

Functionality	**Legacy SDN**	**SDIRO**
Path monitoring	IP SLA/RPM + SNMP	OpenFlow
Customer-specific routing	VRF, PBR	
Programming model	Yang models	declarative language

Table 6.2: Programming complexity of Customer-Defined SLA.

results in the lowest cost path or shortest distance between the ingress and egress point when all paths have the same cost. An example of code snippets is shown in Listing 6.7.

```
def sla_spec1(custid, prefix):
    qry = (Path.query(src=customer.id, dst=prefix)
                  .filter(Path.bandwidth >= 10, Path.delay <= 20)
                  .order(Path.cost, Path.util))
    return qry.fetch(limit=1)

def sla_spec2(custid, prefix):
    qry = (Path.query(src=customer.id, dst=prefix)
                  .filter(Path.bandwidth >= 10, Path.cost <= 100)
                  .order(Path.cost, Path.util))
    return qry.fetch(limit=1)
```

Listing 6.7: Implementation of CDS controller in SDIRO network

6.4.3 Evaluation

While VRF is widely used in ISP networks for Virtual Private Network (VPN) services, it is rather complicated when applied for customer-defined SLA. This is because customer traffic is to be identified and routed at finer granularity. Moreover, programming relies on multiple Yang models. In *SDIRO*, OpenFlow is the single southbound protocol. Its declarative language simplifies the task of provisioning the service. Table 6.2 summarises technologies used in the legacy and *SDIRO* network.

6.5 Scenario 3: Customer-Controlled Routing

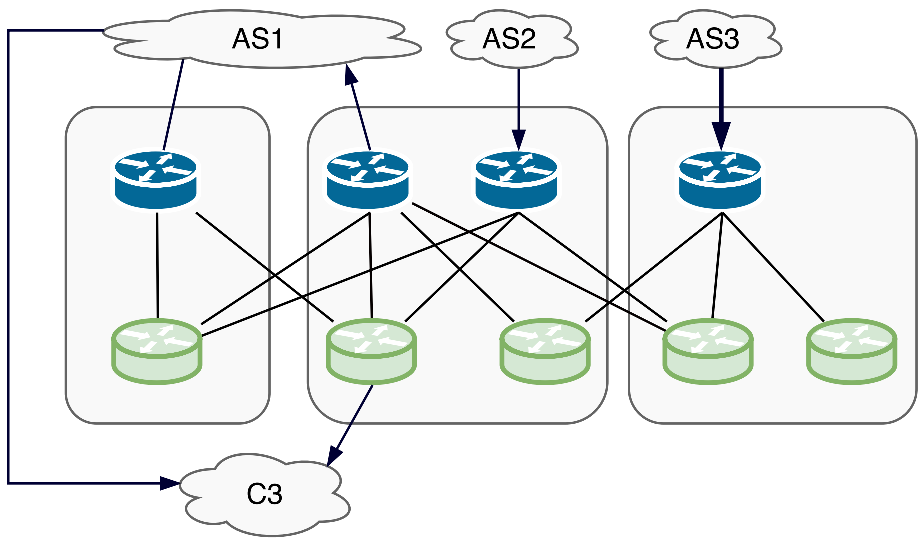


Figure 6.7: Topology demonstrating the CCR scenario. The AcmeISP (middle) has three POPs interconnecting to three ASes. Arrows depicts traffic direction.

The scenario is that a customer C_3 who has two separate transit connections to AcmeISP and AS1, wants to control the incoming traffic to its prefixes. Fig. 6.7 depicts the high-level topology. The majority of incoming traffic to C_3 was originated from $AS3$. Customer C_3 may want to dictate that it does not want to use the link(s) with $AS2$ when the utilisation exceeds 50%. The customer also dictates that traffic from $AS3$ should be split between the its two links if the bandwidth exceeds 60%.

BGP offers very limited capability for inbound routing control. The most effective mechanism is *selective advertisement* which the controller advertises a prefix to only the peers it wants to receive traffic from. Thus, to implement the first policy, the controller withdraws routes to C_3's prefixes previously advertised to AS2. This prevents C2 from sending traffic toward C3's prefixes via AcmeISP.

For the second policy, the edge router(s) used to connect to AS3 will have an additional route (i.e. route via AS1) installed into its forwarding table. That enables the router to balance load between the two links.

The subsequent sections discuss the implementation of the customer-controlled routing (CCR) controller in the legacy and *SDIRO* network.

6.5.1 Implementation on Legacy network

Legacy BGP routers support BGP multi-path routing which allows multiple paths to be used simultaneously for load sharing. However, for a path to be installed into the forwarding table along with the best path, it must have similar attributes to the best path. Specifically, attributes including *local Preference*, *AS path* (both AS number and AS path length), *origin*, *MED* and *IGP metric*. Such requirement obviously restricts the load sharing across two path $AS3 \rightarrow AcmeISP \rightarrow AS1 \rightarrow C3$ and $AS3 \leftarrow AcmeISP \rightarrow C3$ since their AS path attributes and possibly IGP metrics are different. This selection criteria cannot be programmed, therefore such a policy is not technically possible in a legacy network.

6.5.2 Implementation on SDIRO network

Traffic load balancing

In a *SDIRO* network, BGP routing is controlled by a custom BGP software stack running on a dedicated generic server. The *fBGP* prototype built on top of Faucet and ExaBGP allows direct programmability of the RIB table. Thus, it is possible to install any available routes to the forwarding table regardless of their attributes.

In the dataplane, OpenFlow GroupTable feature can be utilised to implement load balancing. To load traffic between the two paths in question, the *fBGP* controller installs them to the FIB table with an action pointing to a group table. This group table is created with *select* type, which allows the switch to select between path using a pre-implemented algorithm (depending on the switch vendor). This capability has not been implemented in the prototype. It, however can be realised through a simple extension to the Faucet controller.

Programming Model

The concept of path mapping in *SDIRO* makes it easy to control routing in this case. The high-level control defines a path mapping between a peer (i.e. $AS3$) and prefix p_1. By turn on and off the path mapping according to a network condition, the controller can implement the desired function. The path mapping abstracts away how the underlying network is constructed. The policy developer/operator will not need to care about.

The feature which supports multiple mappings (i.e. load balancing) is not yet implemented in the *SDIRO* prototype. However, OpenFlow supports the

Functionality	**Legacy SDN**	**SDIRO**
Inbound routing control	NetConf/Yang	OpenFlow
Traffic load balancing	Not supported	
Programming model	Not supported	declarative language

Table 6.3: Programming complexity of Customer-Controlled Routing.

capability to realise it in hardware switches.

6.5.3 Evaluation

Support for customer-controlled routing cannot be practically realised in the legacy network. Table 6.3 summarises the technologies used in the legacy and *SDIRO* network.

6.6 Discussion

6.6.1 SDIRO Provides Better Programmability

Programming in a conventional network is rather complex due to the lack of abstractions. From a system point of view, the transit network is a black box that connects a customer port to a transit port and vice versa. It virtually has a mapping data structure which an external controller can program to make which port should be connected to which port.

SDIRO abstractions and the declarative language make it easy to implement new routing features and policies. The operator focuses on the high-level policy rather than the mechanisms as in the conventional network. *SDIRO* provides better separation of policy and mechanism.

The *path mapping* abstraction simplifies the programming task. The programmer focuses their attention to what traffic (i.e. customer) should be using what path, rather than how this should be done.

The use of OpenFlow opens a lot of possibility to implement advanced routing control and measurements.

6.6.2 SDIRO Allows Better Scalability

Two main scalability issues of the interdomain routing are the growth of the routing table and the routing churn. As shown in the case studies, some routing applications do introduce more routes into the network (i.e. Customer-defined SLA), hence bigger TCAM memories.

There are multiple scalability issues in the conventional network. NetFlow is cpu-intensive. The sample rate can be reduced for scalability at the expense of accuracy. In contrast, the *SDIRO* network relies on OpenFlow counters for network monitoring which can be enabled per entry in the forwarding table. This effectively allows to collect statistics of traffic for each prefix (which is enough for the TE application, whereas Netflow is too fine-grained). The frequency of querying the counters can have negative impact on the switch performance. However, various query techniques can be used (e.g. adaptive query) and the frequency can be increased (e.g. TE application needs not to control the network too frequent than tens of minutes or even hours). *SDIRO* is built based on the open-source controller Faucet whose Gauge is the monitoring. Currently, gauge simply sends a query periodically to get counters for all available flow entries at once, which can be an issue if there are hundreds of thousands of flows.

6.6.3 SDIRO Improves Network Manageability

Firstly, the *SDIRO* architecture introduces fewer protocols and technologies than the conventional approach when implementing new routing applications. Although the technologies such as NetFlow, VRF and PBR are commonly used in ISP networks, their removal will definitely make the network less complex hence simplifying the management.

Secondly, the *SDIRO* architecture has clear separation of software and hardware, thus simplifying many common management tasks such as software and hardware upgrade. *fBGP* runs on separate hardware which makes it easier to upgrade without impacting the forwarding plane. In the legacy network, router replacement is visible to the external network. The physical link and the eBGP session are terminated in the same router. In a *SDIRO* network, eBGP sessions are terminated at *fBGP* server which is running on a separate hardware from the switches where the physical connections are connected. Thus, it enables the configuration where the customer's router is connected to *fBGP* using two physical links, terminated at two different switches. Thus, shutting down a switch for replacement reduces bandwidth capacity but does not disrupt the service.

Third, having the control software running on commodity hardware, allows the operator to add more feature and adjust for their needs. Modifications to the router software is limited to the customisation or configuration allowed by the router vendor. The typical wait time for a new feature can be several years. *SDIRO* on the other hand, is an open-source and based on reliable open-source code bases including Faucet and Neo4j. *fBGP* is based on Faucet which can freely available and can be extended. It is well documented and has growing community.

6.7 Summary

This chapter evaluates and compares two architectural approaches to advanced routing capabilities in a transit ISP: legacy SDN and OpenFlow-based SDN. Overall, OpenFlow makes it easier to realise new routing functionality by giving the operator more control over the forwarding plane.

Chapter 7

Conclusions

The Internet has become an essential part of our lives. Innovation has been and is being very active at the edge of the Internet, such as access technologies, applications and content delivery networks. However, the Internet core which connects the edge networks together remains the same when it was invented. The key routing protocol BGP has not significantly improved, although much research has been paid attention to. The thesis argues that the key obstacle to interdomain routing problems is the deployability of the solution.

That motivates this research to investigate the design of a deployable solution. SDN and OpenFlow facilitate the development of such a solution. SDN breaks the ossification in the Internet by decoupling the control and data plane. SDN offers flexibility and programmability to the network. These are the key capabilities employed in this research to develop an incrementally deployable solution for interdomain routing, *SDIRO*.

The high-level architecture of *SDIRO* is inspired by various previous work on routing control platform. While *SDIRO* does not significantly diverge from other SDN architectures, in designing *SDIRO* the research addresses several unique challenges in interdomain routing including programmability, flexibility and scalability. Specifically, contributions made in this research can be summarised as follows.

Chapter 3 presented the design of a routing control platform, *gRCP* which enables routing programmability in transit ISPs. The design utilises graph data model and graph database to represent interdomain routing information and the design of a query language for routing programming. *gRCP* design also takes into account the scalability and availability.

Chapter 4 addressed the inflexibility of interdomain routing in the current networks through the design of *fBGP*, a programmable border router based on OpenFlow. The main contribution is to show how the flexibility and programmability of OpenFlow are utilised to implement the multipath routing capability *fBGP* while ensuring scalability. Other contribution is to evaluate whether implementing BGP functionality on top of OpenFlow provides equivalent performance to conventional BGP router.

Chapter 5 tackled one of the key problem concerning ISP operators: scalability, particularly the scalability of the data plane. *FIBO* is the proposed solution which addresses the problem by decomposing the forwarding table and distributing it across an interconnected network of switches. The main contribution is a mathematical model of the problem and the formation of the problem as an integer linear problem.

Chapter 6 offered a closer look at the advantages of using centralised control architecture and OpenFlow through a comparison between *SDIRO* and a legacy SDN architecture, *NetCONF*.

In conclusion, the proposed solution for interdomain routing addresses three key technical requirements that are believed to be of importance for transit operators: programmatic routing control, flexible routing policy and scalability. However, like much previous research in the literature, the design does not take into account the business requirements. In addressing the problems of interdomain routing, it is important to have the views of the transit ISPs and other stakeholders such as CDNs. Very limited information on the literature shows what problems are critical to their business and the Internet community as a whole. Moreover, what functional and non-functional requirements are needed by the transit operators and their customers. Thus, a survey about the operators' view on this matter would be essential for researchers to develop the solution that fits for purpose. Moreover, more information about routing policy and management in transit ISPs would be needed for enhancing the practicality of the solution and better evaluations.

Appendix A

Acronym

API	Application Programming Interface
AS	Autonomous System
ASBR	Autonomous System Border Router
BFD	Bidirectional Forwarding Detection
BGP	Border Gateway Protocol
BMP	BGP Monitoring Protocol
CDN	Content Delivery Network
CLI	Command Line Interface
ECMP	Equal-Cost Multiple Path
FIB	Forwarding Information Base
ForCES	Forwarding and Control Element Separation
I2RS	Interface to Routing System
IP	Internet Protocol
ISP	Internet Service Provider
IXP	Internet Exchange Point
LFB	Logical Function Block
LLDP	Link Layer Detection Protocol

LSP	Label Switch Path
MAC	Media Access Control
MED	Multi-Exit Discriminator
MPLS	Multiprotocol Label Switching
NTP	Network Time Protocol
OAM	Operation Administration and Maintenance
ONF	Open Networking Foundation
ORTC	Optimal Routing Table Constructor
OSPF	Open Shortest Path First
PBR	Policy-Based Routing
PCC	Path Computation Client
PCE	Path Computation Element
PCEP	PCE Protocol
PE	Provider Edge
POP	Point of Presence
QoS	Quality of Service
RCP	Routing Control Platform
RIB	Routing Information Base
RIR	Regional Internet Registries
RTT	Round-Trip Time
RPSL	Routing Policy Specification Language
SDIRO	SDN for Interdomain Routing
SDN	Software-Defined Networking
SLA	Service Level Agreement
SDX	Software-defined Internet Exchange
SNMP	Simple Network Management Protocol
TCAM	Ternary Content Addressable Memory

TE	Traffic Engineering
TED	Traffic Engineering Database
TCP	Transmission Control Protocol
VRF	Virtual Routing and Forwarding
VPN	Virtual Private Network

Bibliography

[1] Y. Wang, I. Avramopoulos, and J. Rexford, "Design for configurability: Rethinking interdomain routing policies from the ground up," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 3, pp. 336–348, 2009.

[2] N. Feamster, H. Balakrishnan, and J. Rexford, "Some foundational problems in Interdomain routing," in *In HotNets, 2004*, pp. 41–46, 2004.

[3] J. Chandrashokar and J. Krasky, "Limiting path exploration in BGP," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 4, pp. 2337–2348, IEEE, 2005.

[4] A. Lara, A. Kolasani, and B. Ramamurthy, "Network Innovation using OpenFlow: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 493–512, 1 2014.

[5] Y. Jarraya, T. Madi, and M. Debbabi, "A Survey and a Layered Taxonomy of Software-Defined Networking," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 1955–1980, 1 2014.

[6] Y.-C. Chiu, B. Schlinker, A. B. Radhakrishnan, E. Katz-Bassett, and R. Govindan, "Are We One Hop Away from a Better Internet?," in *Proceedings of the 2015 ACM Conference on Internet Measurement Conference - IMC '15*, (New York, New York, USA), pp. 523–529, ACM Press, 10 2015.

[7] B. Bhattacharya and D. Das, "SDN based architecture for QoS enabled services across networks with dynamic service level agreement," in *2013 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pp. 1–6, IEEE, 12 2013.

[8] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena, and M. F. Magalhães, "Virtual routers as a service: the routeflow approach leveraging software-defined networks," *Proceedings of the 6th International Conference on Future Internet Technologies*, vol. 1, pp. 34–37, 2011.

[9] P. Lin, J. Hart, U. Krishnaswamy, T. Murakami, M. Kobayashi, A. Al-Shabibi, K.-C. Wang, and J. Bi, "Seamless interworking of SDN and IP," *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 475–475, 9 2013.

[10] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. Raszuk, "Revisiting routing control platforms with the eyes and muscles of software-defined networking," *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12*, p. 13, 2012.

[11] A. Gupta, E. Katz-Bassett, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, and R. Clark, "SDX: a software defined internet exchange," in *Proceedings of the 2014 ACM conference on SIGCOMM*, vol. 44, pp. 551–562, ACM, 8 2014.

[12] M. Winther, "Tier I ISPs: What They Are and Why They Are Important," *IDC White Paper*, no. May, pp. 1–13, 2006.

[13] P. Verkaik, D. Pei, T. Scholl, A. Shaikh, A. C. Snoeren, and J. E. V. D. Merwe, "Wresting Control from BGP : Scalable Fine-grained Route Control," in *USENIX Annual Technical Conference*, pp. 295–308, 2007.

[14] Y. Wang and J. Rexford, "Morpheus : Making Routing Programmable," in *SIGCOMM workshop on Internet network management*, pp. 285–286, ACM, 2007.

[15] S. Lee, T. Wong, and H. S. Kim, "To automate or not to automate: On the complexity of network configuration," *IEEE International Conference on Communications*, pp. 5726–5731, 2008.

[16] A. Voellmy, H. Kim, and N. Feamster, "Procera: a language for high-level reactive network control," *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 43–48, 2012.

[17] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: simplifying SDN programming using algorithmic policies," *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pp. 87–98, 2013.

[18] W. Xu and J. Rexford, "MIRO : Multi-path Interdomain ROuting," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 171–182, 2006.

[19] X. Zhao, D. J. Pacella, and J. Schiller, "Routing Scalability: An Operator's View," *IEEE Journal on Selected Areas in Communications*, vol. 28, pp. 1262–1270, 10 2010.

[20] Q. Li, D. Wang, M. Xu, and J. Yang, "On the scalability of router forwarding tables: Nexthop-Selectable FIB aggregation," *Proceedings - IEEE INFOCOM*, pp. 321–325, 2011.

[21] D. Massey, L. Wang, B. Zhang, and L. Zhang, "A scalable routing system design for future internet," in *Proc. of ACM SIGCOMM Workshop on IPv6*, 2007.

[22] J. L. Sobrinho and F. Le, "A fresh look at inter-domain route aggregation," in *2012 Proceedings IEEE INFOCOM*, pp. 2556–2560, IEEE, 3 2012.

[23] P. Mihăilă, T. Bălan, R. Curpen, and F. Sandu, "Network Automation and Abstraction using Python Programming Methods," *MACRo 2017 - 6th International Conference on Recent Achievements in Mechatronics, Automation, Computer Science and Robotics*, vol. 2, no. 1, pp. 95–113, 2017.

[24] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica, "Pathlet Routing," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, (New York, NY, USA), pp. 111–122, ACM, 2009.

[25] X. Yang, D. Clark, and A. W. Berger, "NIRA: A new inter-domain routing architecture," *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, pp. 775–788, 2007.

[26] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen, "SCION: Scalability, Control, and Isolation on Next-Generation Networks," in *2011 IEEE Symposium on Security and Privacy*, pp. 212–227, IEEE, 5 2011.

[27] P. Faratin, D. Clark, P. Gilmore, S. Bauer, a. Berger, and W. Lehr, "Complexity of Internet Interconnections : Technology , Incentives and Implications for Policy," *Proc. of Telecommunications Policy Research Conference*, pp. 1–31, 2007.

[28] E. Gregori, A. Improta, L. Lenzini, and C. Orsini, "The impact of IXPs on the AS-level topology structure of the Internet," *Computer Communications*, vol. 34, pp. 68–82, 1 2011.

[29] V. Giotsas, M. Luckie, and B. Huffaker, "Inferring Complex AS Relationships," *Proceedings of the 2014 Conference on Internet Measurement Conference*, pp. 23–29, 2014.

[30] A. Ghezzi, M. Dramitinos, A. Rangone, E. Agiatzidou, F. T. Johanses, R. Balocco, and H. Løsethagen, "Internet interconnection techno-economics: A proposal for assured quality services and business models," *Proceedings of the Annual Hawaii International Conference on System Sciences*, pp. 708–717, 2014.

[31] V. Giotsas, C. Dietzel, G. Smaragdakis, A. Feldmann, A. Berger, and E. Aben, "Detecting Peering Infrastructure Outages in the Wild," *Proceedings of the Conference of the ACM Special Interest Group on Data Communication - SIGCOMM '17*, pp. 446–459, 2017.

[32] N. Spring, R. Mahajan, and T. Anderson, "The causes of path inflation," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '03*, (New York, New York, USA), p. 113, ACM Press, 8 2003.

[33] S. Y. Qiu, P. D. McDaniel, and F. Monrose, "Toward valley-free interdomain routing," in *IEEE International Conference on Communications*, pp. 2009–2016, 2007.

[34] V. Giotsas and S. Zhou, "Valley-free violation in Internet routing - Analysis based on BGP Community data," in *Communications (ICC), 2012 IEEE International Conference on*, pp. 1193–1197, 6 2012.

[35] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush, "A measurement study on the impact of routing events on end-to-end internet path performance," *ACM SIGCOMM Computer Communication Review*, vol. 36, p. 375, 8 2006.

[36] M. Caesar and J. Rexford, "BGP routing policies in ISP networks," *IEEE Network*, vol. 19, pp. 5–11, 11 2005.

[37] S. Savage, A. Collins, E. Hoffman, J. Snell, T. Anderson, S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The end-to-end effects of Internet path selection," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication - SIGCOMM '99*, vol. 29, (New York, New York, USA), pp. 289–299, ACM Press, 1999.

[38] J. H. Park, R. Oliveira, S. Amante, D. McPherson, and L. Zhang, "BGP route reflection revisited," *IEEE Communications Magazine*, vol. 50, no. 7, 2012.

[39] D. Walton, A. Retana, E. Chen, and J. Scudder, "Advertisement of multiple paths in BGP," tech. rep., RFC 7911, 2016.

[40] E. R. Raszuk, R. Fernando, K. Patel, D. McPherson, and K. Kumaki, "Distribution of Diverse BGP Paths (RFC 6774)," tech. rep., RFC 6774, 2012.

[41] V. V. D. Schrieck, P. Francois, and O. Bonaventure, "BGP Add-Paths : The Scaling / Performance Tradeoffs," *EEE Journal on Selected Areas in Communications*, vol. 28, no. 8, pp. 1299–1307, 2010.

[42] S. Uhlig and B. Quoitin, "Tweak-it: BGP-based interdomain traffic engineering for transit ASs," *NGI 2005 - Next Generation Internet Networks: Traffic Engineering*, vol. 2005, pp. 75–82, 2005.

[43] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, pp. 114–119, 2 2013.

[44] D. Kreutz and F. Ramos, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[45] N. Mckeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, J. S. Turner, and S. Louis, "OpenFlow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[46] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," in *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia on - CEE-SECR '13*, (New York, New York, USA), pp. 1–6, ACM Press, 10 2013.

[47] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow switching: Data plane performance," *IEEE International Conference on Communications*, no. April, 2010.

[48] T. Limoncelli, "OpenFlow: A Radical New Idea in Networking," *Communications of the ACM*, vol. 55, no. 8, pp. 42–47, 2012.

[49] F. Hu, Q. Hao, and K. Bao, "A Survey on Software Defined Networking (SDN) and OpenFlow: From Concept to Implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. c, pp. 1–1, 2014.

[50] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 1617–1634, 1 2014.

[51] R. Dantu, T. A. Anderson, R. Gopal, and L. L. Yang, "Forwarding and control element separation (ForCES) framework," tech. rep., RFC 3746, 2004.

[52] J. Ash and A. Farrel, "A path computation element (PCE)-based architecture," tech. rep., RFC 4655, 2006.

[53] D. King and A. Farrel, "A PCE-based Architecture for Application-based Network Operations," 2014.

[54] F. Paolucci, F. Cugini, A. Giorgetti, N. Sambo, and P. Castoldi, “A Survey on the Path Computation Element (PCE) Architecture,” *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1819–1841, 2013.

[55] S. H. Adara and R. W. Verisign, “Software-Defined Networks and the Interface to the Routing System (I2RS),” *IEEE Internet Computing*, vol. 17, pp. 84–88, 2013.

[56] J. Schönwälder, M. Björklund, and P. Shafer, “Network configuration management using NETCONF and YANG,” *IEEE Communications Magazine*, vol. 48, no. 9, pp. 166–173, 2010.

[57] M. Bjorklund, “YANG-a data modeling language for the network configuration protocol (NETCONF),” tech. rep., RFC 6020, 2010.

[58] K. Phemius and M. Bouet, “Monitoring latency with OpenFlow,” in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pp. 122–125, IEEE, 10 2013.

[59] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, “PayLess: A low cost network monitoring framework for Software Defined Networks,” in *2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–9, IEEE, 5 2014.

[60] S. Agarwal, M. Kodialam, and T. V. Lakshman, “Traffic engineering in software defined networks,” in *2013 Proceedings IEEE INFOCOM*, pp. 2211–2219, IEEE, 4 2013.

[61] D. Drutskoy, E. Keller, and J. Rexford, “Scalable network virtualization in software-defined networks,” *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, 2013.

[62] Z. Chen, J. Bi, Y. Fu, Y. Wang, and A. Xu, “MLV: A Multi-dimension Routing Information Exchange Mechanism for Inter-domain SDN,” in *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, pp. 438–445, IEEE, 11 2015.

[63] N. Feamster, J. Rexford, and E. Zegura, “The road to SDN: an intellectual history of programmable networks,” *ACM SIGCOMM Computer Communication Review*, vol. 44, pp. 87–98, 4 2014.

[64] A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente, and D. Villela, “A survey of programmable networks,” *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 2, p. 7, 1999.

[65] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," *2nd conference on Symposium on Networked Systems Design & Implementation*, pp. 15–28, 5 2005.

[66] I. Juniper Networks, "Fundamentals of Egress Peering Engineering," 2017.

[67] J. Van der Merwe, H. Nguyen, M. Nguyen, A. Ramarajan, S. Saad, M. Satterlee, T. Spencer, D. Toll, S. Zelingher, A. Cepleanu, K. D'Souza, B. Freeman, A. Greenberg, D. Knight, R. McMillan, D. Moloney, and J. Mulligan, "Dynamic connectivity management with an intelligent route service control point," *Proceedings of the 2006 SIGCOMM workshop on Internet network management - INM '06*, pp. 29–34, 2006.

[68] S. Vissicchio, L. Cittadini, and G. Di Battista, "On iBGP routing policies," *IEEE/ACM Transactions on Networking*, vol. 23, no. 1, pp. 227–240, 2015.

[69] N. Kumar and G. Saraph, "End-to-End QoS in Interdomain Routing," in *International conference on Networking and Services (ICNS'06)*, pp. 82–82, IEEE, 2006.

[70] T. Griffin, T. Li, D. Massey, C. Vogt, J. Wang, and L. Zhang, "Scaling the internet routing system: An interim report," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 8, pp. 1233–1237, 2010.

[71] H. Ballani, P. Francis, T. Cao, and J. Wang, "Making routers last longer with ViAggre," *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, p. 453–466, 2009.

[72] P. Skoldstrom and B. C. Sanchez, "Virtual aggregation using SDN," *Proceedings - 2013 2nd European Workshop on Software Defined Networks, EWSDN 2013*, pp. 56–61, 2013.

[73] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable Flow-Based Networking with DIFANE," *ACM SIGCOMM Computer Communication Review*, vol. 40.4, pp. 351–362, 2010.

[74] H. Liu, "Routing prefix caching in network processor design," in *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, pp. 18–23, IEEE, 2001.

[75] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging Zipf's law for traffic offloading," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 1, p. 16, 2012.

[76] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, “Infinite CacheFlow in software-defined networks,” *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN ’14*, pp. 175–180, 2014.

[77] F. Dürr and T. Kohler, “Comparing the Forwarding Latency of OpenFlow Hardware and Software Switches,” Tech. Rep. 2014/04, Fakultät Informatik, Elektrotechnik Informationstechnik, Univ. Stuttgart, Stuttgart, Germany, 2014.

[78] N. Kang, Z. Liu, J. Rexford, and D. Walker, “Optimizing the ”one big switch” abstraction in software-defined networks,” in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, (New York, New York, USA), pp. 13–24, ACM, 12 2013.

[79] Y. Kanizo, D. Hay, and I. Keslassy, “Palette: Distributing tables in software-defined networks,” in *2013 Proceedings IEEE INFOCOM*, pp. 545–549, IEEE, 4 2013.

[80] H. Huang, S. Guo, P. Li, B. Ye, and I. Stojmenovic, “Joint Optimization of Rule Placement and Traffic Engineering for QoS Provisioning in Software Defined Network,” *IEEE Transactions on Computers*, vol. 64, no. 12, pp. 3488–3499, 2015.

[81] C. Pelsser, A. Masuda, and K. Shiomoto, “Scalable support of interdomain routes in a single AS,” in *GLOBECOM - IEEE Global Telecommunications Conference*, pp. 1–8, 2009.

[82] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, “Onix: a distributed control platform for large-scale production networks,” *OSDI*, vol. 10, pp. 1–6, 10 2010.

[83] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, and B. Lantz, “ONOS: towards an open, distributed SDN OS,” *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN ’14*, pp. 1–6, 2014.

[84] R. Anwar, D. Choffnes, and E. Katz-bassett, “Investigating Interdomain Routing Policies in the Wild,” *Imc’15*, pp. 71–77, 2015.

[85] G. Kahraman and S. Bilgen, “A framework for qualitative assessment of domain-specific languages,” *Software and Systems Modeling*, vol. 14, no. 4, pp. 1505–1526, 2015.

[86] J. Park, P.-c. Cheng, S. Amante, D. Kim, D. McPherson, and L. Zhang, “Quantifying i-BGP Convergence inside Large ISPs,” tech. rep., Technical report, UCLA, 2011.

[87] S. K. Singh, T. Das, and A. Jukan, "A Survey on Internet Multipath Routing and Provisioning," *IEEE Communications Surveys & Tutorials*, vol. 17, pp. 2157–2175, 1 2015.

[88] S. Tao, K. Xu, Y. Xu, T. Fei, L. Gao, R. Guerin, J. Kurose, D. Towsley, and Z. L. Zhang, "Exploring the performance benefits of end-to-end path switching," in *Proceedings - International Conference on Network Protocols, ICNP*, pp. 304–315, 2004.

[89] S. Sharma and M. Nourani, "A TCAM-Based Parallel Architecture for High-Speed Packet Forwarding," *IEEE Transactions on Computers*, vol. 56, no. 1, pp. 58–72, 2007.

[90] C. R. Meiners, A. X. Liu, E. Torng, and J. Patel, "SPliT: Optimizing space, power, and throughput for TCAM-based classification," in *Proceedings - 2011 7th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2011*, pp. 200–210, 2011.

[91] O. Bonaventure, C. Filsfils, and P. Francois, "Achieving Sub-50 Milliseconds Recovery Upon BGP Peering Link Failures," *IEEE/ACM Transactions on Networking*, vol. 15, pp. 1123–1135, 10 2007.

[92] R. Draves, C. King, S. Venkatachary, and B. Zill, "Constructing optimal IP routing tables," in *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, vol. 1, pp. 88–97, IEEE, 1999.

[93] S. Denazis, E. Haleplidis, J. H. Salim, O. Koufopavlou, D. Meyer, and K. Pentikousis, "Software-defined networking (SDN): Layers and architecture terminology," *RFC 7426*, vol. IRTF, 2015.

[94] J. Stringer, D. Pemberton, Q. Fu, C. Lorier, R. Nelson, J. Bailey, C. N. A. Correa, and C. E. Rothenberg, "Cardigan: SDN distributed routing fabric going live at an Internet exchange," in *2014 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–7, IEEE, 6 2014.

[95] D. Kafura and G. R. Reddy, "The Use of Software Complexity Metrics in Software Maintenance," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 3, pp. 335–343, 1987.

[96] V. Kotronis, X. Dimitropoulos, R. Klöti, B. Ager, P. Georgopoulos, and S. Schmid, "Control Exchange Points: Providing QoS-enabled End-to-End Services via SDN-based Inter-domain Routing Orchestration," *LINX*, vol. 2429, no. 1093, p. 2443, 2014.