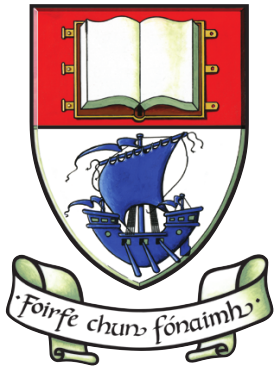
MaaS: An application-centric, dynamic, end-to-end QoS framework utilizing SDN Federation

(MPLS-as-a-Service)



Sidhant Hasija

School of Science and Computing

Waterford Institute of Technology

This dissertation is submitted for the degree of

Masters of Science by Research

Supervisors: Dr. Rashid Mijumbi and Dr. Steven Davy

September 2018

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Masters of Science, is entirely my own work and has not been taken from the work of others save to the extent that such work has been cited and acknowledged within the text of my work.

Sidhant Hasija
September 2018

Acknowledgements

Firstly, I would like to acknowledge with sincere gratitude the supervision provided by my advisory team – *Dr. Rashid Mijumbi and Dr. Steven Davy* who kept their confidence and supported me throughout the project, and helped in getting back on path when strayed. There was a lot gained from their immense knowledge, advice and research approach that went into this Thesis from the onset, and without which it would not have been possible.

Equally boosting was the all-round support and help provided by ENL Unit Manager *Dr. Alan Davy* majorly in guiding the research and project direction as well as being pro-actively supportive of training requirements. I would also want to acknowledge the support of *Dr. Brendan Jennings* for valuable assistance on the related publication, and for the opportunity to be a part of SOLAS training in SDN domain, and to *Dr. Lei Shi* also for that purpose. Hard to miss out also is the help provided by *Infrastructure Group* at TSSG be it in the network design for developing testbed or being receptive of data-centre resource requirements

I would also like to extend my heartfelt gratitude to *TSSG, Waterford Institute of Technology, Science Foundation Ireland and Cisco Systems (Galway)* for providing the opportunity to work on this research project, the funding support and access to many generic skills training.

Last but not the least is the throughout guidance, support and solicited as well as unsolicited help of friends around, similarly from colleagues specially *Dr. Bernard Butler* and being motivated from their professionalism and passion for science.

Finally I would like to thank my family back home who were the continuous source of motivation and drive.

Abstract

Internet is a collection of many distributed autonomous network domains. These domains deploy only limited control services amongst each other, most notably Border Gateway Protocol (BGP), primarily due to understandable security, integrity and management reasons. This construct can thus be attributed to hindering in development of powerful inter-domain control services that perform end-to-end network service provisioning such as QoS.

The benefits though of having a dynamic end-to-end Quality of Service (QoS) provisioning service for multi-domain networks are in tuning to real-time application resource requirements. The need for such an application-centric QoS framework is now more than ever as Internet traffic is continuously increasing and ability to provision network service in real-time at user application level would result in efficiency saving on under and over-provisioning.

Existing end-to-end QoS frameworks such as IntServ and Multi-protocol Label Switching (MPLS) lack dynamicity primarily attributed to distributed *domain control-plane* apart from also being hindered in deployment due to lack of activity in *inter-domain control-plane* as discussed previously. This Thesis proposes solution in form of a dynamic application-centric QoS framework utilizing Software Defined Networking (SDN) breakthrough. SDN controller is proposal of a centralized and thus dynamic *domain control-plane*, while collaboration between such per-domain control-planes referred to as SDN Federation enables ground for securing and enriching *inter-domain control-plane* as well, potentially improving upon the dynamicity limitations of legacy frameworks.

The proposed framework named MPLS-as-a-Service (MaaS), is an implementation of prominent Internet Traffic Engineering (TE) framework - MPLS in SDN controller. The resultant SDN-MPLS control-plane made dynamic by SDN (Federation) enables network operators to swiftly provision QoS Label Switched Path (LSP)s and offer MPLS as a service model compared to present day long-term contracts. The complete design of MaaS is provided in the thesis, while the design of supporting underlying SDN-IP-BGP control-plane is also considered. This highlights a stand-out feature of the proposal as current research direction in end-to-end QoS provisioning is from outside domain control-plane orchestrated

by third-party vendors, while we propose a native and in-network solution courtesy of this SDN-MPLS-IP-BGP control-plane.

An easy to provision multi-domain network testbed is also created through virtual devices. MaaS and the underlying SDN control-plane is developed as well as demonstrated on this testbed. Apart from the dynamic inter-domain QoS property, MaaS is equipped with more adaptive error-handling through a service renegotiation mechanism and scalability enhancement measures over legacy MPLS which as well are demonstrated on the testbed.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

1.1.1 Application-Centric QoS

Internet from its rather humble beginnings has evolved into a ubiquitous resource largely impacting every aspect of human life. This drastic increase in embracement of Internet is evident if we consider that the catalogue of user-applications empowered by Internet has increased from World Wide Web for accessing static documents remotely and E-Mail services, to much more demanding services such as real-time audio-video communication systems, video streaming, IPTV, cloud computing, among others. This trend and utilization is expected to continue as next-generation of users come online, and new network-intensive applications are being developed for increasing user needs and satisfaction such as Internet of Things, Mobile Edge Computing, Content Delivery Networks, 4K and 8K video streaming.

To keep up with and further foster this innovation, Internet networks also need to evolve. These multitude of applications will have different network resource requirements in terms of bandwidth and delay, and varying level of sensitivity to degraded network service in terms of jitter, packet-loss, etc. Therefore, it is important for existing network resource allocation approaches to evolve so as to more finely provision resources at an application level, and to adapt such resources to dynamic and real-time changes in application requirements over time. A significant challenge herein is for network control and data-plane to adjust with the dynamicity imposed by application-level provisioning, as well as scaling up to support per-flow QoS provisioning. Emergence of Software Defined Networking (SDN) and Network Function Virtualization (NFV) however provide an opportunity to solve these challenges.

Moving towards application-centric and dynamic model of QoS would enable on-demand network service provisioning and thus flexibility in terms of mitigating under-provisioning and over-provisioning limitations. It will also lead to increase in revenue and optimization of network resources for service providers, realizing the long cherished goal of service-aware networks and network-aware services.

1.1.2 Inter-Domain QoS Challenge

Network control planes have seen significant advancement over the years, most notably in our context Software Defined Networking (SDN), which opens up the network to application control enabling dynamic configuration and resource allocation. What still fall behinds in development is the inter-domain control plane, which has long been a bottleneck for end-to-end uniform service provisioning in Internet. The nature of this limitation is due to the construct of Internet, which is an inter-connection of many authoritative and autonomous network domains. Although this distributed construct has contributed immensely to the outreach of Internet as we know today, it has restricted the development of novel powerful inter-domain services (like end-to-end QoS) as domains deploy only limited control protocols between themselves (most notably BGP) due to security, management and scalability concerns.

There has always been a movement to devise new services, applications, and architecture to provision end-to-end inter-domain QoS. While MPLS-TE, Intserv and Diffserv Bandwidth Brokers are some of the early solutions at lower layers of networking, following are some of the relatively modern-day movements.

1. **SD-WAN**: Software-Defined Wide Area Network (SD-WAN) aim to provide MPLS like WAN or QoS benefits over public broadband Internet for connectivity between different enterprise remote sites at a cost lower than MPLS, making it suitable for the needs of small retailers and SME customers. It provides gateway customer premises equipment (virtual or physical) for every site, centrally manages them, and thereafter dynamically load balances customer's inter-domain or WAN traffic between all the available connections (DSL, Ethernet, LTE, MPLS) while monitoring the links and keeping a track of their capacities (citrix:sd-wan).

2. **Content Distribution Networks (CDN)**: While not directly a framework to provision end-to-end QoS, content providers are increasingly distributing their data to user edge, thus preventing the need for serving users from a distant central data-centre that would have posed stringent network QoS requirements (cloudfare:cdn).

3. **Decline in Transit**: Also not a framework to provision inter-domain service, increasingly many network service providers are peering together at Internet Exchange Points (IXP) handing over the traffic to the destination domain directly or shortening the transit to it, thus immensely simplifying end-to-end QoS problem (InternetSociety, 2018, p. 5).

1.1.3 A Network-native QoS provisioning service

The recent advancements in inter-domain QoS such as SD-WAN, CDNs and IXPs were effected by dedicated vendors, applications, or at administrative ends. However a more sustainable, dynamic and general-purpose solution would be to develop and deploy more advanced network-native end-to-end QoS provisioning frameworks i.e built into the underlying domain network control-plane, similar to IntServ and MPLS. This would be a more empowering approach as network operators or service providers would be able to directly offer services to application customers, enterprises, or different other operators and providers without the need of third-party vendors like SD-WANs.

Such a native in-network end-to-end QoS service would be fitting into the distributed design of Internet, be scalable and made available to much wider class of consumers. Recent networking enhancements with SDN, NFV and increasing inclination towards open-standard and open-APIs provide an opportunity to solve the security, management and interoperability challenges for such an in-network solution.

1.1.4 Connect-Cisco Work Package

Connect is an Irish research centre for networks, communications, and the Internet of things, funded under the Science Foundation Ireland (SFI). The agenda to CONNECT includes research projects, referred to as target projects, defined in collaboration with an industry-partner wherein the objectives, research-plan and activities are defined as work-packages. TSSG-WIT in collaboration with Cisco, defined a CONNECT targeted project, the focus being to improve operations of Unified Communication (UC) or Enterprise Collaboration Systems (ECS) such as Cisco Spark or WebEx utilizing SDN innovation. Specifically, Work Packages 2 and 3 of the project concentrated on the following aspects, quoting-

1. How SDN can interoperate, across domain borders, ensuring the integrity of the network substrate from a security perspective, as well to ensure that flow specs, QoS etc. can be specified, configured, monitored, and enforced end to end.

2. How ECS can perform when operating across domains leveraging SDN technology, including the possibility of that the ECS is being deployed across domain borders, off-premises, on-premises and with Cloud hosted solutions,

3. Design a network controller protocol to synchronize off-premises / on-premises network controller policies.

4. Using OpenDaylight as the network-programming platform of choice to prototype the demonstrators of the use cases defined.

This Master's Thesis was motivated and set-off by this Work Package (WP). The motivation to look into inter-domain QoS challenges using SDN innovations was borrowed from this WP, as well as the dynamic application-centric property courtesy of and generalized from ECS/UC focus in order to initially concentrate on networking aspects.

1.1.5 Problem Statement

The following statement summarizes the problem incorporating all the requirements stated previously.

"To investigate or devise an inter-domain end-to-end QoS provisioning framework that is network-native and hence in-network as well as general-purpose; application-centric and hence dynamic plus agile, utilizing SDN innovations."

1.1.6 Advantages

Such a dynamic QoS provisioning service should support on-demand inter-domain bandwidth allocation for even SME customers or applications like communication service providers. It should also further foster inter data-centre connectivity and be helpful to provision bandwidth for scientific research networks that exchange large volumes of data. For bigger enterprises or network service provider domains, it can support on-demand Wide Area Network (WAN) bandwidth during instances of peak or increased demand, or link failover saving on over-provisioned backup links. Such a dynamic framework would thus majorly advance the cause of end-to-end WAN QoS even for smaller customers available to bigger players until now.

1.2 Background and Limitations

1. **Intserv, Diffserv, RSVP, and Bandwidth Broker**: Collectively, these technologies can be considered as the very first end-to-end QoS framework of Internet. Intserv at an application's dynamic request can provision end-to-end QoS per-flow even inter-domain. Intserv is primarily aided by Resource Reservation Protocol (RSVP) providing (bandwidth) service-request communication between application hosts and end-to-end network devices to install service. Scalability concerns of Intserv especially in the core of Internet prefer deployment of Diffserv which provisions QoS for rather aggregate flows grouped by marked Class of Service (CoS). Different Diffserv or Intserv domains can interoperate to effect end-to-end uniform service provisioning through deploying a domain centralized request enforcement entity like Bandwidth Broker (Nichols et al., 1999, p. 14).

2. **MPLS**: Similar to how Intserv can reserve end-to-end resources for every flow (*generally source or destination IP prefix*), thus creating end-to-end connections also known as Virtual Circuit (VC), MPLS can create such VCs specific to a 20-bit Label referred to as LSPs or Tunnels. The state among the involved switching devices for a LSP VC is similarly created through RSVP-TE.

Even though both Intserv and MPLS can perform inter-domain end-to-end resource-reservation at an application's dynamic request thus guaranteeing WAN QoS, Label-switching boasts features such as better SLA support through traffic aggregation, as incoming packet flows requiring similar network treatment can be mapped to a common Label at the edge, and thus made to traverse the same LSP tunnels providing identical performance, and many other features (Awduche et al., 1999, p. 8).

However, if we imagine MPLS ideal to be our desired dynamic end-to-end QoS provisioning framework, limitations are imposed in respect that its control-plane is distributed among many different entities and hence rigid (PCECC, 2016). Therefore, it takes provisioning time to the tune of days for network service providers to offer MPLS VC services and also for upgrading service in the future, which makes it suitable and economical for only large enterprise customers having longer term QoS target/needs (velocloud, 2018). Thus despite supporting application-centric or per-flow QoS provisioning by protocol and being an native in-network framework true to our desires, MPLS control-plane is not dynamic enough to be deployed as a general-purpose real-time QoS solution for a wide range of application customers.

1.3 Solution: MPLS-as-a-Service

In response to the problem statement, this thesis proposes a solution framework named MPLS-as-a-Service shortened to MaaS. Inspired by the Traffic Engineering or QoS provisioning capabilities of legacy MPLS and trying to mitigate its rigidity issues, the framework is a proposal of a rather dynamic MPLS control-plane designed to empower on-demand provisioning of inter-domain LSP tunnels. True to our requirements, the framework is dynamic enough to provision QoS at an application's request by way of creating LSPs in real-time as the name suggests, and similar to MPLS and unlike other inter-domain QoS initiatives, it is delivered natively from within the network control-plane argued to be essential for a sustainable and widespread solution.

Primarily, SDN and SDN Federation breakthroughs are incorporated for transition of MPLS into a dynamic and manageable framework in the form of MaaS. Following is a brief overview of how these primary enablers are utilized, followed by the features of resultant MaaS framework.

1.3.1 SDN

Analogous to how emergence of cloud computing provided scalable computing powers for end-users, SDN envisages a similar breakthrough by bringing forth additional scalable computing capabilities for network control. This is achieved by separating control-plane from network data-plane, and implementing it in an application called SDN controller which remotely interfaces with the data-plane. An important proposition of controller is the network control abstraction layer (Fig. 1.1), providing one-stop access to configure the underlying data-plane as well view its operational state.

The proposed framework - MaaS implements a new centralized MPLS control-plane in SDN controller as shown in Fig. 1.1. SDN controller enables richer interaction between different network control and management entities as well as the data-plane via control abstraction layer, thus providing an agile operational environment for MaaS to dynamically provision QoS guaranteed LSP virtual circuits. This empowers network operators with a dynamic MPLS control-plane and offering MPLS-as-a-service model, compared to the distributed and rigid legacy one.

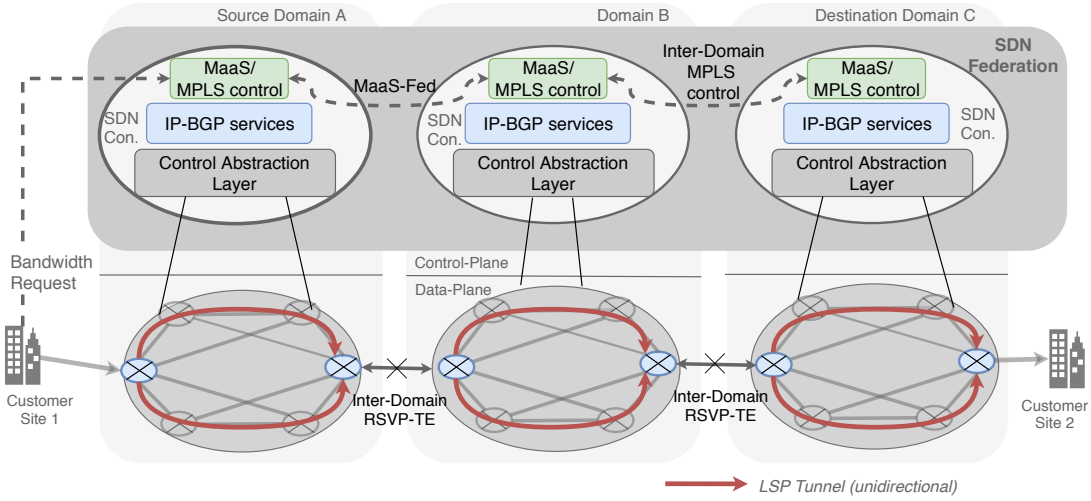


Fig. 1.1 Overview of SDN and SDN Federation as enablers of MaaS.

1.3.2 SDN Federation

Inter-domain MPLS control-plane specifically concerns itself with providing collaboration support for setting up QoS LSP tunnels over multiple domains (Fig. 1.1). This component has been identified of not finding widespread deployment due to following issues.

- *Security:* Its primarily protocol - inter-domain RSVP-TE functions between neighbouring domain border routers (Fig. 1.1), and as a result aggravates security concerns exposing the network to any rogue request from outside (Johnston et al., 2011, p. 8).

- *Management:* Deploying RSVP-TE inter-domain requires explicit peering agreements to be negotiated as for any other inter-domain service, which is a manual, offline and administratively taxing task (Dugeon and Meuric, 2017, p. 3).

SDN controller is a movement towards programmable and centralized network control-plane (Fig. 1.1). Extending this concept, collaboration between different domain SDN controllers referred to as SDN Federation can bring about a programmable and dynamic inter-domain/end-to-end control-plane as well, although distributed (Fig. 1.1). This newer inter-domain control-plane being orchestrated by neighbouring SDN controllers can provide rather secure and manageable grounds for inter-domain MPLS operations, compared to legacy inter-domain control-plane orchestrated by comparatively meagre neighbouring border routers which gives rise to limitations as above. We exploit this enabler to design and implement a new service-request protocol called MaaS-Fed to support in MPLS control process of end-to-end LSP setup, as a replacement of inter-domain RSVP-TE (Fig. 1.1).

1.3.3 MaaS Overview

MaaS is developed and delivered as a SDN controller plugin or service (specifically Opendaylight) per domain as shown in Figure 1.1. In its normal mode of operation, upon receiving a bandwidth request from an application customer through its offered REST APIs, source-domain MaaS plugin utilizes underlying SDN-IP control layer to provision local segments of LSP circuits to tunnel customer traffic (Fig. 1.1). Thereafter, it identifies a subsequent domain through underlying SDN-BGP control layer that can further carry LSP tunnels to destination endpoint, and propagates LSP service-request to this next domain. This communication between neighbouring domain MaaS entities happens through a new RSVP like service-request protocol called MaaS-Fed (Fig. 1.1), which is enabled by SDN Federation as briefed earlier. Resultantly through this distributed collective action, end-to-end LSPs are provisioned in each of the domain between customer's remote endpoints.

Apart from these primary *application-centric, dynamic and inter-domain e2e QoS* features, following other features are possessed or built into the proposed framework MaaS.

1. *Native and in-network:* MaaS works closely with the underlying SDN-IP-BGP control layer of controller (Fig. 1.1). This highlights the advantage of native in-network approach towards end-to-end QoS provisioning as the service in charge (MaaS) can function coherent, closely and in the same realm as other local network control processes. Benefits are in the increased agility of provisioning process than if delivered from outside the network like in case of SD-WAN applications from where rest of the domain control information (like IP-BGP) is not easily accessible.

2. *Generic BGP inter-domain interface:* BGP is the only known widely deployed protocol between autonomous network domains of Internet. Deploying any other protocol would require explicit agreements to be negotiated, while also aggravating security risks and complexity of inter-domain control plane. In this Thesis, generalization of BGP is performed by utilizing BGP update message to transmit messages of other inter-domain protocols as well (MaaS-Fed in our case). Facilitated by programmability of SDN controller which make possible extension of BGP, this keeps the simplicity of inter-domain control plane intact and potentially enables online negotiation for deploying newer service protocols over this generic BGP inter-domain interface.

3. *Adaptive:* The proposed inter-domain LSP service-request protocol (MaaS-Fed) in being orchestrated by neighbouring domain SDN controllers is able to possess better error-handling capabilities then compared to its legacy counter-part – RSVP, which is

orchestrated by meagre network border routers (Fig. 1.1). We thus utilize this flexible interface between controllers to make the QoS process more *adaptive* in scenarios of resource-unavailability by establishing an online *service re-negotiation* mechanism.

4. *Scalability provisions:* Providing LSP creation services at the granularity of even smaller enterprise customers would mandate preparing network to handle large number of provisioning requests, hence posing a scalability challenge in terms of processing (control-plane) and state (data-plane). While SDN control-plane working now outside the confines of a physical data-plane device is scalable by way of being deployed as a controller cluster, there's still a practical limit to the LSP state forwarding devices can efficiently maintain. To somewhat resolve this scalability and LSP state explosion problem, MaaS has provisions for pushing some of the forwarding state to outside of MPLS network domains into customer premises. The state that can be offloaded specifically is the Label Forwarding Equivalence Class (LFEC) that maps IP or other flows to a LSP.

5. *Agile:* Legacy MPLS had the following issues which limited its agility – (i) utilization of latency inducing hop-by-hop RSVP-TE to transmit LSP state among devices, (ii) each network devices allocates its own local Label for a LSP, and (iii) control-plane rigid as distributed among switching-devices (LDP, RSVP) and dedicated control entities (PCE). Contrary to these, proposed SDN-MPLS control plane (MaaS) has direct interface into switching-devices to upload LSP state, allocates a domain-global Label for a LSP segment, and all the control-entities are centralized courtesy of SDN controller. These all improvements collectively progress the agility of MPLS.

1.3.4 Testbed

- *Testbed control-plane:* As evident from the overview above and Figure 1.1, MaaS or our SDN-MPLS plane is dependant on the services of underlying SDN-IP-BGP network Control Abstraction Layer (CAL) in controller. However there's no available off-the-shelf controller distribution that can readily provide such services. Therefore, as part of Testbed setup, we create such a control-plane ourselves in the form of new, existing and modified plugins of Opendaylight (ODL) SDN controller platform as shown in Figure 1.2. These layers apart from providing service to MaaS to help in LSP setup, also perform other essential tasks to keep the underlying data-plane functioning, importantly end-to-end IP routing.

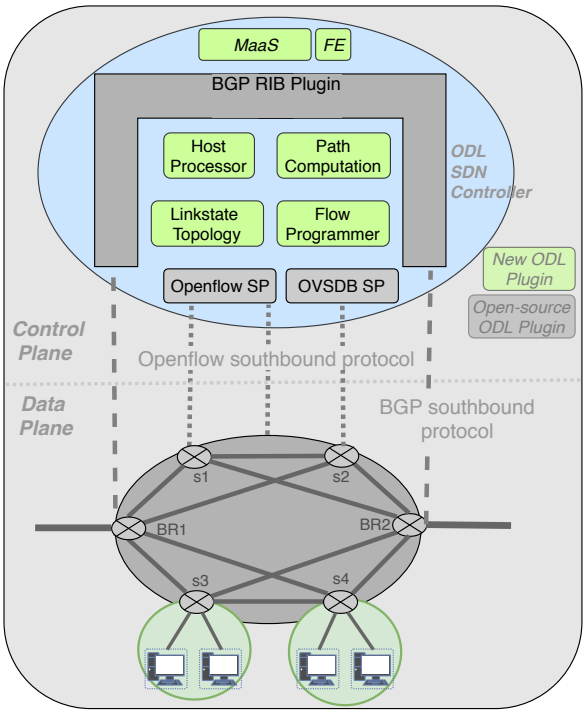


Fig. 1.2 Custom plugins developed for the Opendaylight (ODL) SDN control-plane of our Testbed (green).

- *Testbed data-plane:* Apart from the domain SDN control-plane, we also require a domain data-plane in our Testbed to create LSPs on, and multiple of such domains to minimally resemble an Internet like multi-network construct. Such testbed forwarding devices should essentially be SDN ready for which we use Openflow capable Open-vswitch (ovs), and the entire multi-domain testbed should be easy to setup and tear for which we utilize Vagrant automation tool among others.

While the MaaS framework is explained in detail in chapter 3, such a state-of-the-art multi-domain testbed is detailed in subsequent chapter 4. Its development undertook significant effort, and represents an important outcome of this Thesis work. MaaS is also experimentally demonstrated for its features on this testbed in later chapter 5.

1.4 Contributions

In our endeavour to devise a dynamic end-to-end QoS provisioning framework using SDN paradigm shift, we propose a new SDN-MPLS control-plane (MaaS) which is designed to achieve objectives by creating inter-domain LSPs in real-time. Survey of background works,

public literature and new similar frameworks indicate towards the lack of a distributed and in-network approach to inter-domain QoS utilizing SDN enhancements. The proposal fills this gap, and contributes in other related aspects as listed below-

- Complete design and development of the proposed SDN-MPLS control-plane or MaaS framework on an open-source (Opendaylight) SDN controller platform. It is designed to work closely with the incumbent IP-BGP control planes of Internet although in the SDN context, and the design and development of this underlying SDN-IP-BGP control-plane also considered.
- Apart from dynamic and application-centric end-to-end QoS provisioning, limitations of the existing MPLS deployments other than rigidity also analysed to further devise and incorporate enhancements in MaaS namely - generic BGP interface, adaptive service re-negotiation mechanism, and scalability measures by facilitating delegation of mapping forwarding entries into customer premises.
- A state-of-the-art multi-domain Openflow-SDN testbed is designed that has provisions for automated rapid setup and tear-down, in order to develop and demonstrate MaaS on.

To the best of our knowledge, there's no other work that proposes and demonstrates a multi-domain SDN-MPLS-IP-BGP control-plane as MaaS and underlying SDN controller services do. In terms of the comprehensiveness of the work, the objective of this Thesis was to at first develop an end-to-end functioning QoS framework, which is achieved. In future, individual components such as optimizing resource allocation approaches and analysing scalability will be looked into in a top-down approach.

Chapter 2

Background, Related Works and Limitations

2.1 Introduction

Provisioning QoS end-to-end has long been considered by Internet research and engineering communities. In trying to simplify and generalize those works, they can be divided into 2 broad categories - centralized and distributed. The distinction between these two school of thoughts is owed to the employed inter-domain service-request interface, which arguably is the most important component of such a framework for it bypasses domain boundaries to communicate a customer's network service requirement from a source endpoint domain to intermediate domains and eventually to a destination endpoint domain. While the individual domain control planes play the role of local QoS orchestration entities (Fig. 2.1), this interface acts as a glue between these heterogeneous networks to help establish uniform end-to-end QoS provisioning. Fig. 2.1 example network schematic shows these two interfaces deployed simultaneously, although only one of them is generally employed.

In this chapter, an overview of related research works and industry frameworks is presented, grouped into above described centralized (mostly SDN related) and distributed QoS frameworks (both legacy and SDN incorporating). Special emphasis and detailed overview of MPLS-TE is provided due to its significance in the proposed solution. Limitations of the discussed works in relation to our problem statement are also presented and summarized.

The background overview of end-to-end QoS provisioning framework in this chapter is only presented from the control-plane perspective. Appendix C however presents a brief general overview on how data-plane nodes like switches and routers achieve or actuate QoS for different class of services or application flows.

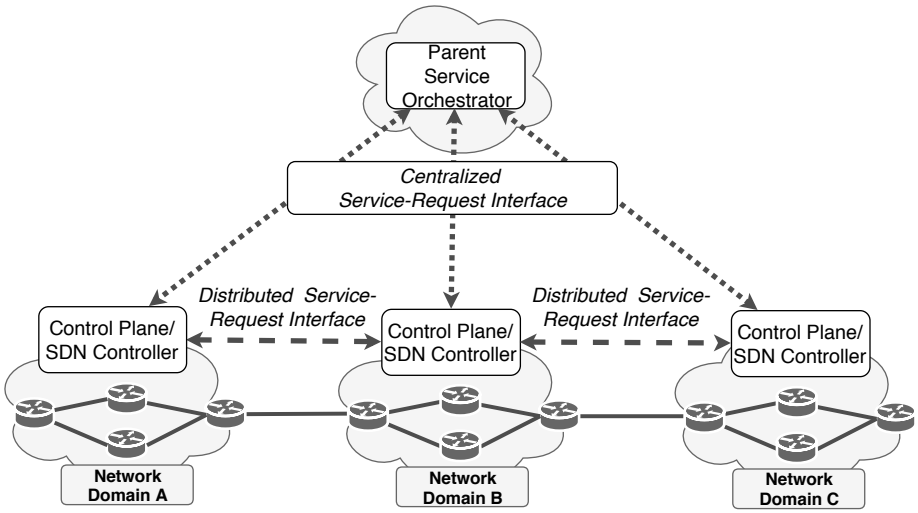


Fig. 2.1 Centralized as well as distributed inter-domain service-request interfaces.

2.2 Centralized multi-domain service provisioning

A defining feature of this category of frameworks is the presence of a centralized parent control entity over individual network domains that orchestrates end-to-end QoS in the participating domains (Fig. 2.1). RTM-SDN framework, as explained below, is a recent example that embodies design tenets of general centralized service provisioning.

2.2.1 RTM-SDN

Developed by International Multimedia Telecommunications Consortium's (IMTC) RTM-SDN activity group, the framework aims at evolving QoS provisioning mechanism for enterprise VoIP traffic from a static one to a dynamic and real-time process by proposing a standardized northbound SDN interface offering from networks to VoIP applications. Developed specifically for UC systems which are the current flagbearer of Real Time Media (RTM) communications, its initial specification proposes an Automated QoE Service (AQS) in SDN controller which provides APIs for UC applications to publish information like call numbers, priority and statistics. Thus presented with real time application state, AQS employs Dynamic DiffServ Code-Point (DSCP) Marking, Call Admission Control and Class of Service Traffic Engineering to provision network QoS dynamically for upcoming or ongoing VoIP/UC session flows (RTM-SDN Group, 2014). This specification was picked up early by UC vendors, and Microsoft Skype for Business SDN Interface was one of the first implementations (Microsoft, 2016).

While the initial version of RTM-SDN considered only intra-domain or local QoS provisioning, a latter specification catered to the wider multi-domain use-case specific to cloud hosted UC systems referred to as UCaaS (RTM-SDN Group, 2017). Its end-to-end QoS provisioning process is explained briefly below in the coherently labelled Fig. 2.2 and following sequence by way of an ideal scenario.

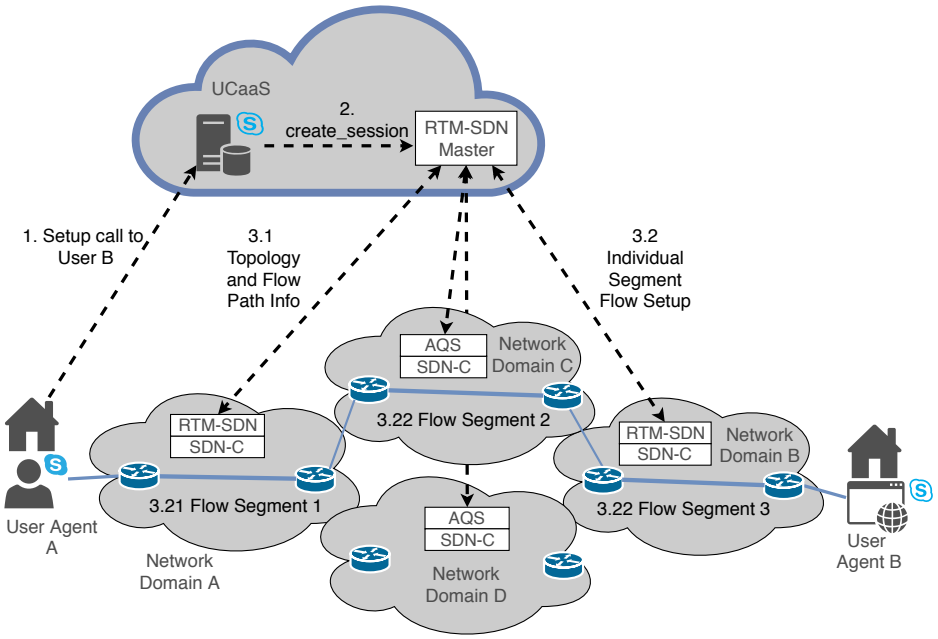


Fig. 2.2 Scenario demonstrating RTM-SDN framework provisioning QoS in real time for UCaaS.

1. UC user or VoIP flow-agent A signals to its server in the cloud (UCaaS) in setting up a call to an outside domain agent B.

2. UC server does its basic function of establishing a SIP/RTM communication session between the two endpoints, and similar to Skype for Business's SDN interface, also publishes this information to a *master* AQS service through its *create_session* API. This particular entity is referred to as master because it is a parent hierarchical entity that orchestrates the function of individual domain RTM-SDN instances.

3. Upon reception, master utilizes underlying domain Automated QoE services (AQS) to setup an end-to-end QoS path.

    3.1. Firstly, master has to decide on a fitting domain level path for the flow. Since it does not have any network control information by itself, the framework also

proposes underlying domains to provide topology and path-computation services for the master to deduce an inter-domain path, a process named as *decompose*.

3.2. Master thereafter uses underlying domain AQS APIs to provision segments of flow path in every involved domain with requisite service levels. The process named *delegate* once finished, provisions end-to-end QoS for application flow.

In the above framework, while master is the face and implementation of inter-domain service-request interface as defined earlier, different SDN-controllers or AQS instances are the local service orchestration entities.

2.2.2 Other Related Centralized Works

Placement and delivery of end-to-end services from a third-party parent entity is a defining feature of all centralized provisioning frameworks, with some of the other investigated works described below.

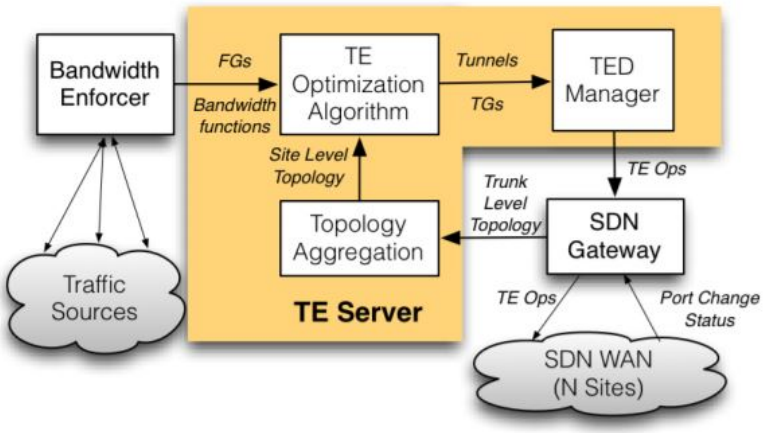


Fig. 2.3 B4 SD-WAN architecture, image from (Jain et al., 2013, p. 5)

.

- B4 (Jain et al., 2013) is a design and implementation of Google's SD-WAN, connecting dozens of its private data-centre domains distributed worldwide, and the applications deployed over which have massive and elastic bandwidth requirements like data replication, index pushes & asynchronous data copies accounting for 90% of internal traffic. Each of the B4 data-centre houses an on-site SDN controller managing the local wide area routing devices (*SDN WAN*, Fig. 2.3), while a global TE application allocates and provisions end-to-end bandwidth for competing applications (*TE Server*, Fig. 2.3).

- NaaS (Duan et al., 2015) proposes an abstraction model for network domains to offer desired services externally to interested parties in a Network-as-a-Service (NaaS) type of approach like cloud systems. It also proposes a Service Delivery Platform (SDP) which acts as a middleware between end-user applications and multiple NaaS domains to provision services such as end-to-end real-time QoS. NaaS abstraction helps in representation of heterogeneous network domains as a uniform resource, while SDP is the manifestation of a centralized parent control entity as shown in Fig. 2.4.

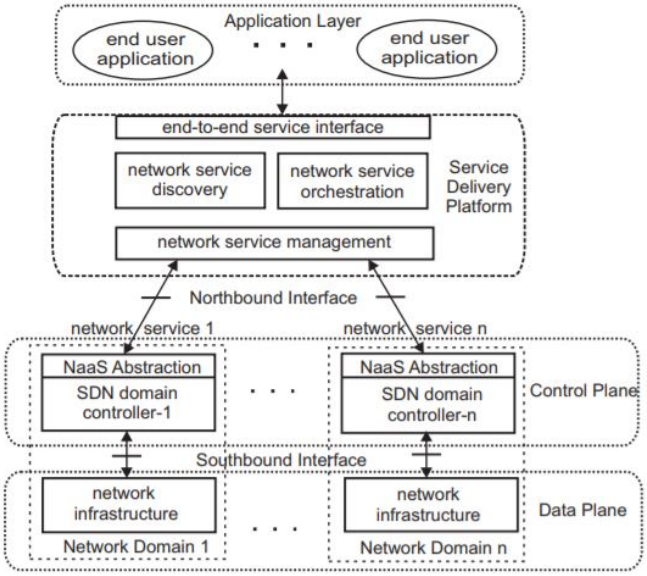


Fig. 2.4 NaaS abstraction model and SDP labelled architecturally, image from (Duan et al., 2015, p. 8).

2.2.3 Limitations

RTM-SDN, B4, NaaS and other similar centralized frameworks are naturally suited for dynamic and thus application-centric QoS provisioning. They are scalable in respect that parent orchestration entities are a one-hop gateway to all network domains, and optimal as these entities have access to federation wide network view at once. However, they have a few inherent limitations as below.

- **Security and Confidentiality:** Underlying individual network domains, via their control-planes (SDN controller) are required to offer sensitive network information to a third-party parent QoS orchestrator, *RTM-SDN master* as an example, for it to deduce domain-level topology and other information essential for end-to-end provisioning.

This firstly raises security and integrity concerns and secondly, optimal application of such topology and path information is difficult remotely as there is layer of local complex management policies that applies to them.

- **Lack of in-network property:** Inter-domain QoS routing function delivered from an external parent entity is decoupled in this process from other standard network functions, like local routing and QoS. However, network domain administrators arguably desire tighter administration and control over outside-domain QoS functions as well and make them coherent and subject to same policy control as other local network processes, herein referred to as *in-network* property.

2.2.4 Summary

Centralized frameworks are thus suited for environments where there is significant degree of trust between involved network-domains, like geo-distributed data canters as in case of Google B4. However from the perspective of different heterogeneous network service domains, an *in-network* inter-domain QoS framework which can function in the same realm and closely with other local network processes without the presence of any parent entity would be rather ideal from the perspective of security as well as ease of management and operation.

2.3 Distributed multi-domain service provisioning

Distributed QoS frameworks are characterized by employing a distributed protocol, like RSVP-TE, for conveying a network service-request hop-by-hop from source endpoint domain to the destination domain, contrary to the centralized frameworks employing a third-party entity for similar duties as we saw previously. Fig. 2.1 shows both the frameworks component simultaneously. Following is an overview of different distributed multiple-domains QoS provisioning frameworks, with a significant emphasis on MPLS which has been the premier distributed TE framework of Internet.

2.3.1 MPLS-TE

Multi-protocol Label Switching - Traffic Engineering (MPLS-TE) is employed by network service providers in offering bandwidth-guaranteed WAN links to enterprises for connectivity between their multiple branch sites and cloud. Contrary to a connectionless switched IP path,

MPLS provisions end-to-end connections or explicitly routed paths between endpoints which are identified and switched on the data-plane by a 20-bit Label. Referred to as LSP, they encapsulate and transport IP or any other network protocol packet in a 32-bit Label header between Layer 2 and Layer 3 and therefore are also referred to as MPLS tunnels or VC.

Following is a brief overview of MPLS Traffic Engineering, its control plane in general, and limitations in relation to our problem statement.

A. MPLS Traffic Engineering Advantages

TE concerns itself with improving network operations on broadly two ends- optimizing network resources and meeting application QoS/SLA targets, and thus is highly correlated to QoS provisioning. Traffic Engineering is at the core of MPLS and the prominent reason for its deployment in many production networks (Awduche et al., 1999, p. 8). Foremost, it provides the ability to setup explicitly routed paths which help in achieving deterministic behaviour by ensuring that end-to-end QoS is maintained. Although more IP friendly frameworks such as Intserv provide similar abilities, Label switching favours traffic aggregation as incoming packets requiring similar network treatment can be mapped to a common LFEC at the edge, and thus core of the network needs to maintain only a single (Label) forwarding state to serve entire traffic aggregate. Labels provide a convenient abstraction of QoS paths in the control-plane, thus LSPs can be efficiently maintained, and also it is easier to switch on Labels than IP addresses at data-plane. Other powerful services such as Segment Routing and VPN are facilitated by MPLS albeit encapsulation and decapsulation overhead during the flow flight.

B. MPLS Control Plane

MPLS control plane is the logical service orchestration entity in charge of provisioning LSPs to accommodate an enterprise's WAN request. Its different components and functions are briefed below as also coherently labelled in Fig. 2.5 in order to overview its functioning.

- *Computation:* Its most important function is Path Computation (PC) - to find out a suitable path through the network between desired endpoints that also has resources available to meet the Service Level Agreements (SLAs) negotiated with customers. For its computational intensity, this function is generally instead carried out by a non-data plane entity, known as PC Element (PCE). Further, a standard PCE Protocol (PCEP) is available for routers and network management systems, referred to as PC Client

(PCC), to communicate with PCE and request a QoS path between certain endpoints (2, Fig. 2.5).

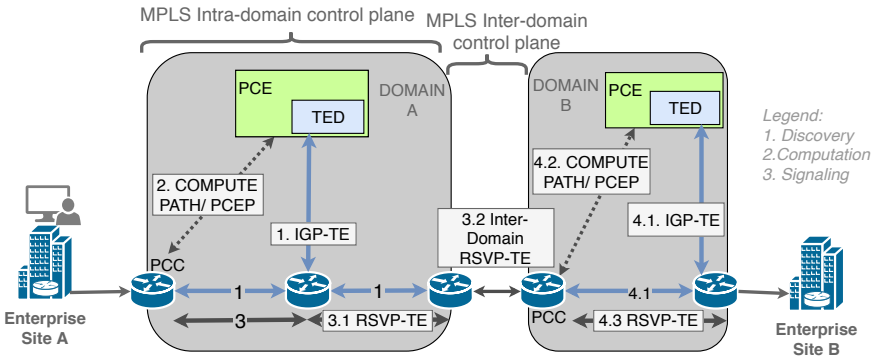


Fig. 2.5 Legacy MPLS control-plane with different components and functions labelled.

- *Discovery:* PCE requires updated network state information to operate on which is maintained by a Traffic Engineering Database (TED). Source of this information for TED is through peering with rest of the network control-plane, for instance, via Interior Gateway Protocol (IGP) routing such as Open Shortest Path First (OSPF-TE) (OSPF-TE, 2008) or Intermediate System to Intermediate System (ISIS-TE) (ISIS-TE, 2008) that disseminate TE state along with routing and topology (1, Fig. 2.5), or even through BGP enhanced for similar purposes- BGP-LS (2016).

- *Signalling:* Lastly, the path computed by PCE is installed and actuated on the data-plane in a process commonly referred to as signalling performed by a service-request protocol, RSVP being the de-facto standard for which (3.1, Fig. 2.5). This is also when a Label is assigned for the path by each data-plane forwarding device, communicated to the neighbouring upstream/downstream node through one of the protocols like Label Distribution Protocol (LDP), BGP-LU or RSVP itself, and they update their Label Forwarding Information Base (LFIB) accordingly.

C. MPLS inter-domain control plane

While MPLS control-plane described above is akin to a local QoS orchestration entity, further as also argued previously, an inter-domain service-request protocol is required to provision TE-LSPs and thus QoS over multiple domains. Inter-domain extension to RSVP is available for this task. Briefly, source domain computes and signals the local segment of large LSP tunnel crossing itself via the MPLS control plane discussed above, and then propagates the service request to next domain through RSVP-TE deployed between border nodes (3.2,

Fig. 2.5). Subsequent domains repeat this function through their own control-plane until the LSP setup reaches destination endpoint domain (4, Fig. 2.5). RSVP is the sole constituent of MPLS inter-domain control-plane and realization of LSP service-request protocol in this and many other decentralized frameworks.

D. Limitations

As a framework capable of providing end-to-end QoS solutions, MPLS has been really instrumental. However, to be suited for our other application-centric plus dynamic requirements, it has a few limitations owing to the control plane being rigid, inflexible and insecure as argued below.

- *Local Labels:* Each router along the end-to-end LSP tunnel assigns a local Label which mandates traffic tunnelled will go through Label swapping at every hop inducing latency. A global Label herein for the entire path in the domain would be empowering.

- *Security and Administration:* RSVP-TE essential as an inter-domain LSP service-request protocol has not been widely deployed because of its security and management limitations, owed to network being exposed to potentially rogue exterior service-requests, and administratively taxing process of negotiating peering agreements which are mandatory for any inter-domain service (Dugeon and Meuric, 2017, p. 3), (Johnston et al., 2011, p. 8).

- *Lack of flexibility:* Also RSVP orchestrated and deployed by meagre domain border routers is not equipped enough to support advanced error-handling and flexible operations essential to instil agility and robustness into an inter-domain framework.

- *Distributed and Rigid:* Provisioning MPLS tunnels for an enterprise takes a lot of processing time mainly due to a distributed and hence rigid control plane . Upfront, SLA negotiations are offline and manual, loose interaction between distributed control entities- PCC, PCE and other network control entities, hop-by-hop internal signalling of RSVP and incompetency of inter-domain RSVP discussed above, all add-up to the rigidity cause.

Considering the importance of MPLS-TE in wide-area network operations, there have been works proposing to embed dynamicity and efficiency into its functioning as briefed below, and thus solving some of the above limitations in the process.

2.3.2 Other Distributed QoS Frameworks

- Traffic Engineering Automated Manager (TEAM) (Scoglio et al., 2004), MPLS Adaptive Traffic Engineering (MATE) (Elwalid et al., 2001), Routing and Traffic Engineering Server (RATES) (Aukia et al., 2000) are some of the early proposals for algorithms and implementation of a central TE server in MPLS networks. The need for this central TE server was realized early since an updated global network view is necessary for optimized LSP computations, and overtime these efforts have standardized into the definition of PCE and TED. With recent movement towards a centralized network control-plane by way of a SDN controller, PCE itself has been investigated to be evolved into a network controller since it already has some of the tenets of centralized control-plane, PCE as a Central Controller (PCECC) thus being proposed for effective transition of legacy MPLS networks into SDN-MPLS networks (Zhao et al., 2017). PCECC proposes allocation of global Labels and direct installation of Label state into data-plane utilizing southbound interface relieving the need of hop-by-hop RSVP signalling, which mitigates some of the highlighted limitations of MPLS-TE.

- Apart from PCECC, there have been other related works to migrate and implement MPLS control-plane in SDN controller. PAC.C proposes the concept of map-abstraction which provides for the availability of sum network state (topology, link utilization and availability, switch flow and queue info) in SDN controller over which MPLS functions can be built efficiently (Das, 2012) . It echoes the concern with having distributed protocols and entities in legacy MPLS dependent on state dissemination, instead arguing that abstraction of entire network control-plane in a centralized controller can better optimize the provisioning process of TE-LSPs, dynamically re-computing and re-provisioning them, as well as failover. Hasan et al. (2016) also proposes a SDN-MPLS control plane with the objective of identifying congestion in the network and dynamically adjusting the route of LSPs or its bandwidth. Tu et al. (2014) suggested the method of splicing MPLS and Openflow tunnels using SDN paradigm.

  The above mentioned works only innovate at the intra-domain MPLS control-plane or local service orchestration layer, and hence are only able to support intra-domain QoS process. Following is a discussion of distributed frameworks that deal with the application of SDN to the multi-domain scenario.

- DISCO (DIstributed SDN COntrol plane) proposes a SDN controller architecture for local network service provisioning, also including a message-queue based communica-

tion or service-request interface between other such domain controller(s) for supporting end-to-end use-cases such as network-wide state dissemination, QoS provisioning and VM migration (Phemius et al., 2014). Lin et al. (2015) also introduces an interface between network operating systems, referred to as West-East Bridge, for sharing network view that supports innovations like source routing (specific BGP updates for different prefixes) and end-to-end QoS. Similarly, SDNI (SDN Inter-networking) proposes an inter-controller protocol for negotiating end-to-end QoS (Petropoulos et al., 2016).

These works innovate at the inter-domain control plane layer to provide grounds for end-to-end service provisioning, though they don't utilize MPLS tunnelling. Following are few works that also incorporate MPLS tunnelling for inter-domain QoS.

- OSCAR virtual circuit service (Monga et al., 2011) was developed to provision bandwidth guaranteed LSP tunnels end-to-end between different Research and Education networks domains (ESNET, GEANT, etc). It was motivated by the need to support networking needs of many scientific research projects that require access to large amounts of data distributed over these domains. For instance, data from ATLAS system in CERN to ESNET data-centres in USA and then to end-users in universities. Its primary innovation is a per-domain central TE server that consists of PCE,TED and other Policy Enforcement entities, which intakes end-user bandwidth request and thereafter interacts with legacy MPLS control plane (majorly RSVP) to provision VCs locally in the domain. For multi-domain service provisioning, it doesn't incorporate RSVP for its security limitations highlighted previously, and instead a protocol referred to as Inter-Domain Controller Protocol (IDCP) was developed in parallel defining messages and procedure for aiding collaboration between different domain OSCAR entities for the purpose of end-to-end LSP setup (DICE, 2010).

- Standardization efforts around IDCP led to the development of Network Services Framework (NSF) by the Open Grid Forum with the goal of enabling automated provisioning of federated network services (OGF, 2014a). The inter-domain LSP service-request protocol in NSF, analogous to IDCP, is referred to as Network Services Interface Connectivity Service (NSI-CS) which provides means to propagate VC setup request over multiple domains (OGF, 2014b). Such inter-domain service-request protocols are complemented by a supporting protocol that provides access to inter-domain topology, for instance, Document Distribution Service (DDS) in NSF (OGF, 2014b, p.4). Fig. 2.6 illustrates these different components of NSF, and another component - network-agent which is analogous to SDN controller. Podleski et al.

(2014) incorporates NSF services to create connections and reserve resources over multiple domains to further create a federated testbed where network resources are spread over these many domains.

OSCAR though suffers from large VC setup time and last-mile provisioning in user's domain. The latter reflects the need for an end-to-end QoS provisioning service delivered natively from the network control-plane (in-network), that will represent a more widespread general-purpose service, and can be deployed by even smaller networks such as campus network domains. Such a native service should also be better able to leverage current inter-domain services such as BGP and thus completely mitigate or reduce the dependency on specific interface protocols like IDCP or NSF.

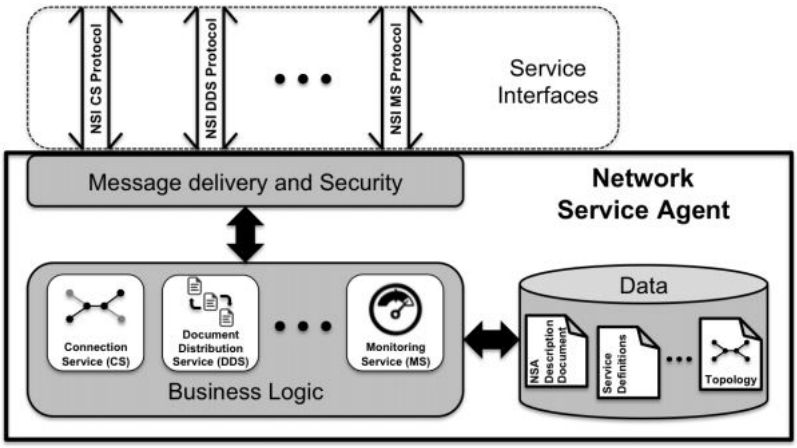


Fig. 2.6 Different components of NSF - Network Agent, CS and DDS. Image from OGF (2014a)

- Within IETF, Backward Recursive Path Computation (BRPC) protocol has been standardized to compute globally optimum multi-domain LSPs (Vasseur et al., 2009). It relies on the collaboration of different domain PCE entities via PCEP protocol to compute a Virtual Shortest Path Tree (VSPT) from source endpoint to destination endpoint, wherein each edge of the tree represents an abstract inter-domain path along with its resources. An end-to-end globally optimum path for LSPs thus can be computed from VSPT at the source-domain PCE, with Dugeon and Meuric (2017) proposing the signalling of this shortest optimum path also through PCEP itself in order to completely mitigate dependency on inter-domain RSVP due to its security and management limitations.

Essential features and related works	Software control-plane (SDN)	Dynamic	Domain privacy aware	Label Switching	Inter-domain QoS Support
Centralized					
RTM-SDN	✓	✓	×	-	✓
Google B4	✓	✓	×	-	✓
NaaS	✓	✓	×	-	✓
Distributed					
MPLS-TE	×	×	✓	✓	RSVP-TE
PCECC	✓	✓	✓	✓	×
Pac.C	✓	✓	✓	✓	×
DISCO	✓	✓	✓	×	Message-queue
SDNI	✓	✓	✓	×	✓
OSCAR	×	×	✓	✓	IDCP/ NSI-CS
BRPC	-	-	-	-	PCEP
MaaS	✓	✓	✓	✓	MaaS-Fed

Table 2.1 Comparison of different related QoS frameworks and entities

2.4 Summary

In centralized QoS frameworks, each domain is required to share sensitive network information and services with a third-party parent entity posing a risk to the privacy and integrity of those domains, and thus are suited for only a sub-set of environments where all network domains are under the administration of single authority. Contrary to this, distributed frameworks represent a more general-purpose solution as they can be deployed by two or more heterogeneous domains without the dependency and bindings of a third-party parent entity. Also, distributed frameworks can host and deliver inter-domain QoS services from operator level i.e. natively and closely tied to the domain network control-plane, referred to as *in-network* property in this document, which is intuitively helpful to induce agility and dynamicity into the end-to-end QoS provisioning process, as will be better brought to light later in subsequent chapter III.

MPLS-TE is a prominent framework having the above described advantages of a distributed service provisioning, however it is not supportive of our real-time QoS requirement,

the relevant limitations to which were described earlier. There have been related works in progressing its agility such as pac.c and others as discussed previously in section 2.3.2 (2), but they only innovate locally at domain control-plane level. Though there also are works, not necessarily MPLS incorporating, that propose dynamic inter-domain services for end-to-end QoS provisioning but they do not support with the design and proposal of an underlying control-plane as well (Sec. 2.3.2-3).

To summarize, there is a perceived lack of end-to-end MPLS works that primarily utilize a major breakthrough in networking of SDN (Federation) to mitigate rigidity in not just local but also in inter-domain LSP/QoS setup process; improve upon security and administration of MPLS or broadly any inter-domain framework; and also closely consider the development of underlying supporting IP-BGP planes in SDN context considering the fact that SDN is still a relatively new paradigm. *Through MaaS in this Thesis, we propose such an inter-domain MPLS framework utilizing SDN Federation, and on broader terms, a complete SDN-MPLS-IP-BGP control-plane*

Chapter 3

MPLS-as-a-Service

3.1 Introduction

MPLS has been a prominent TE framework of Internet. To our preference, it is also naturally in-network in a sense that each domain natively deploys functions to support inter-domain (LSP) service provisioning, and they directly provide Label-switching services to each other without the need of any third-party parent orchestrator. However, its control-plane is rigid and hence lacks dynamicity, as well as has security and management limitations as briefed below.

- Domain MPLS control-plane (PCE, TED, IGP-TE, and RSVP) is distributed and hence rigid that affects the ability to create TE-LSP tunnels dynamically. To satisfy our requirement of a QoS framework suitable for wide-range of application or enterprise customers, small or medium, it is of utmost importance for network resources to be allocated and de-allocated swiftly, and thus agility be induced into LSP setup and management process.

- Inter-domain MPLS control-plane primarily comprising of a signalling protocol - inter-domain RSVP-TE has not found widespread deployment because its mandates prior contracts to be negotiated, and aggravated security concerns as deploying it makes a domain vulnerable to any potential rogue request from outside. These concerns have to be mitigated in order to make it easier for MPLS operators to provide inter-domain services to other operators or applications (Dugeon and Meuric, 2017, p. 3), (Johnston et al., 2011, p. 8).

Inspired by MPLS for its TE capabilities and mitigating its above described limitations, this thesis proposes a solution framework referred to as MaaS in evolving MPLS to dynamic

and application-centric QoS framework of Internet true to our requirement. As the name suggests, MaaS is designed to provision on-demand QoS LSP even over multiple domains at an application's dynamic request. In a nutshell, it includes a SDN-MPLS control plane that takes in real-time application QoS requirements and utilizes network control abstraction layer in the controller to dynamically and optimally provision TE-LSPs locally. Further, it is packed with an inter-controller protocol (MaaS-Fed), as a replacement for RSVP-TE, in propagating the LSP service-request to neighbouring domains in order to effect end-to-end provisioning. This high-level overview of MaaS is illustrated in Fig. 3.1. Furthermore, newer features not supported in legacy MPLS such as service re-negotiation and customer-delegated control of tunnels have been introduced that improve upon the scalability and resiliency of framework.

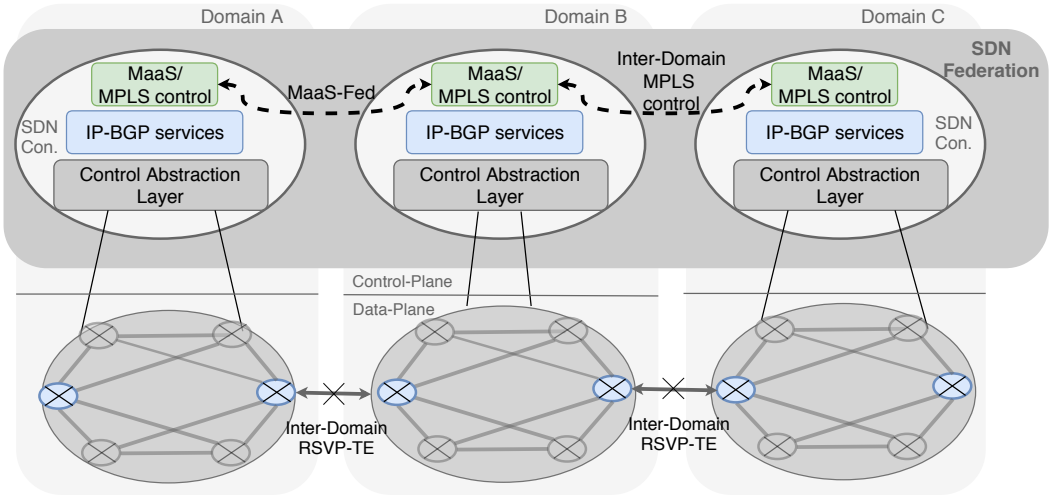


Fig. 3.1 High Level architecture of MaaS showing SDN and SDN Federation innovations.

3.2 Enablers

Primarily, SDN and SDN Federation breakthroughs are utilized by MaaS for mitigating limitations of MPLS and evolving it into a dynamic, general-purpose and application-centric QoS framework as explained briefly below.

3.2.1 SDN Controller

Contrary to legacy MPLS control-plane being distributed, the proposed MaaS framework works by implementing a new centralized MPLS control-plane in SDN controller. The

environment of controller providers a one-stop *network abstraction layer* for MPLS control to dynamically interface with the domain-wide data-plane and also provides for an agile interaction environment with other network control and management entities especially SDN-IP-BGP (Fig. 3.1). The dynamicity of SDN controller is the primary enabler in our endeavour to induce agility into TE-LSP setup and management process of MaaS, resultantly empowering network service providers to offer MPLS as a service model rather than present day fixed contract.

A domain SDN controller can help us innovate only in the process of local QoS/LSP setup, but to consider for end-to-end LSP provisioning over multiple-domains, the concept of SDN Federation is leveraged.

3.2.2 SDN Federation

Inter-domain MPLS control-plane concerns itself with providing collaboration support for setting up QoS LSP tunnels over multiple domains. Its primary protocol RSVP-TE has been identified of not finding widespread deployment primarily as configuring any inter-domain service-provisioning protocol between two border routers aggravates security concern.

SDN Controller is a movement towards programmable and centralized *domain control-plane*. Extending this concept, collaboration between different network SDN controllers can bring about a programmable and dynamic *multi-domain/e2e control plane* as well (Fig. 3.1). Referred to as SDN Federation, it can provide with rather secure, manageable and flexible grounds for deploying inter-domain service protocols in comparison to when such protocols are deployed between domain border routers as in legacy approaches. We exploit this enabler to design and implement a new service-request protocol called MaaS-Fed to support in MPLS control process of end-to-end LSP setup, as a replacement of legacy protocol inter-domain RSVP-TE (Fig. 3.1).

The limitations of legacy MPLS and the innovations proposed by SDN (Federation) are better surfaced in the following sections wherein the entire MaaS framework is explained.

3.3 MaaS Implementation

The proposed MaaS framework is developed and delivered as an ODL (odl:web, 2016) SDN controller plugin per-domain named similarly. In the following sections, complete functioning and features of MaaS are briefed, broken down into its 3 sub-components - intra-domain LSP setup, inter-domain (LSP/QoS) service-request interface, and also a similar

protocol (MaaS-Fed). These sub-components or functions are a feature of legacy MPLS as well, and its proposed evolution MaaS proposes innovations in each of those.

As also stressed previously, one of the important property of our MPLS control-plane or MaaS is its close utilization and interaction with rest of the IP-BGP control plane in controller. Since SDN is still in emerging phases, there is no standard or de-facto concrete implementation of a SDN controller architecture. Therefore, in this Thesis as part of the Testbed setup, we also develop a SDN-IP-BGP control plane leveraging off-the-shelf *Openflow network abstraction layer* provided by ODL SDN controller. In describing the functioning of MaaS plugin, these Testbed plugin and services are also referenced which are self-explanatory to an extent and later detailed in Testbed chapter IV.

3.4 MaaS: Intra-domain LSP protocol

This sub-component of MaaS is in-charge of provisioning domain local segments of customer's end-to-end LSP tunnels, and constitutes the SDN-MPLS intra-domain control plane. In a nutshell, it does so by providing APIs for applications and enterprises to convey their network QoS needs (1, Fig. 3.2), thereafter computes network route paths that combine to meet the desired resource requirements (2, Fig. 3.2), assigns a global Label to each path segment and signals them LSP segments onto the forwarding plane (3, Fig. 3.2). On top of this, MaaS shares information on provisioned LSPs with their associated customer (4, Fig. 3.2), which empowers newer features specifically customer-delegated mapping and control of LSPs.

The sub-functions to this sub-component as summarized above are detailed next, and are rather not completely performed by the MaaS plugin, but through orchestrating the APIs and services of other standard network plugins, filled green in Fig. 3.2, also highlighting its previously stated *in-network* property.

3.4.1 Application Interface

MaaS application-interface, a constituent of logical SDN Northbound layer, provides APIs to application or enterprise customers, for requesting QoS in real-time. Two REST APIs are offered for now - *create-lsp* and *delete-lsp*, to convey creation and deletion of network service dynamically (1, Fig. 3.2). The first two *create-lsp* request parameter objects (Fig. 3.2), src-address and dest-address, signify the tunnel endpoints while *resources* object quantifies

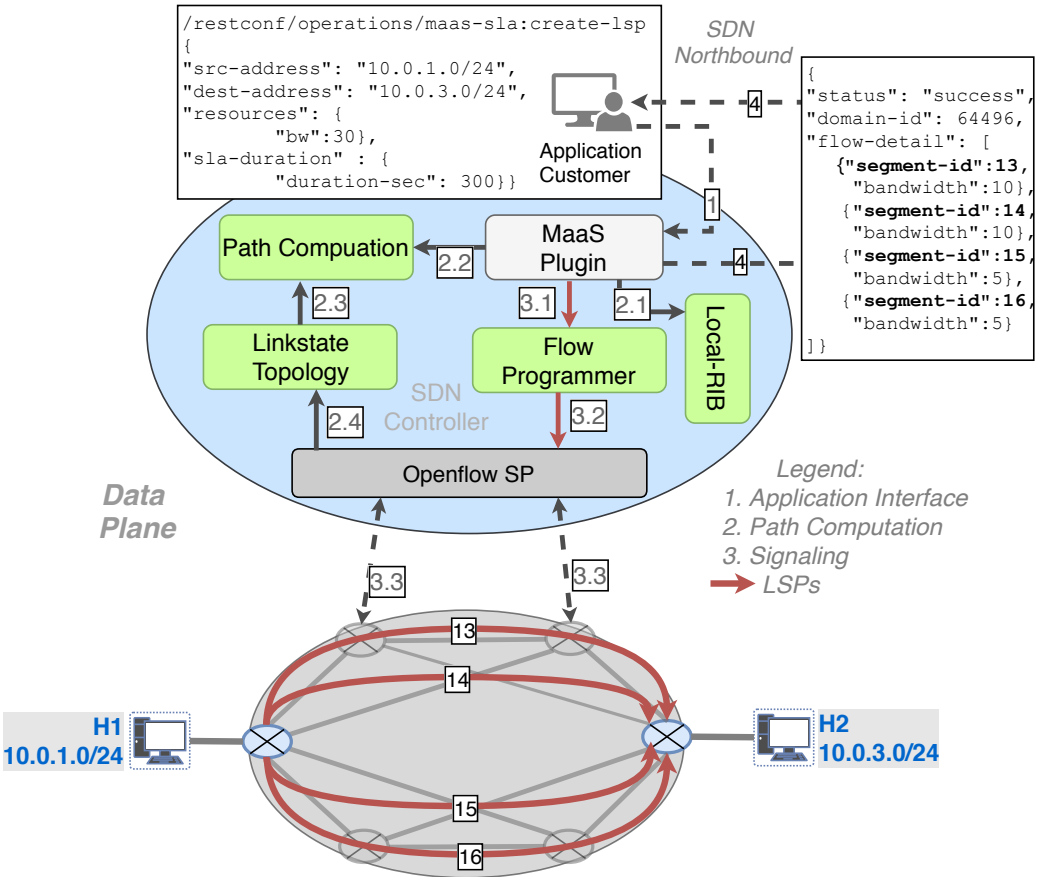


Fig. 3.2 Intra-domain LSP provisioning function of MaaS.

the network QoS requirement to which bandwidth in Mbps is only supported for now. *sla-duration* object defines the duration for which LSP service is desired.

SDN Northbound layer provides an online mechanism for applications to interface with the network for purposes like negotiating SLAs, accessing performance data, etc. Compare this to legacy networking wherein such processes were manual, offline and time-consuming. This innovation is made possible by the coming of SDN controller which opens up the network control to users and applications.

In the making of a dynamic QoS framework, such an interface is a basic requirement as it provides for a verbose real-time communication between applications and networks making them both aware of each other's state, requirements and outage.

3.4.2 Path Computation and Label Allocation

A customer's QoS requirement conveyed through above described *create-lsp* API is met by MaaS through provisioning TE-LSPs in the network. A significant part of this provisioning process is to compute explicit end-to-end paths in control-plane that would eventually become LSPs in the data-plane. For this PC process to happen, access to domain-wide data-plane topology and resource information is a mandatory pre-requisite, which in legacy protocols is made available by TED, IGP, BGP-LS, and while in SDN environments is facilitated through a network CAL. The PC process using this CAL is explained below.

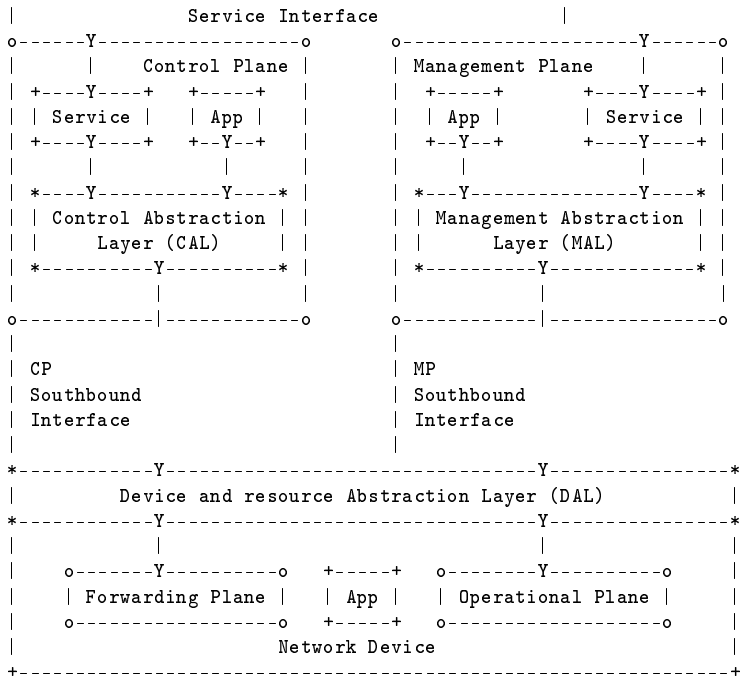


Fig. 3.3 General SDN controller architecture showing different layers and CAL. *Image from Haleplidis et al. (2015)*

- *Network CAL and Openflow SP:* Unarguably, the most basic and important part of any SDN controller platform is network control abstraction layer (CAL, Fig. 3.3). This layer provides an updated network view and remote interface to underlying switching fabric for any interested controller service to utilize. This layer is enabled courtesy of a protocol between controller and data-plane, referred to as southbound interface (Fig. 3.3). Openflow is one widely used Southbound Protocol which we also use.

  A common constituent of network CAL is a Southbound Plugin (SP) which abstracts the complexity of southbound protocols for network application developers (Openflow

SP, Fig. 3.2). Utilizing this CAL or SPs, further complexity of controller environment is generally reduced by other helper plugins such as the ones providing topology and routing services. In Opendaylight and our SDN controller architecture of Fig. 3.2, *Openflow SP, Linkstate-Topology and Local-RIB plugins* form part of the network control abstraction layer leveraging which other SDN applications like *path-computation* can access potentially real-time network topology and state information dynamically for computing LSPs as we see next.

- *Finding network nodes:* Customer provides desired QoS path endpoints in the form of IP prefixes. Prior to triggering the path computation algorithm, the network nodes that host these IP prefixes need to be determined. MaaS plugin therefore queries domain Routing Information Base (*local-RIB*, Fig. 3.2) to find out the network nodes hosting the source (10.0.1.0/24) and destination host addresses (10.0.3.0/24) (*2.1*, Fig. 3.2). *Local-RIB* keeps a track of all hosts local or inter-domain, their IP addresses, and the node they are attached/reachable by listening to any changes in *Linkstate-Topology* model for local hosts and interfacing with BGP process for inter-domain hosts as will be better brought to light later in section 4.3.4.B.

- *Path Computation:* Thereafter knowing endpoint network nodes for QoS/LSP path, MaaS plugin makes use of *path-computation* service (which is analogous to PCE) to request an explicit hop-by-hop local path between those nodes (*2.2*, Fig. 3.2). *Path-Computation* service in turn retrieves network topology and resource information from *linkstate-topology* service (which is analogous to TED) and runs a Constrained Shortest Path First (CSPF) algorithm on it in to compute a suited path that has required resource availability (*2.3*, Fig. 3.2).

  Linkstate-topology plugin thus maintains the desired TE information to compute the path. Apart from topology, the required TE information for this process as well is the capacity and availability at different queues in device interfaces. Each plugin in ODL controller stores data in a central in-memory datastore. Fig. 4.12 shows a JSON representation of the required topology and TE data maintained by Linkstate Topology plugin and queried from the datastore through standard read APIs.

- *Segments and Labels: Path-computation* service computes multiple local paths, referred to as segments, if the entire resource requirements are not satisfied with one. Each of these tunnel segments is allocated a unique Label, and they might not necessarily be providing equal network QoS as overlaid on the Fig. 3.2 data-plane.

3.4.3 Signalling

Once the Label-switched Paths are computed, lastly they have to be installed or actuated among the involved network devices, or to say, LSP state has to be transitioned from control-plane to data-plane in a process known as signalling. In legacy MPLS, signalling is performed through RSVP-TE protocol which hop-by-hop from source node to destination node installs Label forwarding state and performs resource-reservation for the desired tunnel. Thereafter, each of the node then assigns a local Label for the path and shares it with the upstream/downstream node completing the provisioning process.

Signalling process however in SDN-MPLS control-plane can be made considerably convenient. Since SDN controller is characterized by having a southbound protocol interface (Openflow for instance) to control each network device, therefore no hop-by-hop signalling protocol like RSVP is required to put MPLS tunnel state among the devices as it can be made directly through the southbound protocol. Moreover, a domain-global *Label* can be allocated for the entire tunnel segment by controller acting upon its network-wide view and control, instead of a per-device *Label* as in legacy approaches.

MaaS framework devises and incorporates above SDN signalling innovations. Services of *flow-programmer* and *Openflow SP* are utilized by MaaS plugin in signalling a LSP on data-plane. While *flow-programmer* plugin converts an explicit path provided by *path-computation* service into relevant Openflow flow-modification rule objects (match-action pairs), Openflow SP installs these rules or changes into LFIB of every involved node to complete the provisioning process (*3*, Fig. 3.2). Specific to openvswitch devices used for experimentation, this data-plane LFIB modification consists of an Openflow rule in every ovs node along the end-to-end path that matches every incoming packet containing the assigned Label for LSP segment to an output interface-queue pair. ovs supports traffic metering and policing at interface/queue ingress to ensure that LSP flow traffic is within contract and they do not oversubscribe the queue and congest the device.

While the process till here provisions LSPs for a customer's QoS needs in real-time, some other innovations are proposed by MaaS empowering additional features as explained below.

3.4.4 MaaS Response

In legacy MPLS control-plane, there is no online mechanism to share essential information on LSP provisioning back to requesting application customer. Contrary to this in MaaS

framework, once the LSP segments are provisioned, we send their Label and Bandwidth information back to the customer as a response to the *create-lsp* API request (4, Fig. 3.2). The motivation behind this is to empower customers to themselves map or tunnel IP flows into their provisioned LSPs, resulting in a more responsive and scalable approach as explained next.

3.4.5 MPLS Forwarding Plane Innovations

For a MPLS virtual-circuit to be functional, two types of forwarding-plane modifications have to be carried out in the data-plane- LFIB to create tunnels and LFEC to send IP flows through them tunnels, as briefed below.

- *LFIB:* The Label forwarding state for the LSP tunnel has to be installed in each of the involved network devices. This consists of a forwarding-table entry that maps any incoming *Labeled* packet to an output interface corresponding to the LSP's next-hop. Fig. 3.4 shows LFIB table of all involved Label Switching Routers (LSR) for an arbitrary tunnel that has a *Label numbered 13*. In MaaS framework, this installation is done by the signalling process as explained previously.

- *LFEC:* End-hosts directly do not produce a *Labeled* packet. Forwarding-table modifications thus also have to be made that push a Label header onto the incoming IP packets so as to tunnel them into provisioned LSP. This forwarding-table entry is required only once, generally at the tunnel ingress (*BN1*, Fig. 3.4). A related table entry also has to be made at the tunnel penultimate-hop or egress that pops Label header to reveal original IP packet at the end of MPLS domain (*BN2*, Fig. 3.4). Multiple IP flows can be mapped to one common Label-switched Path, and hence the term *Label Forwarding Equivalence Class* used for referring to them group of flows and their associated Label.

- *LFEC Mapping Innovation – Customer Premises Equipment (CPE):* The above described LFECs for a newly provisioned LSP tunnel are generally created by MPLS service providers themselves, for example in Fig. 3.4 scenario by ISP domain through its SDN controller at node BN1 (M1). However, to reduce the forwarding-state in network access or core and make the framework scalable, these LFEC mapping entries can be instead alternatively pushed to customer domain, for instance, at customer edge router (*M2.1, CE1,* Fig. 3.4). Though this is only feasible if Customer Edge (CE) router is managed by the MPLS provider themselves. The evolution of NFV makes this easier as the need for an expensive managed customer premises equipment can

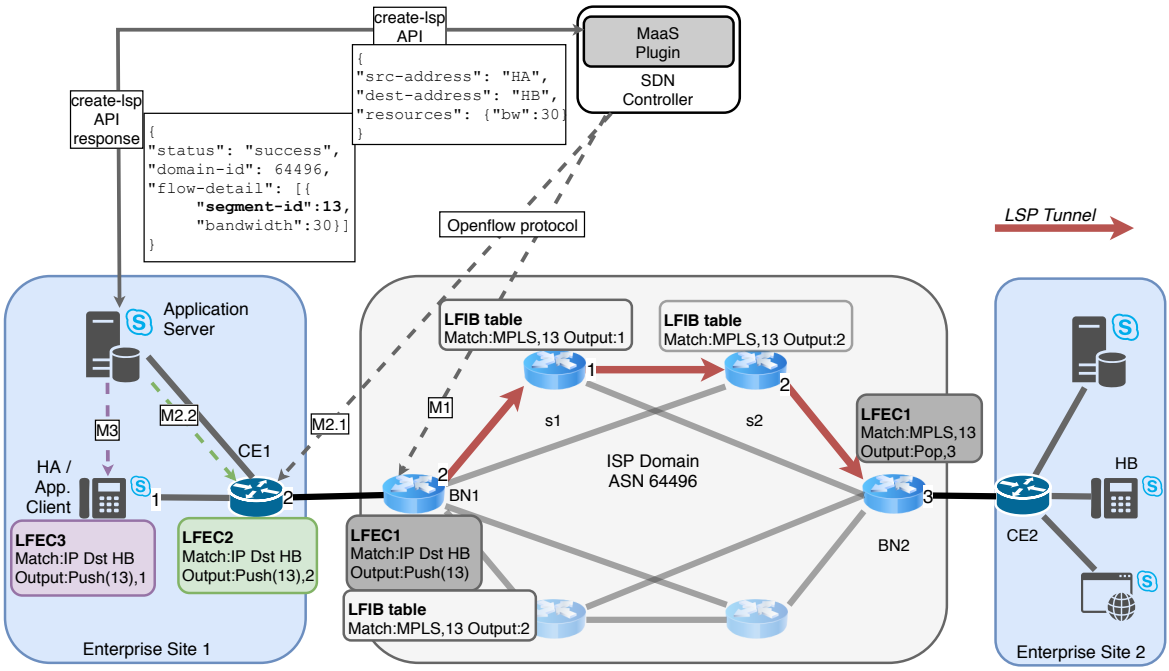


Fig. 3.4 Schemtic showing LFIB tables and different proposed locations for LFEC rule installation.

be mitigated though a software based virtual CPE, a breakthrough also utilized by SD-WAN products (MEF, 2017).

- *LFEC Mapping Innovation – Customer delegated Mapping:* Alternatively, the process of creating LFEC mapping entries in CE routers can be done by the customer themselves (*M2.2 LFEC2,* Fig. 3.4), thus mitigating the need for MPLS operators to manage customer premises equipment. This is made possible since as a response to the *create-lsp* API request, MaaS returns the provisioned LSP segments Label and Bandwidth information back to the application customer as explained previously and shown in Fig. 3.4 and also Fig. 3.2(4).

  Another efficient approach is for the application-servers to directly manipulate clients or flow-agents in producing flows already containing desired Label header (*M3 LFEC3,* Fig. 3.4) instead of Labels being pushed later by the network. This though only makes sense if entire end-to-end network is MPLS enabled but advantages are in mitigating encapsulation latency incurred later during flow flight. Again, this innovation is empowered by SDN control enabling a dynamic interface between networks and applications, and a similar approach has been utilized by NEAT wherein SDN controller

instructs end-hosts on an optimum transport protocol considering real-time network view (Santos et al., 2017).

The framework courtesy of *create-lsp* API response supports any of the above described approaches for creating LFEC mapping entries. However, for the latter two customer-delegated approaches (*M2.2 and M3,* Fig. 3.4), a supporting feature has to be also built into the customer application for them to parse the MaaS response and perform mapping at customer edge routers (M2.2) or endpoints (M3). For our demonstration later in chapter V, we follow M2.2 approach by building support for LFEC mapping in a simulated application.

Customer delegated approach for LFEC mapping incorporated in MaaS has following advantages over legacy network-based approaches-

- *Scalability*: Some of the MPLS data-plane state (LFEC mapping) is pushed back into customer premises equipments. This enhances scalability of our QoS provisioning framework as otherwise provisioning LSPs for a large number of applications and customers would have required making many more LFEC mapping entries at the edge of MPLS network domain posing a problem of state explosion.

- *Reduced Network Workload*: Though the proposed approach, domain network is relieved from LFEC mapping/tunnelling operations and can concentrate on other core control operations.

- *Application-based TE:* Customers know best on how to utilize or allocate their provisioned LSPs among their many competing application-flows. By delegating the LFEC mapping and hence control of LSP tunnels to them, they can better allocate proportion of purchased LSP bandwidth to competing flows through simply assigning *Labels* on the basis of flow priority. This results in a more optimal and straight-forward approach than if mediated by network.

3.4.6 Contribution Consideration

Much of the perceived agility and dynamicity advantages of the SDN-MPLS control-plane presented above over its legacy counterpart are also inherent advantages of centralized network control or SDN. This Thesis work thus does not claim a significant contribution as of yet, as also there have been similar works previously. Although none of the related works feature similarly complete and open-source implementation down to the lowest layers as MaaS and featuring mapping innovations. Rather, the contributions of the proposed SDN-MPLS control-plane or MaaS surface when a wider inter-domain scenario is considered

as next. The components and functions to inter-domain SDN-MPLS control-plane are thus explained next.

Fig. 3.2 only considers an internal QoS scenario where both the application source and destination endpoints are hosted in the same domain in order to explain the intra-domain LSP setup process. Considering wider inter-domain QoS problem where endpoints reside in different domains, on a higher level, it can be solved if each involved domain between source and destination endpoint provisions LSP service locally through their own domain-controller MaaS entities, thus provisioning end-to-end QoS. However, there are other specific challenges involved in this- a distributed LSP service-request protocol has to be devised that supports communication, response and handling between domains; an inter-domain interface for that protocol to run on; and an inter-domain topology resolution mechanism for figuring out an explicit domain-level path or a next-hop domain to propagate LSP provisioning request forward. Following components cater to these requirements to help achieve multi-domain end-to-end service provisioning.

3.5 MaaS: Inter-Domain Interface and Federation Engine

A. Overview

Generally speaking, an inter-domain interface is required to enable communication between domains for purposes that could be propagating QoS or resource-reservation request or any other arbitrary task. Such interfaces along with the protocols and services that run over them constitute the inter-domain control plane such as commonly known eBGP. For our MaaS framework, one such interface is needed to communicate LSP service-request hop-by-hop from source-domain to intermediate domains and onto destination-domain in order to realize end-to-end provisioning. Legacy MPLS control planes employ inter-domain RSVP-TE for such an interface requirement, but as has been previously discussed, it suffers from security and management problems.

For such an interface requirement of the SDN-MPLS control-plane or MaaS framework, Federation-Engine is proposed and also implemented by way of a plugin in Opendaylight SDN controller (Fig. 3.5). It provides native APIs to be utilized by any arbitrary controller service such as MaaS plugin (1, Fig. 3.5) for communication with its counterpart in any BGP peered domain. A distinguishing and important feature of Fed-Engine is that instead of opening a new TCP/IP session between two neighbouring domain entities for this purpose, it utilizes existing widely deployed eBGP inter-domain interface to communicate service data.

On receiving plugin message and the AS number of recipient domain, Fed-Engine creates a new BGP route update payloaded with serialized plugin message (2, Fig.3 3.5), and sends it along the controller BGP pipeline destined towards the desired AS domain (3, Fig. 3.5). Upon reaching there, the payload message is parsed, de-serialized and handed over to the counterpart service (4, Fig. 3.5). As can be seen, Fed-Engine relies heavily on the BGP control-plane (*brown*, Fig. 3.5), which is explained later in Testbed chapter IV.

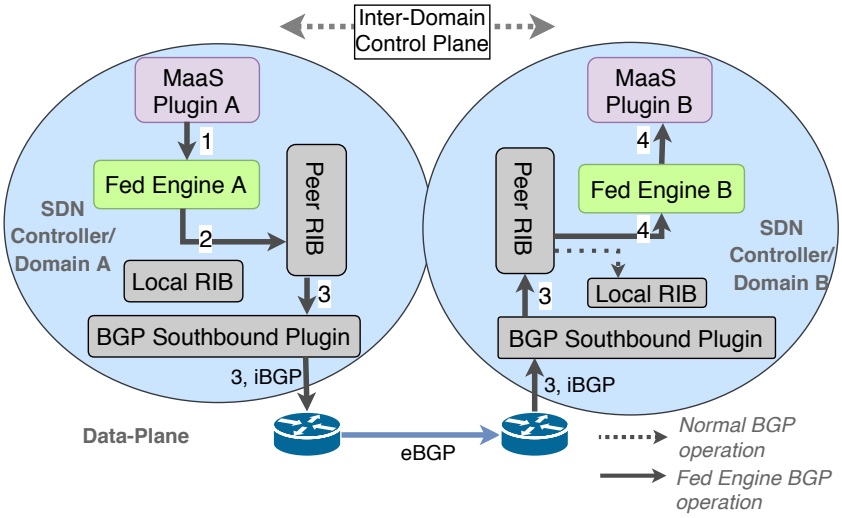


Fig. 3.5 Illustration scenario depicting generalized BGP inter-domain interface empowered by Federation Engine.

B. Generic Inter-Controller Interface

In the context of SDN Federation, it is intuitive to re-imagine inter-domain interface as being deployed between neighbouring SDN controllers rather than between border routers as done in legacy networking. Moreover, with a single holistic control entity in SDN controller, it is also possible for many different service protocols to utilize a single communication-channel or interface between neighbouring domain-controllers rather than an interface specific to each as is the current practice, the term *generic inter-domain control-plane* thus being coined for such an interface.

BGP is an already widely deployed interface between autonomous domains, though it is only deployed between border routers, not SDN controllers. However, device-plane BGP process is conveniently inter-operable with SDN control-plane, and in this way BGP can be extended from operating just between border routers to between SDN controllers (Fig. 3.5). Moreover, existing BGP standard already has an extensible mechanism by way of *optional transitive attributes* which can be leveraged to encode any other plugin message.

Such extension of BGP is not new and has been proposed in the past for services like - exchanging SLA parameters (Shah et al., 2017), path provisioning requests (Chen et al., 2017), and propagating congestion state (Prior and Sargento, 2006), though with not much success. However, SDN controller enabled programmability, softwarization and centralized agile control-plane provide a rekindled opportunity to add flexibility into BGP.

BGP is thus ideal to be extended from a routing protocol to a *generic inter-domain interface* and cater to all of domain's external tasks than just *routing*, which also fits true to its definition of Border Gateway protocol. To realize this innovation, we use SDN-BGP control-plane available off-the-shelf from Opendaylight community (*odl-bgpcep-bgp*) as the foundation (odl-bgp, 2018). Thereafter, we implement a wrapper over it called *Federation Plugin* that provides APIs for controller plugins like MaaS on one end, and interfaces with *odl-bgpcep-bgp* plugin on the other, thus enabling native inter-domain communication capabilities over BGP.

C. Advantages

Federation-Engine in being used as an inter-domain collaboration interface, provides following advantages over its legacy RSVP counterpart.

- *Security:* Both Fed-Engine and inter-domain RSVP-TE handle LSP service-request from other outside domains, and therefore expose local network domain to any external rogue request. However, Federation Engine is potentially much more secure as it is orchestrated and hosted by domain SDN controllers compared to RSVP being hosted at meagre border nodes. Security measures and access policies are much efficiently applicable at the controller then at the border routers.

- *Simplicity and Rapid Innovation:* In general any inter-domain protocol, including RSVP-TE, requires pre-negotiated contract to be in place before being deployed as they function across the autonomous boundary of domains. Further, if there are multitude of such protocols and services, it would mandate agreeing on explicit contract for each, which generally happens through slow offline manual means posing administrative inertia in deployment of novel inter-domain services. Through Federation-Engine, all such protocols can be transported over one generic BGP interface, limiting the complexity of inter-domain control-plane which otherwise would have contained as many interfaces/channels as protocols. Newer inter-domain services like MaaS and others can be conveniently deployed over Fed-Engine and generalized BGP. This favours rapid innovation contrary to legacy inter-domain RSVP-TE which is not agile

enough for extension as it would require upgrade in physical network equipment handling protocol. Fed Engine thus provides a convenient and agile methodology for enriching inter-domain control plane with future emerging services.

- *Administration:* eBGP for long has been the only widely deployed inter-domain control protocol. Keeping this construct allows inter-domain control-plane to progress without requiring overhaul of administrative process in deploying newer inter-domain services.

D. Controller Route

The *extended* BGP update message produced by Fed Engine is different from the *normal* BGP route update message in terms of structure and handling as explained below. Therefore to distinguish between the two, Fed-Engine message is referred to as simply *controller route* named so because as opposed to the scope of normal BGP update message being an IP prefix, the scope of Fed-Engine message is rather a destination domain SDN controller.

- *Different Handling:* Controller route update does not contain any IP routing information as in a standard BGP route update, instead just some third-party plugin or service data. For this reason, it has to be stripped early from the normal BGP pipeline to prevent corrupting the Routing Information Base (RIB), as is done by Fed Engine in 4, Fig. 3.5.

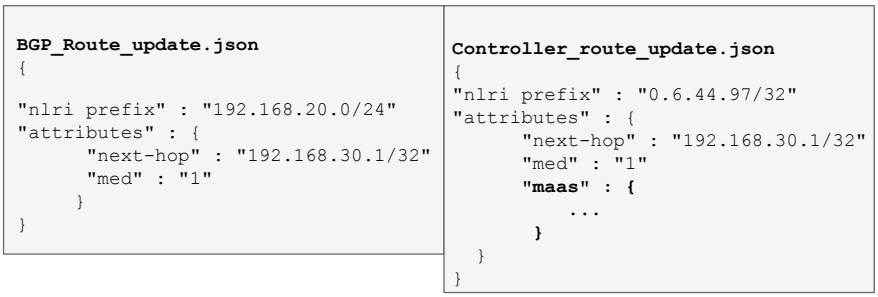


Fig. 3.6 Example JSON format of normal BGP update and controller route update produced by Fed Engine.

- *Different Structure:* Related to above, for controller route update to be handled differently and stripped early from the BGP pipeline, it has to be distinguishable from standard route update. Therefore, controller route update in being originated by Fed Engine is identified as containing AS number of the destination domain in *nlri* field, as shown in Fig. 3.6 (right). Notice it containing the AS number of domain B from Fig. 3.5 formatted as an IP prefix in mandatory *nlri* field. This is only a stop-gap

solution as an address identifier specifically for the AS number, as part of *Address Family Indicator* (AFI) mechanism (IANA, 2018a) of Multiprotocol BGP (Bates et al., 2007) can provide a solution fitting into the design of BGP.

Fig. 3.6 also highlights how *maas* or any other plugin message is payloaded onto a BGP update message as a field of attributes object. A total of 256 fields are possible in the attributes object. While some fields like *next-hop* are mandatory and well-known, there's still a large number of space left for future extensions chich is leveraged by MaaS and Federation Engine. IANA (2018b) maintains an updated table of all the well-known and optional attribute fields of BGP.

E. Summary

Federation Engine is a single point of origin for all the controller inter-domain functions, MaaS or others, as well as security and management policies are easy to apply therein. Specific peering agreement for each of the federating services need not to be negotiated offline, as they can be done online in-band also through the Engine. Simplicity of the inter-domain control-plane is also preserved in keeping the construct of BGP. Under the hood, Federation Engine is implemented by modifying Opendaylight BGP plugin which also happens to be the SDN-BGP control plane of our implementation. After refinements, the code is intended to be contributed upstream into the open-source Opendaylight community to be useful for other similar initiatives related to extending BGP.

3.6 MaaS-Fed: Inter-Domain LSP Protocol

3.6.1 Introduction

An inter-domain LSP service-request protocol is required that defines messages and rules of engagement for different domain MaaS frameworks to collaborate and provision end-to-end MPLS virtual circuits. MaaS-Fed protocol is thus developed in MaaS plugin for this purpose (Fig. 3.7). MaaS-Fed propagates LSP *service-request* from the domain wherein application QoS request is received i.e. source-domain towards destination-domain, and also propagates LSP *service-response* in the reverse upstream direction. Federation Engine (FE) interface briefed earlier is utilized to communicate these service messages inter-domain.

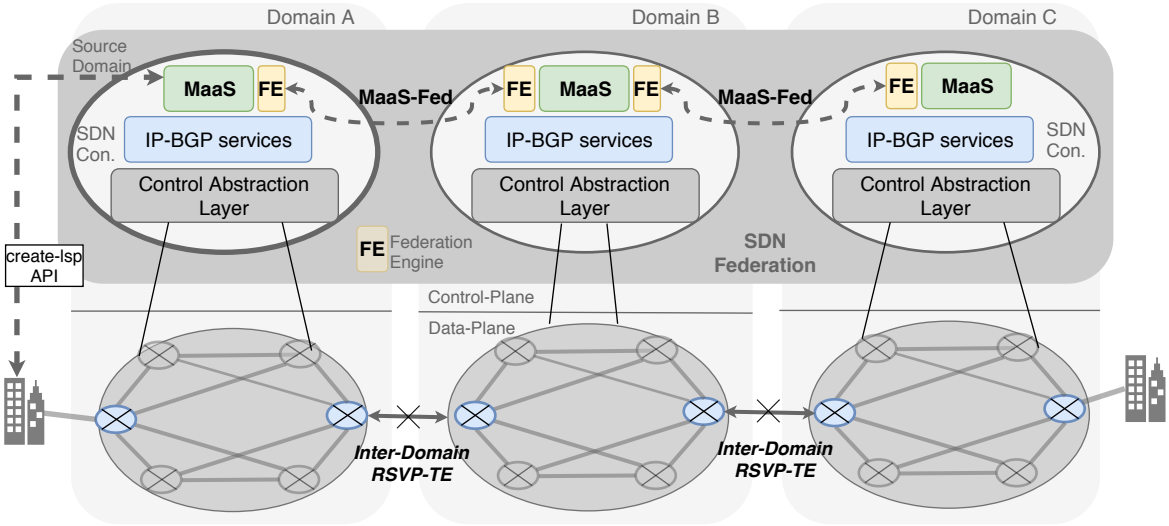


Fig. 3.7 High Level architecture of MaaS showing MaaS-Fed as inter-domain LSP service-request protocol with communication aided by Federation Engine.

A. Motivation

The need to re-invent LSP service-request protocol considering the availability of a de-facto standard in inter-domain RSVP-TE arose due to the limitations of incumbent in supporting advanced error-handling and TE features which are necessary for an agile and application-centric MPLS framework like ours. Although in maintaining similar semantics, RSVP-TE could have been extended and modified to include desired new features, but the movement of inter-domain field of control from border-routers to SDN controllers empowers rapid innovation and development leveraging programmability rather than modifying and mitigating the limitations of existing solutions. MaaS-Fed is powered by the fact that domain SDN controllers as orchestration entities on either side of a service-request protocol are capable of supporting a much wider set of features, compared to RSVP which is orchestrated by meagre network border nodes (Fig. 3.7). MaaS-Fed is still functionally similar to RSVP which works hop-by-hop between border nodes, while the new protocol works hop-by-hop at a level higher of domain controllers.

B. In-Network Approach

Parallel to a service-request protocol is requirement for a federation-wide topology exchange mechanism through which explicit domain sequence to setup required LSPs can be determined, or simply a next-hop domain can be figured onto which QoS or LSP provision request will be forwarded through MaaS-Fed. Consider the following scenario – an enterprise HQ

domain might be multihomed i.e. connected to multiple ISP domains, and only one of those ISP domains might be connected to the enterprise's branch domain to which it wants a QoS WAN link. Source domain i.e. enterprise HQ MaaS needs access to this federation domain-level topology information so as to figure out the appropriate next domain onto which LSP service-request will be propagated. Similar end-to-end QoS frameworks usually employ a dedicated topology exchange service for this purpose such as DDS in Network Services Framework (OGF, 2014b, p.4). On the contrary, MaaS being an in-network framework can benefit from easy access to the BGP RIB from which the information on domain-level path can be easily derived through AS_PATH and NEXT_HOP attributes mitigating the need for a separate protocol. This shows the utility of an in-network solution, and the advantages of having an interactive SDN-IP-BGP-MPLS control-plane.

C. Features

Apart from facilitating end-to-end service provisioning, following two other features are built in MaaS-Fed to resultantly improve *robustness* and *scalability* of our QoS provisioning framework.

- *Adaptive Service Re-negotiation:* MaaS has been equipped with a service-renegotiation mechanism courtesy of advanced error-handling capabilities in MaaS-Fed. Application desired network resources in their entirety will not always be available in the end-to-end network. Therefore, the presence of an agile communication protocol like MaaS-Fed between involved network domains as well as applications in order to negotiate revised service terms and adapt to intermittent resource shortage goes a long way in making the framework *robust* and hence dynamic.

- *Scalability:* Maas-Fed is equipped to carry and share verbose LSP segments provisioning information (Label and Bandwidth) between domains and ultimately to source-domain from where this information is shared with the requesting customer. This way, control of LSP tunnels can be delegated to customer themselves for their efficient and optimum utilization, making the new MPLS framework *scalable* as the forwarding state and responsibility for IP flow to Label mapping is pushed to customer premises. This mapping innovation was explained in section 3.4.5 for an intra-domain scenario, while for the inter-domain scenario it is aided by MaaS-Fed as explained later in this section.

The above three functions and features of MaaS-Fed are described next in relation to the Fig. 3.8 scenario and also experimentally demonstrated later on a testbed in chapter V.

3.6.2 End-to-end QoS

This subsection briefly explains *MaaS-Fed* facilitated end-to-end QoS-LSP provisioning process specifically in relation to the multi-domain topology and scenario shown in Fig. 3.8. In the scenario, an arbitrary application customer requests a 25 Mbps QoS path between its two inter-domain endpoints using create-lsp API of its source-domain controller's MaaS plugin (*Domain A*, Fig. 3.8). This is a wider case of Fig. 3.2 scenario, wherein a similar QoS path was requested but between local endpoints. This example scenario for end-to-end LSP setup also brings out the essential features and message types of MaaS-Fed.

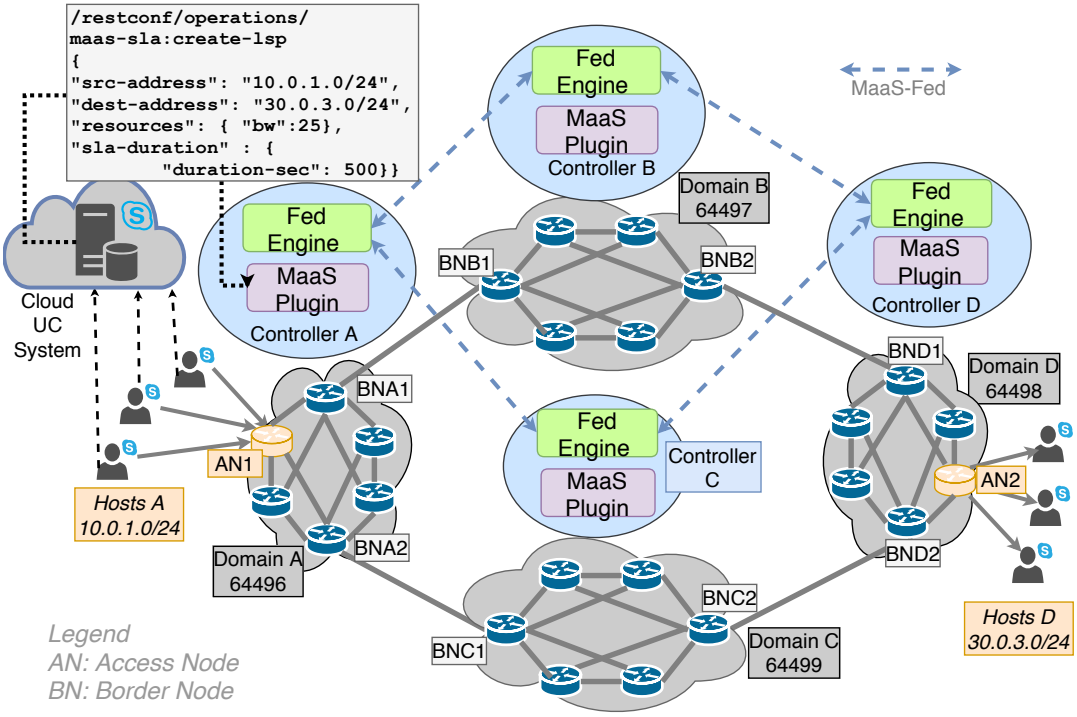


Fig. 3.8 Example scenario and network topology to demonstrate end-to-end QoS provisioning.

As stated briefly previously, end-to-end LSPs can simply be provisioned if each domain between the inter-domain endpoints sets up tunnels locally, while a LSP service-request protocol like MaaS-Fed functioning to ensure coherence between different domain provisioning processes. Fig. 3.9 sequence diagram and below description illustrate this for our above Fig. 3.8 scenario.

- *Domain A MaaS:* Upon receiving a *create-lsp* request, domain A MaaS has to figure out the local network nodes to compute and construct LSPs from. While source endpoint

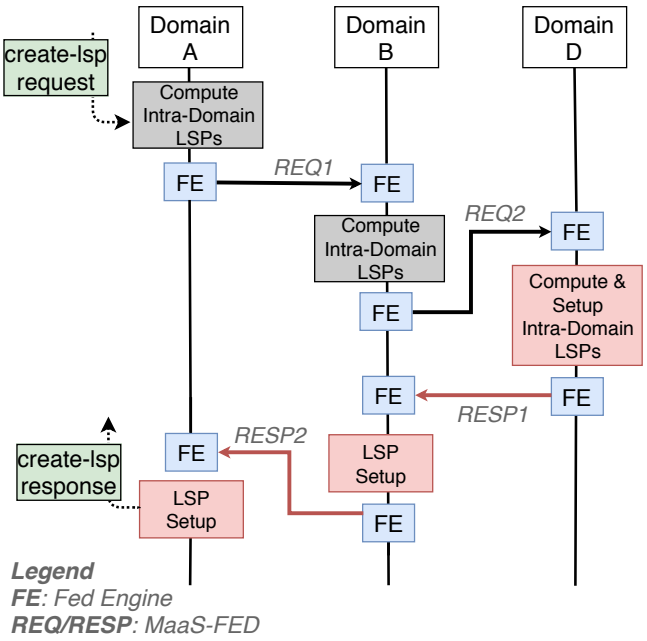


Fig. 3.9 Sequence diagram illustrating end-to-end LSP provisioning process in reference to the Fig. 3.8 scenario.

node can be easily queried through local Routing Information Base (RIB), which is AN1 in this example, the destination endpoint node will be a border-node (BN) and has to be queried through BGP RIB. In this example, the RIB would output two possible exit BN - BNA1 and BNA2 (Fig. 3.8) and the MaaS plugin can select any choosing the former in this case. This exit BN is the gateway from where destination hosts can be reached, and thus LSP service-request is propagated further to the eBGP domain peered with this selected BN i.e Domain B. The LSP service-request is communicated via MaaS-Fed REQuest message (*REQ1 Domain B*, Fig. 3.9). Prior to that, intra-domain LSP setup protocol as explained earlier in section 3.3 is initiated, with the difference being computed local tunnels are not signalled on the data-plane yet as the source-domain cannot be sure if the LSPs will be provisioned successfully in the end-to-end network or not (*compute LSPs, Domain A*, Fig. 3.9).

- *Domain B MaaS:* For any domain MaaS plugin, significance of receiving a *REQ* message is akin to receiving a *create-lsp* API request from customer. Therefore, a process similar to previous domain A is also performed at domain B – compute the local LSPs but restrain from signalling; determine a next-hop domain via BGP-RIB; and propagate LSP provisioning REQuest to it (*REQ2 Domain D*, Fig. 3.9).

- *Domain D MaaS:* Domain D while processing a REQ message figures out that it itself is the destination domain, as its BGP-RIB would have indicated that the destination endpoints are hosted locally (*30.0.3.0/24,* Fig. 3.8). So contrary to the previous domains, it computes as well al also signals them LSPs on the data-plane. Thereafter, it propagates a MaaS-Fed RESPonse message in the reverse upstream direction (*RESP1*, Fig. 3.9).

Domains that receive a RESP message are assured that LSP provisioning in downstream domains has been successful, as a result the LSP tunnels that were only previously held in the control-plane can be confidently signalled on the data-plane (*LSP-Setup, DomainA and DomainB,* Fig. 3.9). Finally, as also mentioned previously, information on provisioned segments is shared back with the customer as a response to the create-lsp API request (Fig. 3.9). For reference, the provisioned tunnels in this case are shown illustrated in Fig. 3.10 along with their Labels.

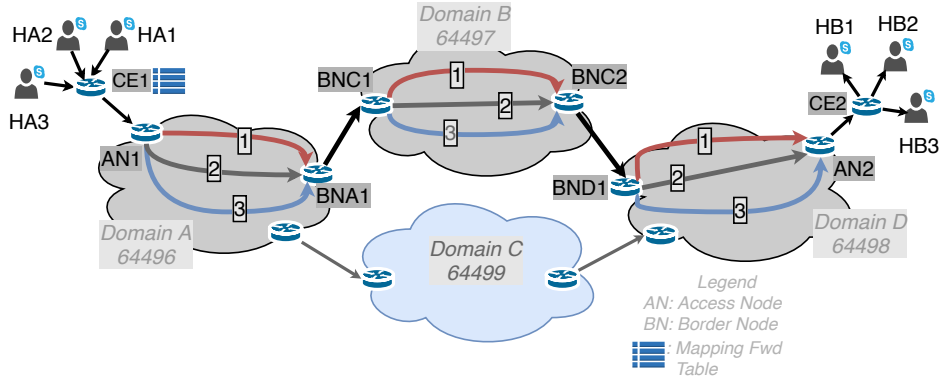


Fig. 3.10 Illustration depicting LSPs provisioned by each domain in reference to the Fig. 3.8 example scenario.

A. MaaS-Fed messages

Different messages exchanged during the previous inter-domain QoS provisioning scenario of Fig. 3.9 are shown labelled in their JSON format in Fig. 3.11 and explained below. Recollect that these MaaS-Fed messages are transported as an attribute of BGP route update message referred to as a *controller route* and aided by Federation Engine, which Fig. 3.11 further stresses on.

- *Service Parameters and uuid:* The common and most important fields of every *MaaS-Fed* message be it *REQ* or *RESP* are the LSP service parameters, which are exactly

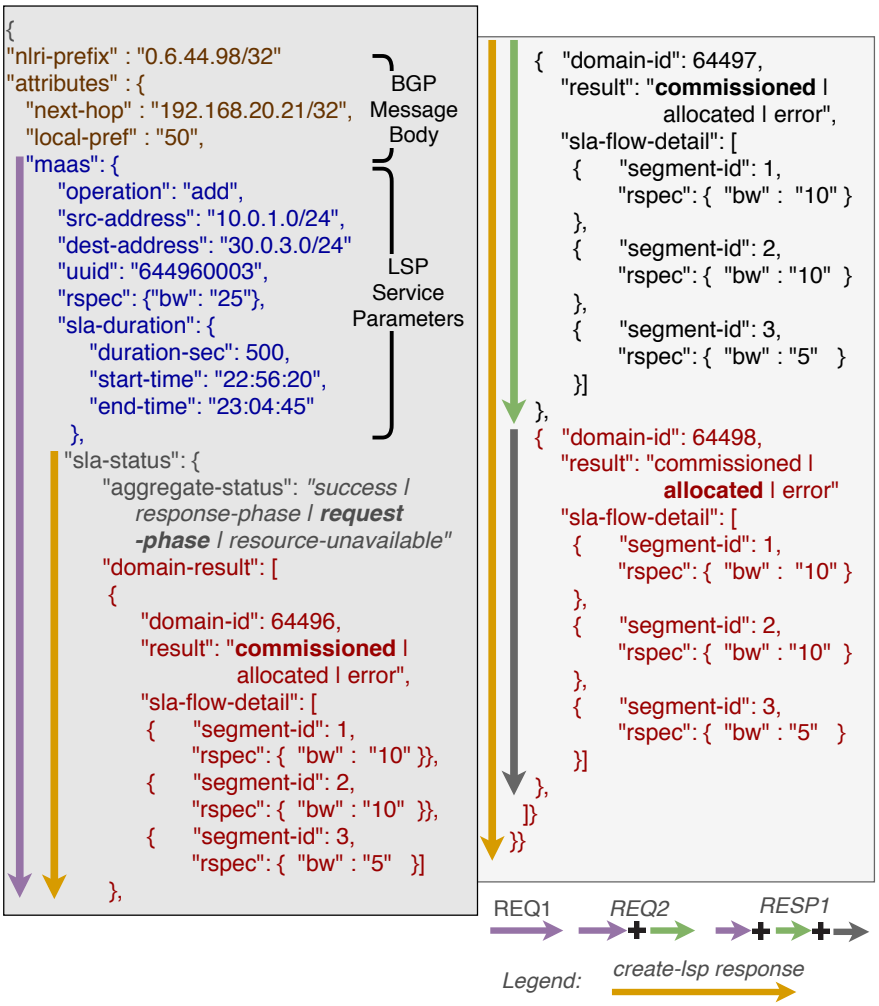


Fig. 3.11 Different MaaS-Fed messages illustrated in their JSON format in relation to the considered example scenario.

similar in structure as received from customer via *create-lsp* API at source-domain, except for the addition of an *uuid* field also by the source-domain (Domain A, Fig. 3.8). UUID represents a federation global identifier for all the tunnel segments, or a particular customer request (Fig. 3.11) which is helpful in tasks of updating or terminating the service.

- *REQ:* The purpose of REQuest message is to communicate the LSP service-request received at the source-domain via *create-lsp* API further ahead to intermediate and destination domain in order to effect uniform end-to-end provisioning. LSP service parameters in the *REQ* message therefore fulfil this purpose (Fig. 3.11).

Another function derived from this message is to communicate the *local* LSP processing detail to all the other domains. This is done through *sla-status* object wherein every domain adds its information on LSP processing through a new *domain-result* object. Notice *sla-status* object in *REQ1* message (Fig. 3.11) propagated by source Domain A containing its processing details, specifically list of all the LSP segments, their Labels and allocated QoS/BW. *REQ2* message propagated by Domain B appends its own *domain-result* object. This addition makes each domain aware of end-to-end LSP state, essentially source-domain in our context, and is beneficial for their efficient utilization as will be brought to light later.

- *RESP:* RESPonse message travels the same path taken by REQ messages, except in the reverse upstream direction from destination domain to the source domain as evident from Fig. 3.9 sequence diagram. Its purpose is to convey the upstream domains that QoS or LSP setup has been successful in downstream domains, and they can signal or create tunnels in the data-plane which were earlier only present in control-plane. Similar to REQ, this message also carries the *sla-status* object, and domains propagating this message update their *domain-result* object's *result* field from commissioned to allocated in order to reflect the installation of tunnels on forwarding-plane (*RESP1*, Fig. 3.11).

- *create-lsp response:* Finally shown in Fig. 3.11 is the *create-lsp response* object primarily comprising of the *sla-status* object as received from the RESPonse message at source-domain. Sharing *sla-status* with the requesting customer provides them the information on provisioned tunnels and they can themselves take control or utilization of them tunnels as will be better brought to light next.

3.6.3 Error Handling and Service Re-negotiation

A. Motivation

The previous hypothetical scenario demonstrated MaaS and MaaS-Fed under ideal conditions wherein the amount of network resources requested by the customer were considered to be adequately present in the multi-domain network. However practically considering, there will be circumstances wherein one or more of the involved domains might have insufficient resources (bandwidth) compared to what is desired for leading to complete service failure. For this reason, error-handling has to be built into distributed QoS framework (MaaS) and

protocol (MaaS-Fed) like ours so that alternate measures can be invoked, such as discovering new inter-domain paths.

B. Advantage- *Adaptivity*

The term *crankback* is used for referring to this mechanism in distributed service-request protocols wherein error or failure in a *provisioning* node is notified back to upstream *requesting* node so that it can invoke alternatives (Farrel et al., 2007). In legacy LSP service-request protocol - inter-domain RSVP-TE, *crankback* is built around only propagating back error notification to the requesting domain without any added information. However, considering proposed shift in inter-domain control-plane from between border-routers to between SDN controllers, crankback mechanisms in future service-request protocols can be expanded to also share back extra information from the point of error-occurrence such as – *partial resource availability and time in future when the full request can be met*, acting upon complete network-view in SDN controller. This information can help support *service re-negotiation* or *re-configuration* at the source-domain or customer end, and accounts for big improvement over simple legacy signalling that leads to complete service rejection, as we build in MaaS-Fed. This also makes the MaaS framework *adaptive* as both application and multi-domain network can adjust to intermittent resource outage and shortage.

C. Example Scenario

To demonstrate this adaptive service renegotiation feature of our framework, we take the previous example scenario again and simulate an erroneous resource constraint situation in one of the intermediate domains, so that error-handling (crankback) and service re-negotiation mechanism gets triggered. Fig. 3.12 thus shows the resultant protocol sequence in creating end-to-end LSPs for this modified Fig. 3.8 example scenario.

In the scenario, Domain A MaaS receives a QoS request from the application customer (*create-lsp request 2*, Fig. 3.12). After processing the request internally by computing and reserving the LSP resources, it propagates a MaaS-Fed *REQ* message to neighbouring domain B (*REQ1*, Fig. 3.12), for end-to-end provisioning in a process similar to previous original one. However, domain B finds itself short of bandwidth resources as they all have been utilized in bulk by a prior alternate QoS request (*create-lsp request 1*, Fig. 3.12). As a result, domain B invokes crankback mechanism and sends back error information using MaaS-Fed *REQ_ERROR* message. Upon receiving this REQ_ERROR message, Domain A looks for

alternate inter-domain path to the destination endpoints in Domain D, and in this particular example scenario, there is one available successfully via Domain C (*REQ2*, Fig. 3.12).

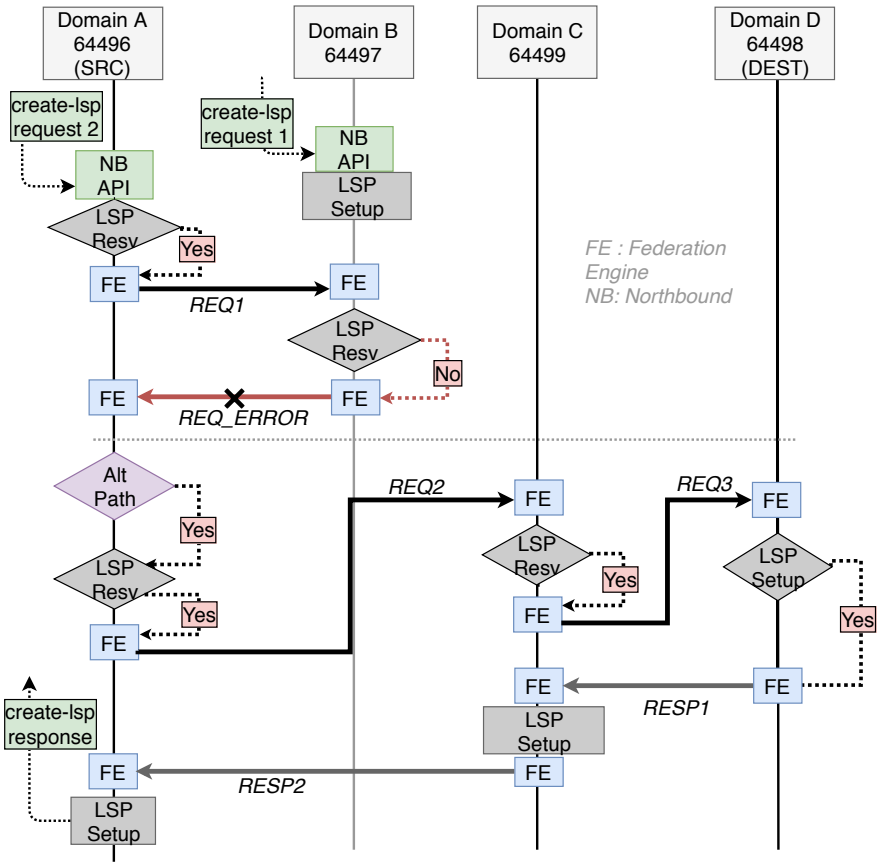


Fig. 3.12 Sequence digram explaining the process of end-to-end LSP setup in case of an error in one of the intermediate domains, in reference to Fig. 3.8 topology.

The service renegotiation mechanism in MaaS(Fed) however only gets triggered when the customer's demand is not met through any of the inter-domain paths. To synthesize this, hypothetical scenario considered above is now expanded to create a resource-constraint condition in both of the intermediate domains B and C so that original service terms are not met through any of the inter-domain paths. This process is demonstrated with Fig. 3.13 sequence diagram. Source-domain A upon receiving a QoS request (*create-lsp request 3*, Fig. 3.13), propagates the request further to Domain B (*REQ1*, Fig. 3.13), but receives a *REQ_ERROR(1)* message from there due to lack of network bandwidth resource. Therefore, it tries an alternate inter-domain path via Domain C (*REQ2*), and a similar error notification is received from there as well (*REQ_ERROR2*, Fig. 3.13).

Since original service-request cannot be satisfied, *service renegotiation* mechanism is initiated at source-domain A. The enabler here is the MaaS-Fed's *REQ_ERROR* messages. These are propagated by the erroneous domains with a *cb_support* object containing relevant error information - partial resource availability, and the time in future when the full service can

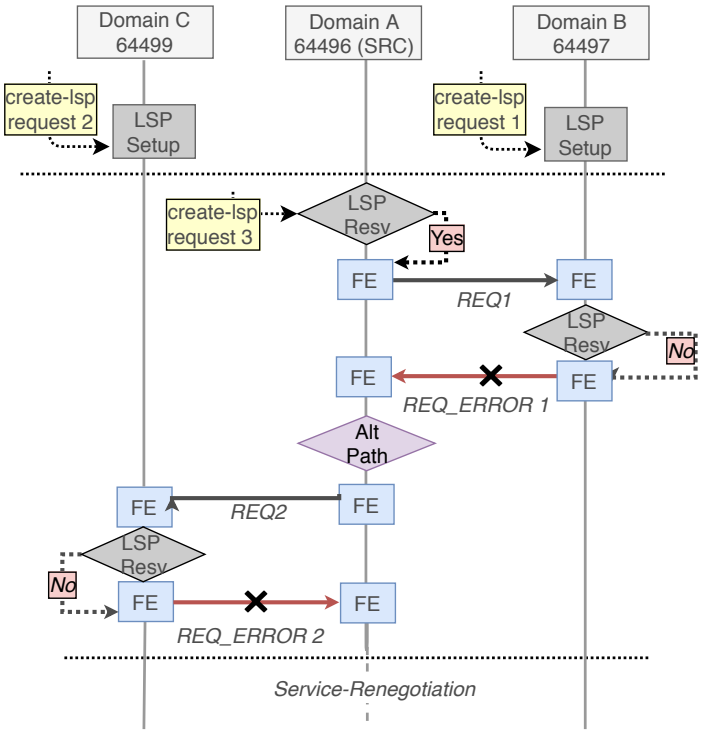


Fig. 3.13 Sequence digram showing the REQ_ERROR message propagated by domains in case of a service provisioning error, in reference to Fig. 3.8 topology.

be met. This object as shown in Fig. 3.14 provides necessary information to source-domain for negotiating revised service terms with the customer considering the resource outage and limited availability. Further, but not highlighted in any of the illustration, source-domain can share this aggregate error information with the customer, and upon approval, another *REQuest* for partial available resources can be initiated which can be confidently met. Additionally, LSPs created with partial resources are optionally upgraded to original full service terms in the entire multi-domain network once the complete desired resources are available, as would have been notified to even the customer through *cb_support's future-support* time field (Fig. 3.14).

This adaptive network behaviour enabled by MaaS courtesy of service-renegotiation mechanism results in optimum utilization of network resources. The major enabler here is the SDN Federation breakthrough, as through it a flexible communication mechanism between multi-domain network and application is enabled i.e. MaaS-Fed. It also helps achieve deterministic network performance as application is kept in loop of any irregularities, providing it the ability to also mitigate on its part.

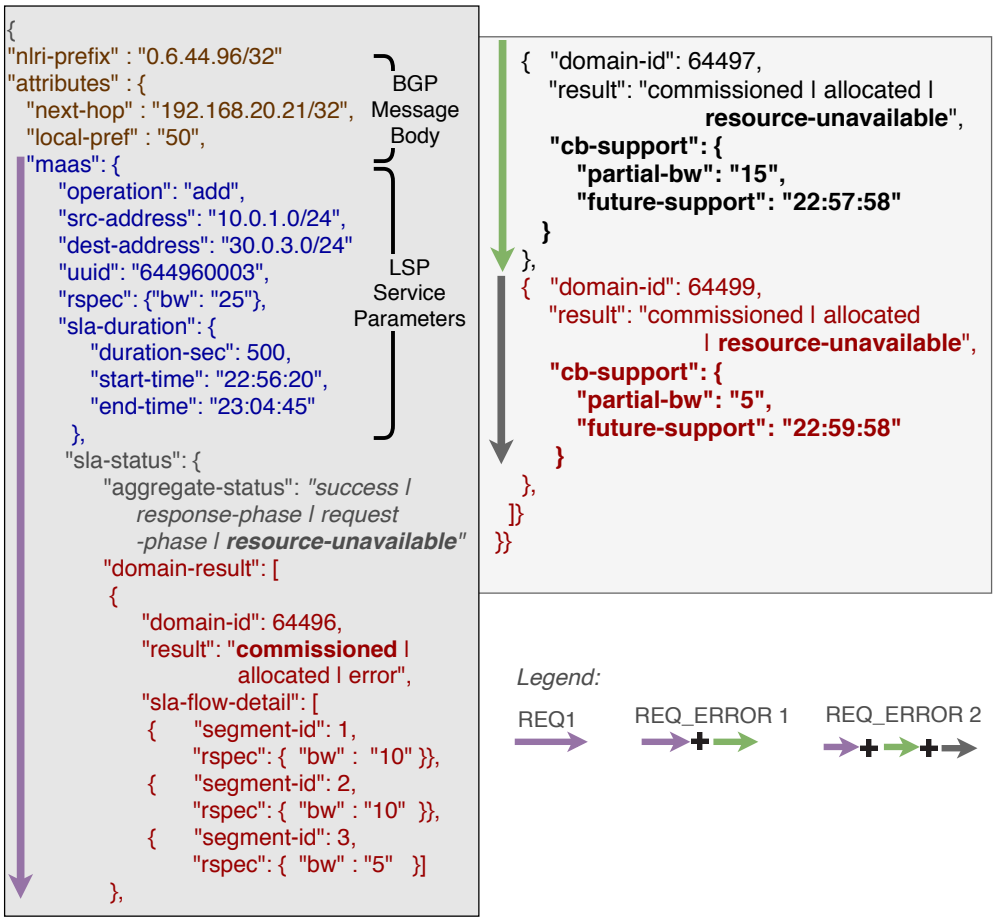


Fig. 3.14 Illustration to depict MaaS-Fed REQ_ERROR message in reference to the considered error-handling scenario.

3.6.4 Customer delegated control of tunnels

Previously we looked at creating LFEC table entries in data-plane, for the purpose of mapping one or many IP flows into a provisioned Label-switched tunnel (Sec. 3.4.5). Different approaches for creating LFEC entries were discussed in there, and specifically the benefits of delegating their creation to customers to be done in CPEs was proposed, but only for an intra-domain scenario. This subsection extends upon that concept and demonstrates the benefits of customer-delegated mapping or control of tunnels for an inter-domain scenario, which is facilitated by MaaS-Fed.

Traditionally, LFEC mapping would have been performed by every domain's MPLS control-plane through encapsulation entries at network ingress (*AN1 BN1 CN1*, Fig. 3.15) and decapsulation at penultimate hop or egress (*AN2 BN2 CN2*, Fig. 3.15). Since in MaaS, source-domain gets hold of every other domain's provisioned segments and *Label* information

through MaaS-Fed *RESP* message, mapping to each domain's LSPs can instead be done only once at source-domain ingress (*AN1 DomainA*, Fig. 3.15), with Top of Stack (ToS) Label still getting decapsulated at every domain egress. Such LFEC entries for the hosts and tunnels in Fig. 3.15 are shown in Table 3.1. However, there are advantages in delegating this mapping responsibility and location to customer domain as briefed below.

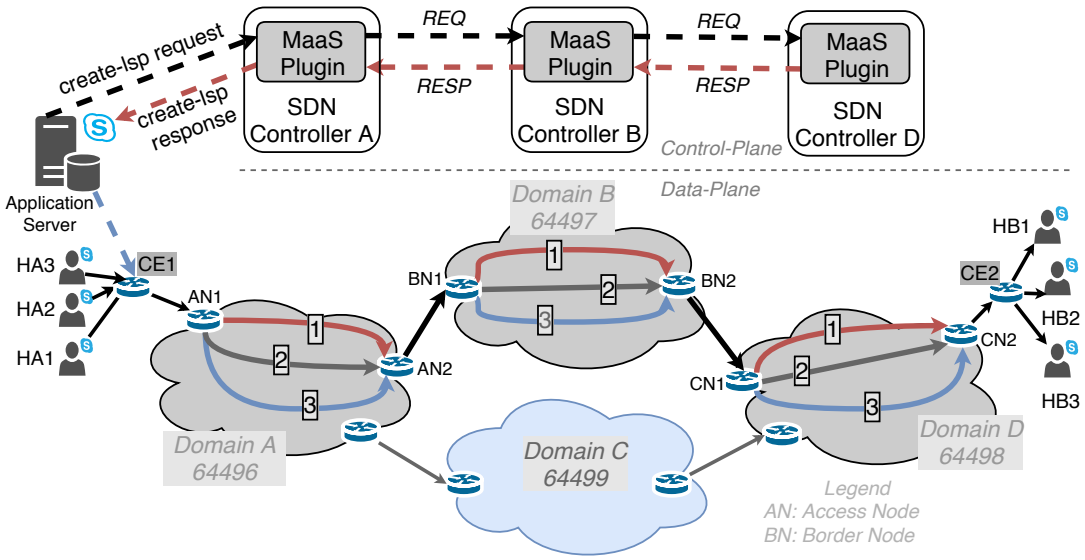


Fig. 3.15 Illustration showing empowering application customer (*via create-lsp response*) to perform LFEC mapping in their own premises (*CE1*).

IP Flow	Source	Destination	Allocated Label Stack	Resultant BW
Flow 1	HA1 - 10.0.1.11	HB1 - 30.0.3.11	1,1,1	10
Flow 2	HA2 - 10.0.1.12	HB2 - 30.0.3.12	2,2,2	10
Flow 3	HA3 - 10.0.1.13	HB3 - 30.0.3.13	3,3,3	5

Table 3.1 Mapping between IP flows and Label segments of Fig. 3.11 referred to as LFEC, only for reference in relation to Fig. 3.15 topology

- Network domains in handling IP to Label mapping process, tunnel incoming customer flows by load-balancing them into one of the segments. This process treats all the flows similarly which is not supportive of Traffic Engineering. Domains can still prioritize some flows based on observing DSCP values, but it is customer application itself which possesses the true and updated knowledge on how much of its purchased bandwidth should be rightly allocated to differently flows or IP subnets, VLANs,

Diffserv codepoints, etc. Thus, delegating mapping to customer application promotes a dynamic, real-time and network-independent Traffic Engineering process.

- Customer delegated mapping also enables coarse-grained scalable operations, as an application customer can buy bulk bandwidth for its many flows and then allocate or sub-divide them by simply allocating them a segment or a Label from every domain (Table 3.1). For instance, customer application can create LFEC mapping entries shown in Table 3.1 in its own authority device's forwarding table (*CE1*, Fig. 3.15), flows enter the end-to-end network already encapsulated with appropriate *Label* and thus enjoy designated QoS levels.

The benefits of customer-delegated mapping or control are even better realized if the entire e2e network between inter-domain endpoints is made MPLS-enabled, and the application can manipulate flow-agents to produce packets already containing right Label headers that decreases the encapsulation latency incurred later in the traffic path. For instance, Table 3.1 LFEC mapping can be done at flow source directly (*Hosts HA*, Fig. 3.15), instead of at CE1 in there. One of the major challenges with this approach is that since Label headers for all the domains are added at once creating a stack, there is an overhead as a result of increased packet size. Though this can be mitigated by further progressing MaaS-Fed communication between MaaS plugins to allocate global Labels for the entire Federation.

The above described properties of MaaS- end-to-end QoS, service renegotiation and customer-delegated mapping are further better brought to light when functionally demonstrated later in chapter V. Prior to that, a testbed network infrastructure is required for MaaS to be built and deployed on, as explained in the following chapter.

Chapter 4

WaterFed Testbed

A network Testbed is required that provides us with a data-plane (switches, routers, links, end-hosts) to develop as well as demonstrate QoS/LSP provisioning gains of MaaS. Additionally, a SDN control-plane is also required that makes the data-plane function and creates a network abstraction layer to make the process of developing MaaS plugin easier. This abstraction layer consists of services like path-computation, flow-programmer, BGP and Openflow southbound plugin which were also referenced in the previous chapter. This chapter explains the design of such a state-of-the-art Testbed which along with MaaS represents a significant effort and outcome of this Thesis.

4.1 Introduction

A. Requirement

A Testbed is an important component of laboratory focussed research and development initiatives, and empowers rapid prototyping, feasibility studies and demonstration of a solution. The character of a Testbed is in close replication of its target infrastructure or world, which can be achieved through commonly established means like simulation, emulation or a small scale modelling.

We also require a Testbed in this project to develop and deploy our proposed application-centric QoS framework - MaaS. Since our target environment is multiple autonomous network domains as in Internet such as ISPs, cloud service providers, or enterprises without any loss of generality, we require our testbed to minimally mimic similar infrastructure, and essentially the network domains should be software-controlled (SDN).

While the requirement from Testbed *data-plane* is straightforward i.e. to be able to provision multiple network domains, the high-level requirements from the *control-plane* are as listed below.

1. **Intra-domain routing (SDN-IP control-plane):**
   The testbed control-plane should be able to proactively provision intra-domain IP routing among the testbed hosts located in same domain, and that too in the SDN context, referred to sometimes as SDN-IP routing (ONOS, 2016). The broad difference being that in legacy IP networks, routing is controlled distributedly through IGP protocols like OSPF/ISIS functioning alongside data-plane, whereas in the SDN-IP context, it is configured centrally by routing services residing in SDN controller which remotely manipulate device forwarding tables.

2. **Inter-domain routing (SDN-BGP control-plane):**
   Secondly, the testbed control-plane should be able to provision routing between the hosts residing in different network domains. Since eBGP is the de-facto routing protocol in the Internet for such purposes, similar construct should be maintained and BGP should aid inter-domain routing.

   Fulfilment of the above two, provides us autonomously functioning networks just as in Internet through our Testbed.

3. **Network Control Abstraction Layer**
   Lastly through the testbed control-plane, we wish to have a network control abstraction layer that provides high-level APIs to be leveraged in the rapid and speedy development of *MaaS* plugin. This concept of abstracting the complexity of underlying domain control-plane for application developers is one of the core philosophies of SDN, and many open-source SDN controller frameworks provide off-the-shelf plugins for this abstraction layer, which we further extend using plugins like *path-computation* and *flow-programmer*.

B. Background

The commonly available means to design a functioning network Testbed are simulators, emulators, or large-scale physical experiment networks as described briefly below.

1. *Simulators*: As the name suggests, they simulate different network topologies and protocols as a software process. Some examples of simulators are NS-2/3 or OMNET,

however, although they provide rapid iteration with different network scenarios, they lack the reconfigurability, control and environment of actual network devices and do not involve any real packet flow, and the solutions developed for the same cannot be confidently ported to the target network.

2. *Emulators*: Network emulators provision a virtual network that mimics an actual physical network, provide an interface and configuration similar to hardware ones, can process a real network flow packet as them, and can be extended or interfaced with other physical or virtual networks. They also provide tools to inject synthetic imperfections into the network such as latency, jitter and limiting maximum throughput.

3. *Large Scale Physical Testbeds*: While emulators provide the ability to configure virtual network even in a small environment like personal computers, there is an availability of large scale testbeds that allocate a mix of virtual and physical network as well as compute resources, collectively referred to as a slice, for multiple tenant user/projects on much powerful production-grade remote switches and servers (Huang et al., 2016).

To take a pick from above techniques for our purpose, Large-scale Testbeds are not easily accessible plus migrating and deploying new services from local development environment to remote servers induces overhead in provisioning time. Moreover, they are surplus to our requirement when an ***emulator*** can provide a somewhat similar virtual network environment locally, adequate and convenient for our experimentation purposes. Lack of resources though in personal computers for emulating a large network are more than compensated by access to a large Openstack based private cloud hosted herein Telecommunications Software and System Group (TSSG), Waterford Institute of Technology.

C. Overview

In choosing to go ahead with *emulators*, this section presents a brief overview on how we emulate a multi-domain network that fulfils our requirements stated previously. At the start of project and even now, there wasn't any available emulator tool that could off-the-shelf provision required data-plane readily for our purposes, neither a SDN controller distribution that could readily provide the desired control-plane services. This provided us with an opportunity to custom or purpose build a state-of-the-art Testbed utilizing many commonly available technologies and open-source tools. The detailed design is presented in following sections, while a brief overview of involved components and approaches is presented below.

- *Mininet* to emulate a virtual network domain topology featuring SDN/Openflow capable switches (OpenVSwitch).
- Modified *ODL* SDN controller as the control-plane of each emulated network domain.
- Testbed network-domain majorly comprising Mininet data-plane and ODL control-plane packaged as a *VM image* for automated replication and provisioning, as well as to enforce necessary segregation.
- *Virtual Extensible LAN (VXLAN)* tunnel to indirectly connect border nodes of neighbouring network topologies/domains residing in different VMs. This vxlan tunnel thus also acts as the inter-domain link.
- *Quagga* utilized to run BGP on border nodes- eBGP peering with other border nodes in neighbouring domains, and iBGP session with domain control-plane i.e Opendaylight SDN controller.
- Finally, we use *Vagrant* to automate deployment and setup of Testbed on Openstack cloud.

Named as WaterFed for the motive behind it being to progress federation among network domains, this testbed represents an important outcome of this Thesis and its design and development undertook a significant effort. In trying to decode its development, it can be viewed from two perspectives- data plane and control plane. Although purpose-built for helping in research and deployment of our focused application-centric QoS framework, the testbed data-plane and to a large-extent control-plane as well are generic enough to be utilized for other future SDN Federation research activities in the institute or in the wider community with the planned open-sourcing of extended ODL plugins and other scripts after refinement. In the next two sections, both the Testbed planes are explained in detail.

4.2 Testbed Data-Plane

WaterFed data-plane is the logical layer that comprises multiple *Mininet* virtual networks, with each of them acting as an autonomous domain and constituting – virtual switches, links and end-hosts void of any control service or protocol. Data-plane to be over-viewed can be further sub-divided into *intra and inter-domain* components as below.

4.2.1 Intra-domain Topology Components

In materializing the wider multi-domain Testbed, a single virtual domain was designed and tested first utilizing Mininet and Open vSwitch as explained below.

A. Mininet Virtual Network

Considering that we chose to emulate our Testbed, Mininet, a popular network emulator is thus a critical component. It provides convenient Command-Line Interface (CLI) and Python APIs to provision a virtual network topology comprising multiple switches, links as well as hosts on a single system as explained below (mininet, 2018).

- ***Virtual Hosts:*** Mininet utilizes *process-based virtualization* to spin up multiple virtual hosts as individual running process of a physical or virtual Linux machine (mininet, 2018). It also makes use of *network namespaces* support in Linux kernel to provide these virtual hosts (processes) with separate network interfaces, routing and Address Resolution Protocol (ARP) tables (*H1 H2*, Fig. 4.1). These virtual hosts behave just like real hosts, and we can login to them via ssh and execute any program located on root machine. We utilize them as end-users, traffic-generators or flow-agents of our emulated network.

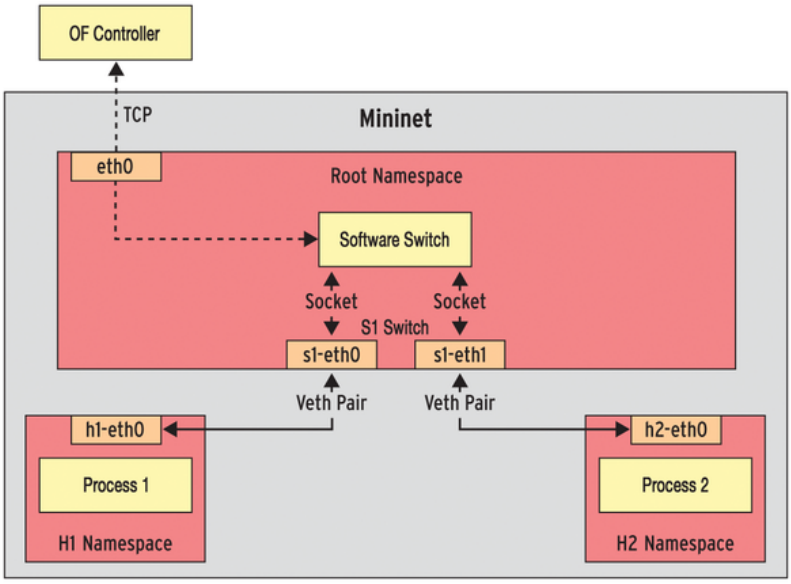


Fig. 4.1 Mininet overview depicting virtual software switches, virtual hosts as OS processes and virtual links as Veth pair. Image from Wette.

- ***Virtual Links:*** Virtual hosts are connected to a software switch via virtual ethernet (Veth) interface pairs as shown in Fig. 4.1 *(H1, H2)*. These *Veth* pairs function as a virtual link to which characteristics like bandwidth, latency and packet-loss are modelled via Linux traffic control (tc) utility.

- ***Virtual Switches:*** Apart from virtualizing hosts and links, Mininet also provides APIs to emulate a software switch supporting default *Linux Bridge or Open vSwitch* for these purposes. We utilize the latter which is another important enabler of our Testbed in itself and is explained next.

B. Open vSwitch

Open vSwitch (ovs) is an open-source production quality software switch which we incorporate in our Mininet emulated network topology. It comes with a powerful CLI of its own, although we instead use Mininet Python APIs which are rather convenient and high-level in order to instantiate multiple of these switches, link them together, and with virtual hosts in a desired topology.

Running in software, *ovs* provides the behaviour of a vendor switch with the difference being that no network control service runs on it. The switch datapath (forwarding table) is instead accessed and configured through popular *Openflow* southbound protocol from a SDN controller, and its interfaces configured for bandwidth and queues similarly through related OVSDB management protocol (Fig. 4.2), thus suited for SDN. Still in active development, we use ovs version 2.5.2 which supports rich set of data plane features specifically MPLS Label Switching, queuing and metering that are also prominent for our emulation. Appendix C.3 gives a brief overview on how this switch implements these QoS and Label switching features.

4.2.2 Final intra-domain topology

Fig. 4.3 shows the final domain topology undertaken for further experimentation through developing control-services and demonstrating MaaS on. There were iterations of experimental topologies before finalizing on this one. The chosen topology as can be seen has multiple IP subnets and multiple path between devices to enable multi-path routing, the objective of this all being to have a respectable yet simplistic and consistent network for experimentation. A single Python script incorporating Mininet and ovs APIs as well as common Linux commands provisions such a network which makes the process of its replication for achieving multi-domain topology easy and efficient.

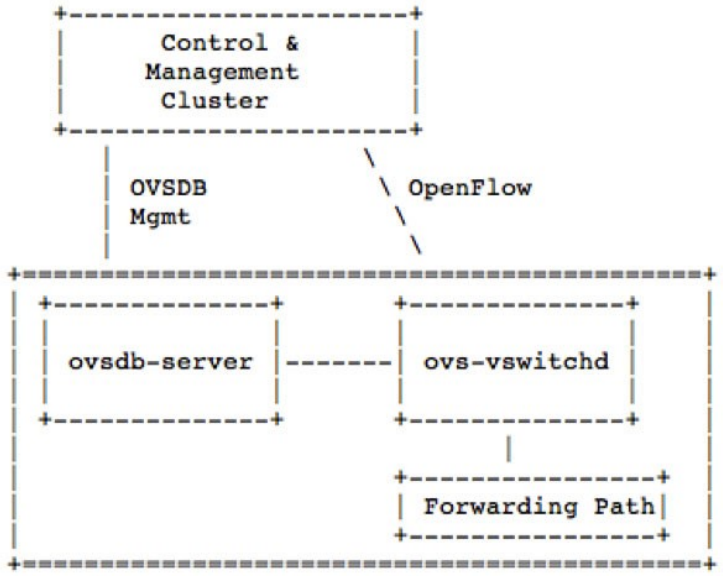


Fig. 4.2 Open vSwitch overview schematic showing its control and management protocols - Openflow and OVSDB. Image from ovsdb-ietf (2013).

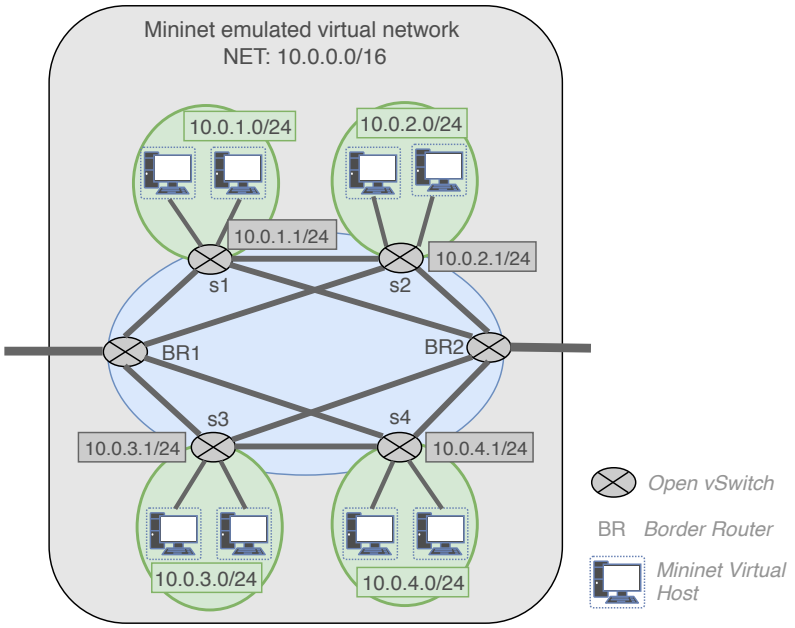


Fig. 4.3 Domain network topology utilized for experimentation.

Although this specific topology was used while developing and testing control-plane services like MaaS on, these services though are built generic enough to work for any other arbitrary L3/L2 topology. There are other specific configurations that went into the emulation

of such a network beyond simple orchestration of mininet APIs. This includes bootstrapping tasks like statically configuring IP, SSH, ARP and routing tables on virtual hosts, switches as well as on root host. These are essential to setup the network minimally before control-plane of SDN controller takes over.

4.2.3 Inter-Domain Extension

In simpler terms, to achieve a multi-domain network testbed, we replicate the Fig. 4.3 mininet virtual network multiple times, each in a new VM. Thereafter, the then neighbouring network domains are connected via VXLAN tunnels between their border Ovs nodes which act as inter-domain links (*vxlan,* Fig. 4.4). Apart from this, eBGP is configured over these inter-domain links for facilitating inter-domain route exchange (Fig. 4.4), and also measures for swift and automated provisioning of Testbed on cloud through a Vagrant provisioning tool. These individual components and processes that went into expanding the previously explained intra-domain network into multi-domain one are detailed below. Fig. 4.4 herein labels and pictorially illustrates the discussed components.

A. Virtual Network to Virtual Domain

We can replicate a Mininet *virtual network* multiple times, but that will not necessarily present us with as many *virtual network domains*. In qualifying a network to an autonomous domain, the most basic requirement is to have an isolated autonomous control-plane. A modified Opendaylight SDN controller, detailed in the next section, thus forms that required control-plane for each of our testbed domain. *Therefore, to achieve an inter-domain topology, apart from replicating the data-plane i.e. Mininet virtual network, we also replicate its control plane i.e. SDN controller as shown in Fig. 4.4 (not shown for Domain 1).*

B. Computational Limitation and separation into VMs

Multiple *mininet* virtual networks can be easily emulated on a single personal computer, as *mininet* can scale up to 100s of software switches and hosts on a single personal computer. Our Fig. 4.3 reference topology though only comprises of 6 switches and less than 10 hosts, and therefore 4-5 network domains or mininet networks, which are sufficient for an early-stage experimentation Testbed, can be easily provisioned on a single personal computer.

However, multiple *mininet* networks would also warrant their respective SDN control planes or ODL controllers. Each Opendaylight instance requires about 6GB of RAM, so more than 2 controllers cannot be easily instantiated on a single personal computer. Therefore, the

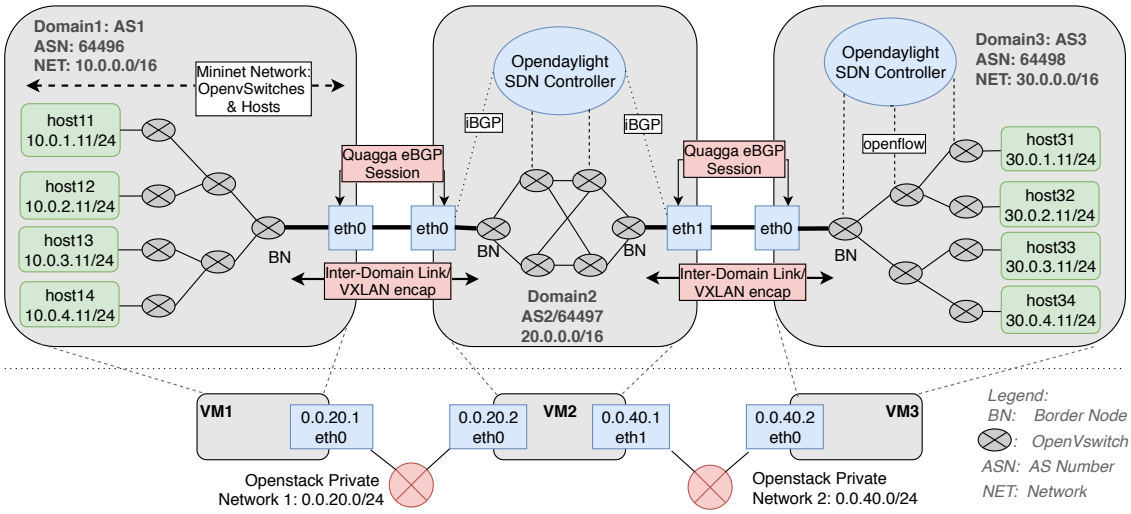


Fig. 4.4 WaterFed Testbed schematic showing multiple network domains with essential components labelled.

testbed is instead provisioned in the institute's Openstack private-cloud. All the scripts and components required to instantiate a virtual domain are thus packed in a base VM image, uploaded on the cloud (glance image repository in Openstack), and from there they are easily spun and replicated to achieve multiple network-domains in multiple VMs (Fig. 4.4).

The benefits of abstracting a virtual domain into VM are foremost scalability as elastic and flexible computing resources are made utilized for the testbed as a result of cloudification. Secondly, it makes the Testbed provisioning process easier as through cloud APIs, the process can be automated as explained later and potentially large multi-domain topologies can thus be created.

C. Inter-Domain Links

Testbed virtual domains in different VMs though have to be linked together to create what would be inter-domain links. A private network on the cloud is thus created for this purpose for each of the inter-domain link, as shown in Fig. 4.4 and also specifically in the below figure. Neighbouring domain VMs in our Testbed topology are assigned an interface each from a shared Openstack private network (*192.168.1.0/24*, Fig. 4.5) which Openstack provides REST APIs for, and thereafter these interfaces attached to the ovs border nodes (BN) on either side which Mininet provides APIs for, to enable an in-direct inter-domain link (Fig. 4.5).

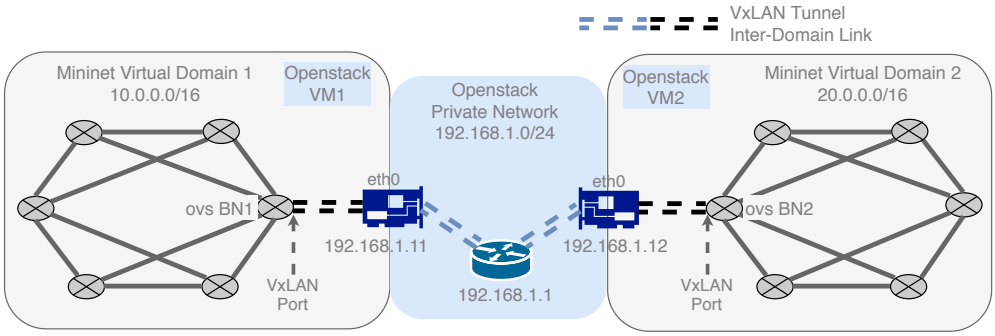


Fig. 4.5 WaterFed Testbed Inter-domain link and VXLAN tunnel schematic.

D. VXLAN Tunneling

Each of the virtual network running in a separate domain VM will have a distinct routing space, with BGP operating to exchange routes inter-domain for enabling end-to-end reachability. Notice *10.0.0.0/16* for Domain1 and *20.0.0.0/16* for Domain2 in Fig. 4.4 as also shown in Fig. 4.5. However, for instance, an IP packet travelling from one domain network to another will not be routed, as it cannot overcome the provider (Openstack) physical or virtual network (*192.168.1.0/24*, Fig. 4.5), which does not have the route to any of the mininet virtual networks. Therefore, we provision VXLAN tunnelling between the border nodes of neighbouring domains. Any packet that leaves the VXLAN interface of *ovs* border node is automatically encapsulated, tunnelled to the other end, and decapsulated there.

For instance, a packet destined for 20.0.0.0/16 when arriving at ovs BN of Fig. 4.5 VM1 is directed out from its VXLAN port which automatically encapsulates packet with an outer Layer4 UDP header with Layer3 destination IP as the other end of tunnel (*192.168.1.12*, Fig. 4.5) which can be easily routed by Openstack private network. Upon reaching the egress of tunnel at ovs BN2 of Domain 2, outer header is removed and original packet propagated further. Openvswitch (ovs) itself provides APIs to create VXLAN tunnels as shown in Fig. 4.6. This way, there exists a point-to-point inter-domain link between the neighbouring virtual networks overlaid on the cloud provider network (Openstack).

E. BGP Inter-domain Routing Support

BGP is the de-facto inter-domain routing protocol of Internet. Maintaining similar construct is thus essential for a Testbed that is to be used for research and development of an inter-domain service. Support for BGP is also required as we use it to form our generic inter-domain interface (Federation Engine) explained previously. While an ODL SDN controller would be the holistic *intra-domain* control-plane of each of our emulated network domain, BGP would

```
#Domain1
  ## Create a vxlan port and hence tunnel (part 1)
  ovs-vsctl add-port BN1 vxlan1 -- set Interface vxlan1 type=vxlan
   ↪  options:remote_ip=192.168.1.12 option:key=123abc
#Domain2
  ## Create a vxlan port and hence tunnel (part 2)
  ovs-vsctl add-port BN2 vxlan1 -- set Interface vxlan1 type=vxlan
   ↪  options:remote_ip=192.168.1.11 option:key=123abc
```

Fig. 4.6 Open vSwitch command line interface to create vxlan ports and hence tunnels, provided in relation to previous Fig. 4.5 scenario.

become the sole constituent of *inter-domain* control-plane propagating essential routing information. BGP is however not supported natively by *ovs* devices, therefore we bind and configure an external software BGP process (Quagga) onto our ovs border nodes (*BN*, Fig. 4.4). Quagga is an open-source software routing suite for Unix platforms supporting BGP among other protocols (Paul Jakma, 2017).

F. Automated Provisioning

Any network Testbed should be easy to setup, modify and delete. The requirement for multiple network domains in respective VMs makes the provisioning trickier and time-consuming in our case, and thus needs to be automated. Otherwise a great deal of manual effort would be required in spinning up domain VMs on cloud, configuring networks, copying latest *Mininet* topology, *Quagga* scripts and other development builds and binaries from local dev environment to each of these Testbed VMs.

We thus use *Vagrant* for automation which provides Ruby APIs to spin up VMs on different cloud environments, associate network interfaces to them from provider (Openstack) networks, configure security policies, install private keys, copy files from local development environment, and execute start-up scripts on VMs, among other features (HashiCorp, 2017). Running such a Vagrant script locally which in turn uses Openstack Rest APIs significantly reduces time and effort in provisioning of our Testbed.

Following is the list of tasks configured in *Vagrant* provisioning script for creating the desired WaterFed Testbed-

1. *Spin Domain VMs:* The first step is to spin up multiple instances of a virtual network domain with blueprint already manually stored as a VM image or snapshot on target cloud (*os.image*, Fig. 4.7). Vagrant just requires the Openstack API endpoint, access

credentials and base VM image name as inputs to be able to spin up desired number of VMs and thus domains (Fig. 4.7). Per-VM configuration (name, networks, etc) has to be separately supplied as explained next and in Fig. 4.8.

```
config.vm.provider :openstack do |os|
   ## Openstack Cloud API endpoint
   os.openstack_auth_url = 'http://10.10.20.20:5000/v2.0'
   os.username           = 'shasija'
   os.password           = '*********'
   os.flavor             = 'm1.xlarge'
   ## Base VM Image name
   os.image              = 'domain_vm7'
   os.security_groups  = ['odl','default']
end
```

Fig. 4.7 Vagrant provisioning script snapshot showing cloud access parameters.

2. *VM Network configuration:*

   Thereafter we script Vagrant APIs to link them with different cloud networks. Each domain VM is linked to a common public Openstack virtual network for management uses such as doing ssh into them, and also to one or more distinct private networks which act as an indirect inter-domain links as discussed previously (Fig. 4.8).

3. *Firewall and Security:* The firewall for the domain VMs can be configured through assigning an Openstack security group which specifies the socket connections allowed (*security_groups*, Fig. 4.7). Security groups are pre-configured and stored on the Openstack cloud, Vagrant plugin just allocates them to VMs and does not provide the capability to create them.

4. *Copying development builds:* The base VM image of our testbed network domain only contains the Mininet Libraries, Java Runtime Environment (JRE) for Opendaylight controller, Quagga for BGP, and other essential tools like xTerm and Wireshark. The mininet topology script and the controller builds are dynamically copied from the local DEV environment to every new Domain VM instance using Vagrant *rsync* APIs and then triggered remotely (*synced_folder*, Fig. 4.8).

Resultantly, we have an emulated network Testbed that can be automatically provisioned through minimum prior configuration and integrates the active development versions of Mininet topology script and Opendaylight SDN controller from the local system.

```
## VM1 (name=as1)
config.vm.define "as1" do |as1|
   as1.vm.provider :openstack do |os|
      os.floating_ip = "87.44.19.103" ## Public IP for VM
         os.networks=[
            {  ## Management Network uuid
               id:'960beef5-0c45-4799-9ad5-2ba36014f6fc'
            },
            {
               name:'pathman20',        ## Openstack private network name
               address:'192.168.20.21' ## and static IP to assign to VM
            },
            {
               name:'pathman40',
               address:'192.168.40.21'
            }
         ]
    end
    ## Local DEV Folder to copy to remote VM (as1)
    config.vm.synced_folder "/odl/karaf/target", "/odl/", type: "rsync"
end
```

Fig. 4.8 Vagrant provisioning script snapshot showing VM configuration parameters.

4.3 Testbed Control-Plane

Network control-plane refers to the logical layer constituting broad array of network services and protocols such as OSPF and BGP that serve the data-plane. Generally speaking, while the network data-plane actuates flow of packets by looking at the forwarding or routing tables, it is control-plane which is responsible for filling up these tables efficiently. Thus an efficient control-plane is necessary for optimized network operations, and resultantly has been a subject of significant research and developments with SDN and NFV as prominent recent proposals and drivers.

4.3.1 Overview

In the context of this Thesis, a network control-plane is really essential from 2 perspectives. Firstly, it is required for the basic functioning of our SDN WaterFed Testbed and should enable routing between end-hosts in the same domain and even inter-domain hosts. Secondly, a control-plane is required to formulate a network control abstraction layer (CAL) that would shield the complexity of network control and provide high-level abstract APIs for developing further network services such as MaaS. Such network CAL or control-plane

services were also previously heavily referenced while presenting MaaS in detail, for instance, *path-computation* and *flow-programmer* services in intra-domain LSP setup protocol, *BGP plugin* in Federation Engine and also in MaaS-Fed. These requirements for an SDN(IP-BGP) control-plane were also listed previously in the requirements subsection 4.2.1.A of this chapter.

Control-plane of a software-defined network takes the form of a SDN controller application. In this respect, many open-source SDN controller development platform are available majorly ONOS and Opendaylight which bootstrap and ease the development of desired control-plane. We use Opendaylight as the development platform of choice as suggested by the Work Packages. Our Testbed control-plane thus takes the shape of different plugins of Opendaylight SDN controller platform.

In this chapter next, brief overview of Opendaylight is presented followed by the design of our control-plane detailed from two perspectives pertaining to the requirements- individual *application plugins* which provide specific APIs to MaaS such as for accessing topology and path computation, and the two major *functions* required from control-plane which these plugins collectively perform- intra-domain and inter-domain routing.

4.3.2 Opendaylight

Opendaylight is an open-source SDN controller platform supported in development by many major network device vendors to advance the industry adoption of SDN. Its mission statement says "OpenDaylight (ODL) is a modular open platform for customizing and automating networks of any size and scale. The OpenDaylight Project arose out of the SDN movement, with a clear focus on network programmability. It was designed from the outset as a foundation for commercial solutions that address a variety of use cases in existing network environments (odl:overview, 2018)". Its major features are briefed below.

- *Active:* ODL has an active development community comprising over 1000 developers from over 50 industry members, courtesy of which many control-plane service plugins are readily available off-the-shelf such as Openflow and BGP significantly preventing the need for developing controller from scratch. The full list of ODL applications is available at odl:karaf (2018).

- *Modular:* ODL platform has a modular design which allows anyone in the ODL ecosystem to leverage services created by others; to write and incorporate their own; and to share their work with others. It is distributed as an Apache Karaf container

which provides users the ability to chose what features or plugins they want to deploy, and maintaining dependency between them.

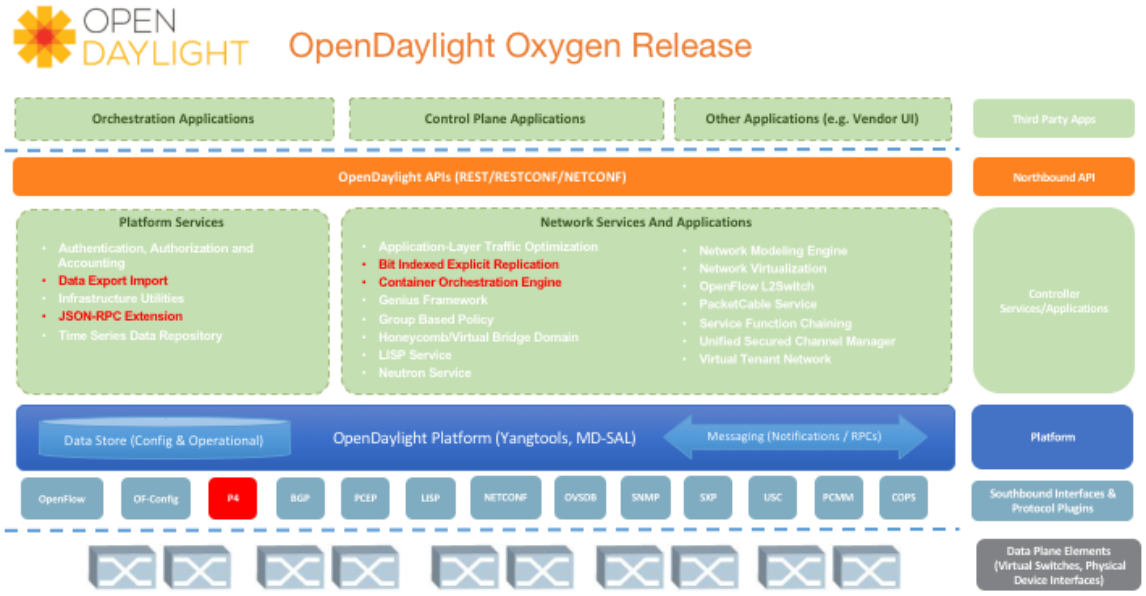


Fig. 4.9 General overview of Opendaylight architecture. Image from odl:carbon.

- *Model-Driven:* At the core of Opendaylight is its Model-Driven Service Abstraction Layer (MD-SAL) (Fig. 4.9) which provides messaging and data storage functionality to applications based on their defined data and service models. These models are defined through a domain specific language - *yang* (Bjorklund, 2010). A "producer" application model implements an API and provides the API's data; a "consumer" application model uses the API and consumes the API's data. The SAL matches producers and consumers from its data stores and exchanges information. A consumer application can find a provider that it's interested in. A producer can generate *notifications*; a consumer can receive *notifications* and issue *RPCs* to get data from providers. A producer can insert data into SAL's *data-store* (Fig. 4.9); a consumer can read data from SAL's storage (odl:overview, 2018).

MD-SAL is an important feature of ODL platform and is further explained in more detail in Appendix B through an example scenario.

4.3.3 Controller Applications/Plugins

In legacy networks, control-plane is built into the forwarding devices, and hence network operators are dependent on vendors such as Cisco for control services. SDN paradigm breaks this dependency by enabling softwarization, and the control plane could be custom built by operators themselves in the form of desired applications or plugins in SDN controller, a construct we also follow for our Testbed controller as shown in Fig. 4.10.

Referencing Fig. 4.9, Opendaylight or in general any SDN application can be built in 2 ways. Either they can be hosted inside the controller platform interacting with other application plugins using specific Java or generic MD-SAL API interfaces, or either hosted outside of controller environment and interacting with other applications using their REST APIs. For our testbed control-plane, we follow the former approach of building in-controller application plugins shown in Fig. 4.10 as it offers more flexibility and less dependency on REST APIs through which entire functionality might not be exposed. In this subsection, newly developed ODL plugins (*green*, Fig. 4.10) and other essential open-source ODL plugin which form the SDN control-plane of our testbed are briefed for their role and functions. These plugins have been heavily referenced earlier during the course of this Thesis and collectively fulfil the aforementioned requirements from Testbed.

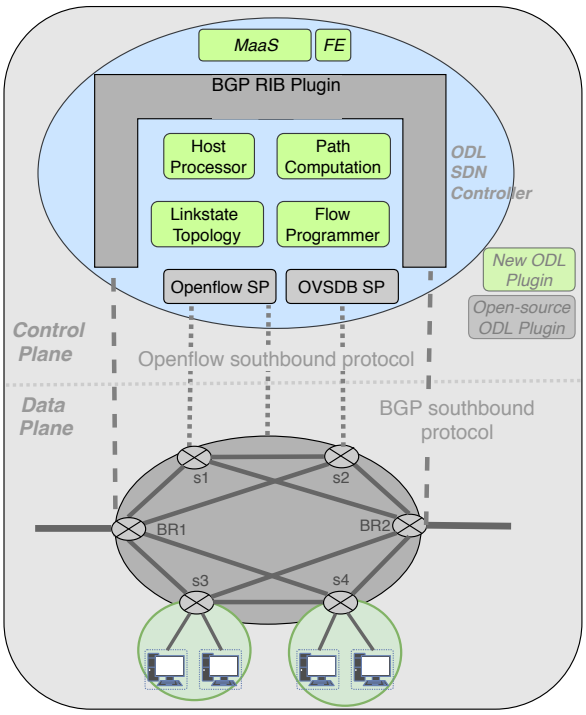


Fig. 4.10 Custom plugins developed for the Opendaylight SDN control-plane of our Testbed (green).

A. Openflow and OVSDB SP

Generally speaking, *southbound plugins* provide APIs to talk with remote data-plane devices for tasks that could be to change their forwarding table, or access their link-state. Southbound Plugins are specific to the *southbound protocols*, a term used for control protocols which support interaction between control-plane (SDN controller) and data-plane (network-devices) which is required since they are physically separated in SDN environment (Fig. 4.10). SPs therefore abstract the complexity of southbound protocols for other applications that wish to interact with data-plane. SPs can also be built for traditional control protocols like BGP and RSVP, apart from the ones relating more to the SDN era like Openflow and Netconf.

Multiple southbound plugins can be housed in the SDN controller accordingly to talk with different type of devices in data-plane. Since we have only two types of devices in our testbed, OpenvSwitches (ovs) and Quagga, we only thus require southbound plugins specific to the protocols supported by them devices- openflow and ovsdb SP for ovs as briefed below, and BGP for Quagga which is explained later.

- Openflow SP (Fig. 4.10) exposes complete functionality of Openflow protocol supported by the network device which is generally as per the protocol standard specification. This plugin is available and maintained open-source within the ODL community. Majorly for our purposes, it maintains a topology providing information on all the data-plane Openflow devices (ovs) and links between them, which other applications can query through MD-SAL Java or Northbound REST APIs. It also provides similar APIs to program the forwarding table of Openflow devices in the topology.

- While ovs network-devices support Openflow as the control protocol, for a management protocol, they support *ovsdb* interface. OVSDB SP similarly developed within the ODL community provides services to manage Openflow devices through maintaining a topology which provides access to information like MAC and IP addresses on the data-plane switches/links, details of queues on each of the link, and also the ability to create them.

Southbound Plugins bridge the gap between device vendors and network application developers as vendors themselves can supply SPs providing unified and high-levels APIs for applications developers to configure and access even their proprietary network devices. They are an important component of ODL or any other SDN platform and forms the lower most sub-layer of logical network control abstraction layer in controller.

B. Linkstate Topology Plugin

Linkstate topology plugin (Fig. 4.10) is developed to maintain a topology that represents complete network state and view, contrary to Openflow SP topology as discussed previously only providing information on network graph (nodes and links). To make a complete network topology, Linkstate Topology plugin starts off with the base network graph as provided by Openflow SP (Fig. 4.11)-

i. augments each of the link in there with information on queues (QoS) provided by ovsdb SP (Fig. 4.12),
ii. augments each of the link with real-time counters such as total bytes and packets received or transmitted as requested from Openflow SP's statistics service and also derives more meaningful information from those counters such as utilized and available bandwidth (Fig. 4.12),
iii. and also adds information on end-hosts to the original Openflow SP topology model as shown in Figure. 4.11.

```
{
   "topology": [
      {
         "topology-id": "example-linkstate-topology",
         "link": [...],
         "node": [...],
         "openflow-linkstate:host": [...]
      }
   ]
}
```

Fig. 4.11 Reference linkstate topology as produced by the plugin of the same name. Lists are collapsed to reflect on the outer structure of the topology.

The Linkstate Topology plugin does not generate any new service or information in our controller platform, but merely collates information from various other native ODL plugins and processes it in some cases like with calculating available link bandwidth. This plugin is developed so as to maintain a one-stop shop to all the network state and view by way of wrapping around SPs, required by and useful for other controller applications/plugins as will be reflected in the overview of other applications next.

C. Host Processor

Host-processor plugin (Fig. 4.10) has been custom developed with the purpose of discovering and maintaining information on all (testbed) network hosts, be it local or external hosts. These end-hosts are in the form of Mininet virtual Linux hosts (Sec. 4.2.1.A).

```
{
"link": [
   {
      "link-id": "openflow:10:1",
      "source": {
         "source-node": "openflow:10",
         "source-tp": "openflow:10:1"
      },
      "destination": {
         "dest-node": "openflow:1",
         "dest-tp": "openflow:1:3"
      },
      "openflow-linkstate:link-name": "openflow:10>openflow:1&&b10>a1",

      "openflow-linkstate:qos": {
         "qos-id": "fcb71395-989c-445c-b9bd-2f458a6aef3c",
         "qos-config": {
            "max-rate": "20.000"
         },
         "queues-ofls": [
            {
               "queue-id": "queue://31a886d6-d941-44c9-b9a9-72290086d9ce",
               "queue-number": 10,
               "queues-config": {
                  "reserved": "0",
                  "min-rate": "10.000",
                  "max-rate": "10.000"
               },
            }
            ...
         ]
      },

      "openflow-linkstate-statistics:te-metrics": {
         "max-bandwidth": "20.000",
         "utilized-bandwidth": "0.000"
      },
      "openflow-linkstate-statistics:flow-capable-node-connector-statistics": {
         "transmit-drops": 0,
         "receive-drops": 0,
         "packets": {
            "received": 84,
            "transmitted": 84
         },
         "bytes": {
            "received": 7108,
            "transmitted": 7260
         },
         ...
      }
   },
   ...
]
}
```


Fig. 4.12 Reference linkstate topology as produced by the plugin of the same name. Notice the queues (qos) and statistics information augmented to the link.

- The local or intra-domain hosts are tracked by subscribing to any changes in linkstate-topology plugin's data-tree specifically the *openflow-linkstate:host* sub-tree as shown

in Figure. 4.11. This type of service is provided by MD-SAL trough data-change events as explained in Appendix B, and Fig. 4.13 shows code snippets for subscribing to these changes in hosts, and the delivery of change events to subscribers.

- The external (inter-domain) hosts are similarly tracked by subscribing to changes in BGP RIB maintained by the plugin of the same name (Fig. 4.10).

The purpose of discovering and maintaining information on all the hosts in this plugin is to thereafter provision reachability (routing) to them from all the other nodes/switches in local (testbed) domain network, which is done through utilizing services of other new ODL plugins as below.

```
import org.opendaylight.controller.md.sal.binding.api.DataBroker;

public class InternalHostListener implements DataTreeChangeListener<Host> {

   public InternalHostListener() {

      //identifier for the data-tree node.
      InstanceIdentifier<Host> treeIndentifier =  InstanceIdentifier.builder(NetworkTopology.class)
                                 .child(Topology.class, "example-linkstate-topology")
                                 .child(Host.class)
                                 .build();

      //register or subscribe to any changes in the node specified by the identifier delivered to
      //this class.
      DataBroker.registerDataTreeChangeListener(LogicalDatastoreType.OPERATIONAL, treeIndentifier,
                        this);
   }

   //function onto which the change events are delivered, defined in the implemented class
   //DataTreeChangeListener above.
   @Override
   public void onDataTreeChanged(Collection<DataTreeModification<Metadata>> changes) {
      for(DataTreeModification<Metadata> change: changes){
            ...
      }
   }
}
```

Fig. 4.13 Code snippet to subscribe for changes in *host* data-tree of *linkstate-topology*, and its delivery.

D. Path Computation

Path-computation plugin is newly developed to provide services for path computation between nodes or hosts in the local domain network. It retrieves network topology information

from *linkstate* plugin to run the path computation algorithm on. Majorly, it provides following 2 services-

- It provides a shortest path computation service utilized by the *host-processor* plugin to find a route from all the nodes in the network to the node attached to a discovered host, or a border-node in case of an inter-domain host for the purpose of provisoning best-effort IP routing in the domain network.

- It also provides a CSPF algorithm service that is utilized by the *MaaS* plugin as we saw previously, to compute paths for the LSPs that also have the QoS (bandwidth) resources desired by the requesting customer. CSPF service computes multiple paths if the entire resource-request cannot be satisfied with one, and also still returns partial paths if the entire resource-request cannot be satisfied through any number of paths to help in service re-negotiation.

This plugin is analogous to PCE in our ODL SDN controller. Like *host-processor* plugin previously, it is also vastly dependent on the wide range of information provided by *Linkstate-Topology* plugin.

E. Flow Programmer

Flow-programmer plugin is developed to simplify the process of installing explicit paths computed by the path-computation plugin onto the openflow data-plane, be it a best-effort IP route or a Label-switched Path. It provides services that take it an explicit hop-by-hop path from path-computation plugin which can be at the request of host-processor plugin or MaaS plugin, converts that path into Openflow rules (match-action pairs), and finally uses the services of Openflow SP to install those rules onto the devices which modifies their forwarding-tables and hence installs that path on data-plane.

The value of flow-programmer plugin is that an explicit IP route or LSP maps to many dependent Openflow match-action pairs on forwarding-plane, and this plugin maintains dependencies between them pairs which simplifies the process of updating or deleting the path when such a need comes in. It thus provides a further abstraction layer of an Openflow network in our controller over what is provided by Openflow SP.

The functions of flow-programmer and other plugins explained above are better brought out in the next section through an example scenario when they collectively perform intra-domain and also inter-domain routing.

F. BGP RIB Plugin

While Openflow SP provides the ability to talk with Openflow capable *ovs* devices, BGP RIB plugin provides the ability to talk with BGP capable *ovs* nodes (Quagga) in our Testbed. This plugin provides complete BGP Speaker functionality similar to vendor BGP routers but also with added flexibility and programmability as a result of SDN control. Available open-source from within the Opendaylight community and known as *odl-bgpcep-bgp* project, it is an important constituent of our desired ODL control-plane as it facilitates inter-domain routing and also *Federation Engine*, and is detailed in Appendix A.

4.3.4 Important Controller Functions

The plugins/applications explained above, which constitute our Testbed SDN control-plane, apart from providing specific service APIs for other controller applications like MaaS, also collectively perform two important network control functions, intra-domain and inter-domain routing. These two functions were also the requirements from our Testbed as was mentioned earlier in Section 4.1.A. The orchestration of these is explained next through example scenarios.

A. Intra-Domain Routing

Fig. 4.14 shows the scenario as well as the service-chain that works to automate intra-domain routing onto our Testbed virtual network. The example scenario relates to provisioning of routing for Host H1 from one of the network nodes (s3). The below sequence which is listed coherent to Fig. 4.14 scenario briefs this function.

1. *Host Discovery:* Foremost, end-hosts have to be made discoverable by the SDN controller. Provisions have to be made therefore on the data-plane devices so that the hosts could be tracked by the SDN controller. For this purpose, on every switch a static low priority forwarding entry is made, specifically on s2 in our Figure. 4.14 scenario that forwards all the incoming unrecognised IP/ARP packet towards the controller. Therefore, when host H1 powers up and generates its first packet, the switch s2 would not be able to recognize source ethernet or network address, and will forward the packet to the controller via the Openflow southbound interface (1, Fig. 4.14). Openflow SP thus tracks hosts in an openflow network through listening to Packet-In events, and thereafter adds them into its maintained network graph (topology).

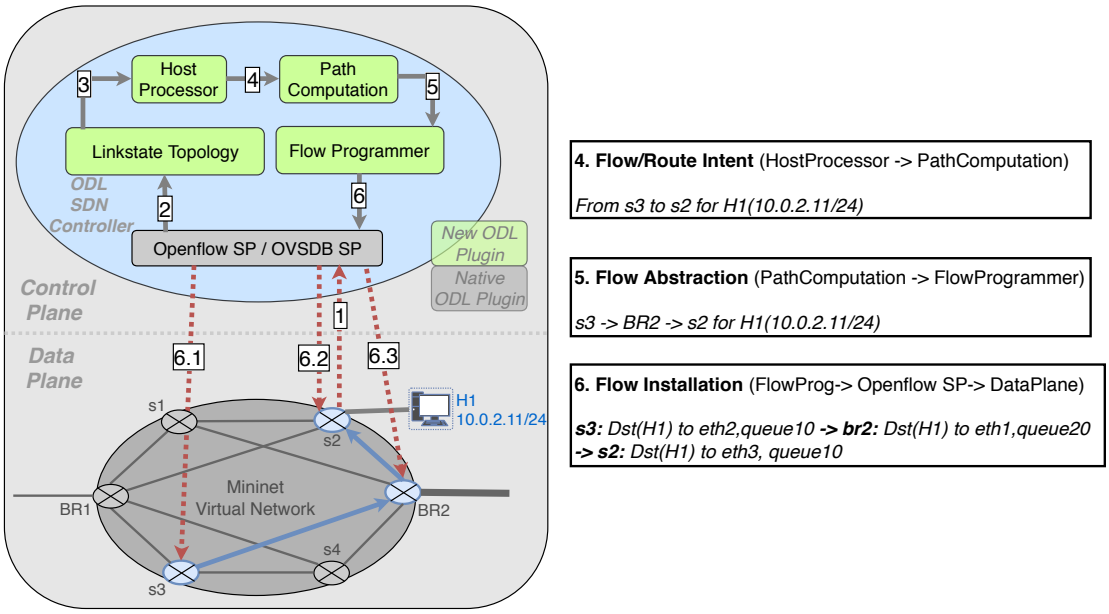


Fig. 4.14 Service-chain involving controller plugins that performs intra-domain routing function for host H1 from node s3.

2. Linkstate Topology registers for any change in Openflow SP's network graph, which in this scenario is the addition of host H1, and updates its own topology model with this change (2, Fig. 4.14).

3. Host-Processor plugin as we saw previously maintains information on all hosts, local or external, and therefore listens to this addition of a new host in *linkstate topology* (3, Fig. 4.14).

4. *Flow Intent:* Host-Processor plugin upon discovering a host endeavours to configure reachability (routing) to it from all the nodes in network. For the purpose of this example, we specifically only concentrate on routing from s3. It thus uses path-computation APIs to convey an *intent* for best-effort routing of host H1 from s3 (4, Fig. 4.14).

5. *Flow Abstraction:* To satisfy the received intent for a flow/route, path-computation plugin utilizes network topology data to compute an explicit hop-by-hop path between the intended endpoints, which in this scenario is *s3 -> BR2 -> s2* (5, Fig. 4.14).

6. *Flow Installation:* The intent is fully satisfied when flow-programmer plugin breaks down the abstract hop-by-hop path into Openflow match-action rule objects that are then installed on the data-plane using the Openflow SP APIs (6, Fig. 4.14).

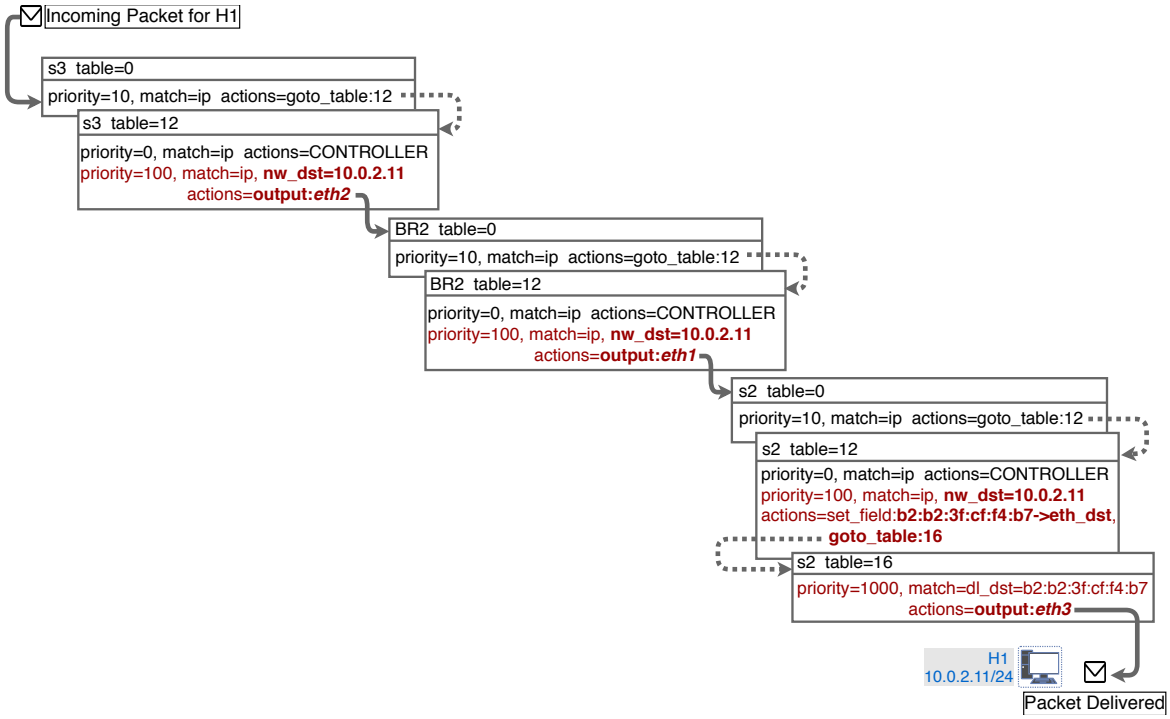


Fig. 4.15 Openflow forwarding tables for the nodes involved in the IP route from s3 to s2 for Fig. 4.14 scenario.

Fig. 4.15 shows the resultantly modified forwarding tables after *flow-programmer* plugin via *Openflow SP* uploaded openflow match-action rule objects (highlighted in red) into them to enable routing for host H1 from node s3. Similar to s3, intra-domain routing is configured on the data-plane from all the switch nodes or optionally a subset of them as desired by the *host-processor* application. In this project, we function with Openflow 1.3 which is also supported by our network device –OpenVSwitch. Openflow 1.3 introduced multiple flow tables (MFT) compared to its predecessor 1.0 only supporting a single forwarding table (ONF, 2015). MFT provides greater flexibility and organization of flow rules as we incorporate and shown in Fig. 4.15.

Resultantly through this function, a layer of shortest path IP routing is laid onto the data-plane for every discovered host. This function also performs the reverse process in which forwarding entries or routes are deleted whenever a host is lost. Next, provisions have to be made, aided by BGP, to extend this intra-domain routing function to inter-domain one as there is reachability required also between the hosts residing even in different domains.

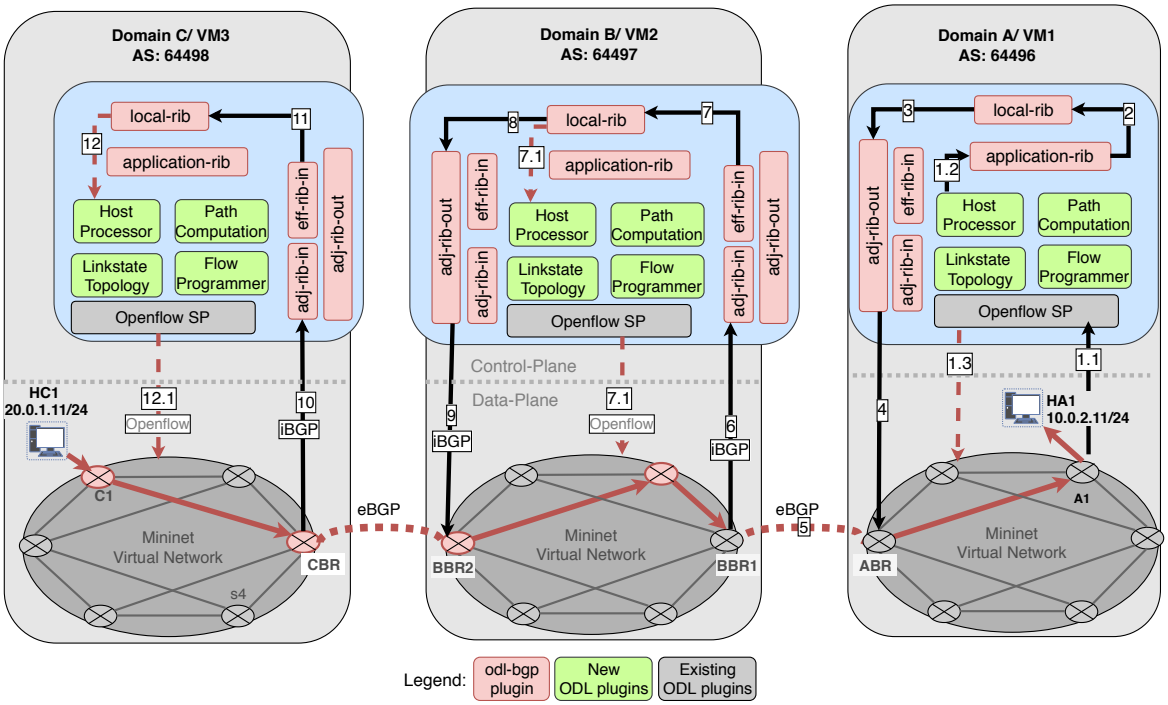


Fig. 4.16 Inter-domain routing function and scenario for reachability from HC1 to HA1.

B. Inter-Domain Routing

Inter-domain routing in our SDN Testbed can be considered as an extension of intra-domain routing function explained previously, with additional coordination provided by BGP. If the source-domain controller configures routing to its local IP host from all its Border Routers (*ABR, HA1,* Fig. 4.16); advertises host's prefix inter-domain via eBGP process on those BRs (4-5, Fig. 4.16); and subsequent domains that receive this route advertisement configure routing for that specific IP prefix from their internal network nodes to their own Border Router that can reach into prefix's source-domain directly or indirectly (*C1, CBR*, Fig. 4.16), best-effort inter-domain e2e routing can be simply provisioned in this manner. Employing BGP is thus integral for this process as it is also in Internet, and is explained in detail in Appendix X.

Fig. 4.16 and following overview illustrate the just briefed inter-domain routing function through an example scenario this time, wherein end-to-end routing is to be configured for host HA1 from node C1.

- Referencing Fig. 4.16, once the host HA1 is discovered by the Domain A controller, intra-domain routing is provisioned to this host on the data-plane from all the nodes in network including border router ABR which is specifically important in this scenario.

Host-Processor plugin orchestrates this routing function using the services of *path-computation* and *flow-programmer* plugin. Additionally, it also enters HB1's prefix to BGP *application-rib* (1.2, Fig. 4.16).

- From Domain-A application-rib, the route for HB1 prefix spills into local-rib and from there it is shared with all the iBGP peers of controller by adding the prefix/route to their corresponding adjacent-rib-out (2-4, Fig. 4.16). iBGP peers further advertise this route to their eBGP peers (5, ABR) and in this way a local prefix (HA1) reaches a neighbouring domain (BBR1).

- From BBR1 eBGP router, external routes (HA1) are shared internally with its iBGP peers, domain controller B in this scenario (6, 7). Host-Processor thereafter listens to this new addition of an external prefix in local-rib, and configures intra-domain routing for this prefix from all the nodes (specifically BBR2) to next-hop i.e BBR1 (7.1).

- Domain controller B also similarly shares the route for external prefix HA1 with its iBGP peers (BBR2 in this case) and in this way the route is propagated further inter-domain (8,9). Once it reaches domain-controller C (10, 11), it also performs the similar process of laying out intra-domain routing from all the nodes to exit next-hop, specifically from C1 to CBR to effect inter-domain routing from HC1 to HA1.

In the next chapter, features of MaaS detailed in previous chapter are experimentally demonstrated on the WaterFed testbed detailed in this chapter.

Chapter 5

Experimental Demonstration

5.1 Introduction

In this chapter, the major features of our proposed MaaS framework that are - end-to-end real-time QoS, service re-negotiation mechanism and customer delegated mapping which were detailed earlier in chapter III are now demonstrated through simulated experiments or scenarios. In a nutshell, in the WaterFed Testbed of chapter IV, IP flows are started between inter-domain endpoints while the flow QoS (bandwidth) is recorded continuously. Improvement or patterns in the recorded QoS metrics when MaaS comes into action relative to the entire time duration quantitatively demonstrates the proposed features. Before presenting the results and scenarios, the experimental setup used for this purpose is briefed.

5.2 Setup

A. Network Topology

The network topology and scenario of Figure 5.1 which was also considered earlier to explain the herein demonstrated inter-domain MaaS features is re-used to perform the experiments as well. The considered topology consists of 4 virtual network domains, each of them linked and BGP peered with 2 other domains. This represents a fairly simplistic as well as sufficiently complex multi-domain topology to put to test our MaaS framework or SDN-MPLS plane performing guaranteed QoS routing, and also in the process SDN-IP-BGP control-plane which prerforms best-effort shortest-path routing.

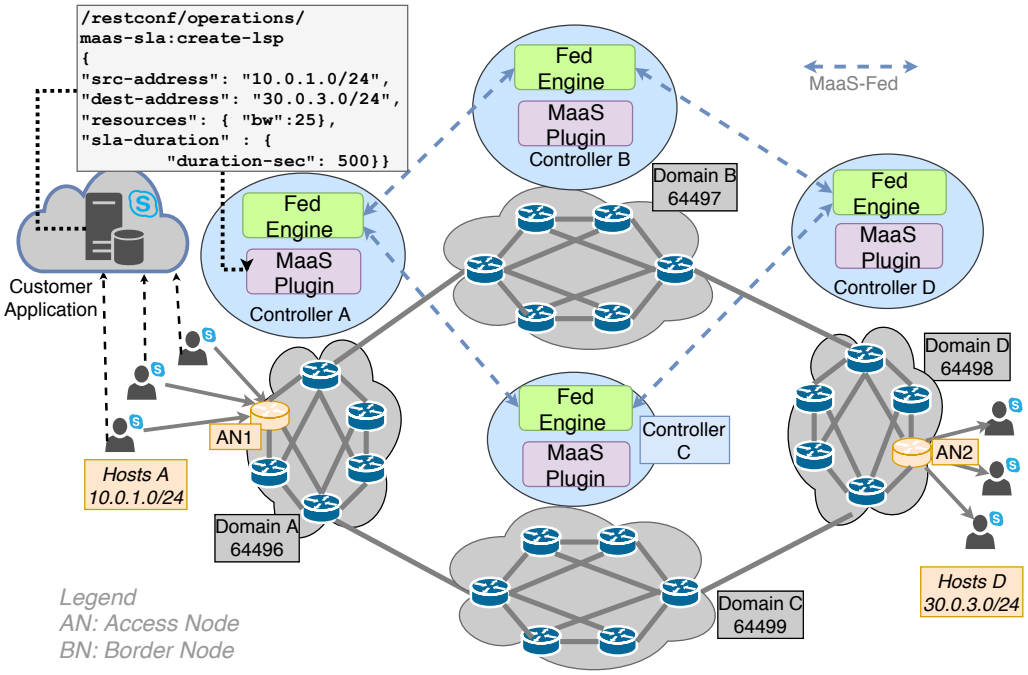


Fig. 5.1 Network topology and customer application illustration, considered to demonstrate features of MaaS through simulated experiments.

B. Customer Application

To be able to truly demonstrate our application-centric framework, a customer application is thus required that would request QoS between inter-domain endpoints. We simulate such an application as shown in Figure 5.1 through a simple Python script, that firstly starts flows between its endpoints (*Hosts A,D*) and then uses *create-lsp* API of its source-domain A to request guaranteed QoS for them flows.

C. Customer endpoints and flow-agents (iPerf)

The endpoints of above described customer application are in the form of virtual Linux hosts provided by *Mininet*, which is also true in general for any endpoint in our Testbed. These Linux hosts by default do not have any program that can generate IP flows to a destination. Therfore *iPerf* which is a popular tool to produce custom TCP or UDP IP flows is used for this purpose. It works in a client-server model wherein the flows travel from client to server by default or even vice-versa. Thus, *iPerf server* and *client* serve as the flow-agents of our simulated customer application as shown in Figure 5.2.

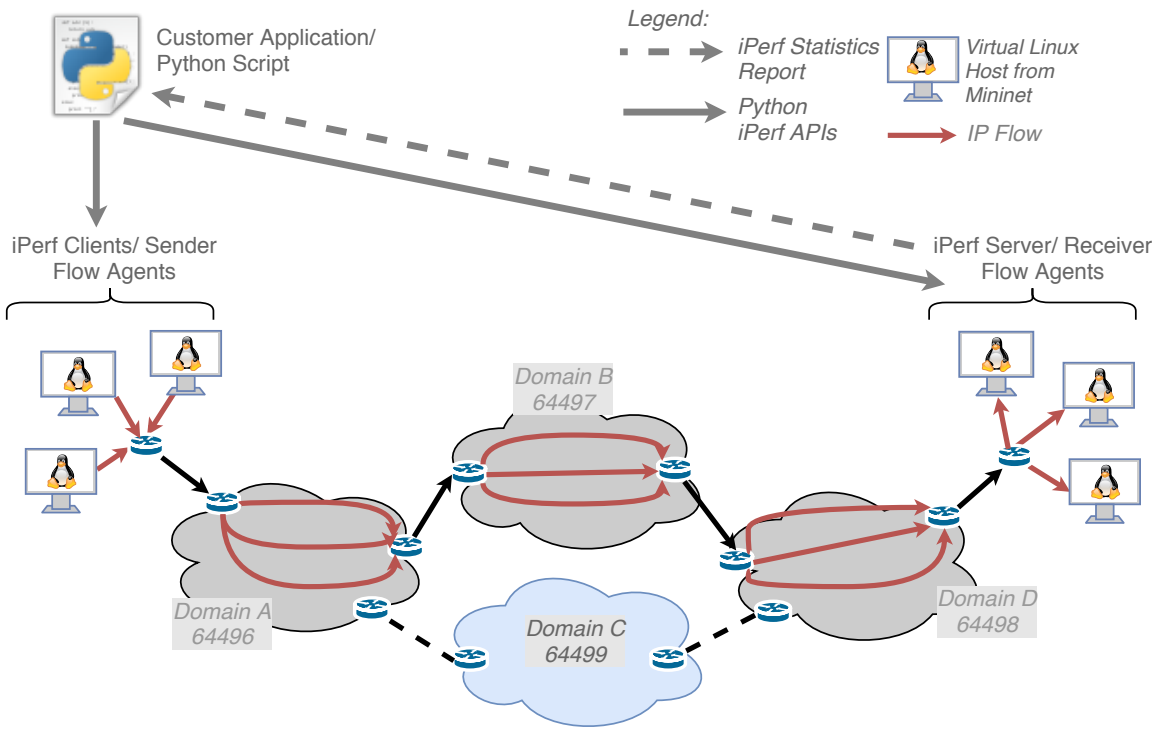


Fig. 5.2 Customer application, end-hosts and flow-agents used in the experimental setup for demonstrating MaaS features.

The ability to produce flows is also essential from the perspective that through measuring its end-to-end bandwidth, the performance or QoS gains of MaaS LSP provisioning can be easily demonstrated. In this respect, iPerf is best suited as one of its major feature is to generate detailed statistic reports (bandwidth, jitter, packet-loss) for the IP session both at sender and receiver (Fig. 5.2). Moreover, there are Python APIs available for iPerf using which our simulated customer application can easily start iPerf client or server program in different virtual hosts and thus IP flows between them, as well as fetch results (Fig. 5.2).

5.3 Dynamic end-to-end QoS

Firstly, real-time QoS provisioning done by MaaS framework is experimentally demonstrated with the results shown in Figure 5.3. The different phases in that visualization are analysed below.

- *shortest-path routing (0 to 25 secs):*
  3 IP flows are started by our (simulated) customer application between 3 pair of inter-domain endpoints previously shown in our considered multi-domain network topology

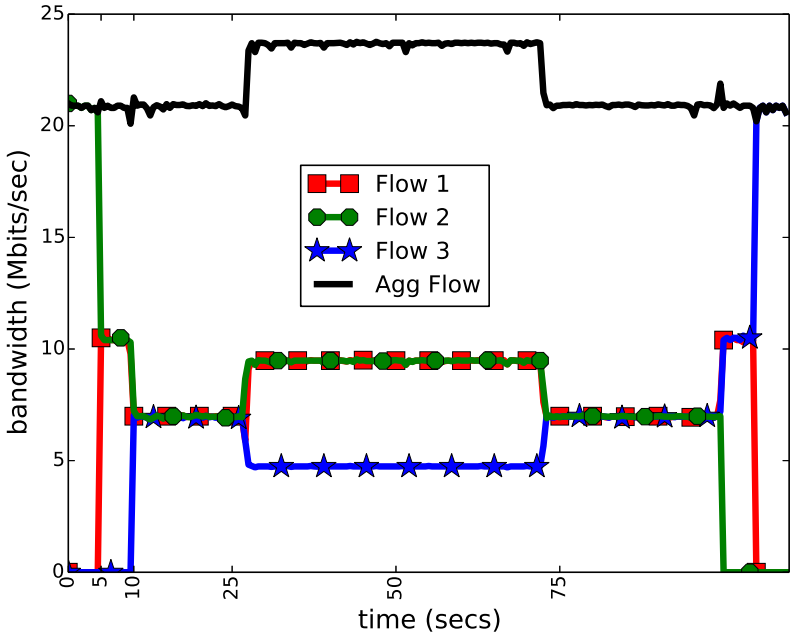


Fig. 5.3 End-to-end bandwidth for 3 iPerf flows, when they are routed across the shortest-path and when they are tunnelled through the QoS LSP segments.

and shown again in Figure 5.4 (HAs, HBs). These source and destination hosts of the flows are also represented in first 3 columns of Table 5.1.

These 3 flows are not started simultaneously but at an interval of 5 seconds each. Therefore it can be seen in the above Figure 5.3 graph that the maximum link capacity of 20 Mbps configured in our *mininet* testbed is getting split between them 3 flows as they start. *Flow 1* occupies entire 20 Mbps untill when *flow 2* starts at t=5s, and when *flow 3* starts at t=10s entire link bandwidth gets equally allocated to 3 flows, as can be seen in the graph. In this phase, all 3 flows are routed across the same shortest inter-domain path without any specific QoS or TE allocation, the configuration of which is a function of our testbed SDN control-plane as was explained earlier in section 4.3.4.

- *MPLS QoS routing (25 to 75 secs):*
  At about t=25 seconds, customer application requests a 25 Mbps QoS path for its ongoing flows for a duration of 50 seconds. Upon receiving the request as shown in Figure 5.1, different domain MaaS plugins collaboratively provision end-to-end LSP segments. These segments are shown illustrated in Fig. 5.4, while the protocol message sequence between different MaaS plugins is illustrated in Fig. 5.5 and real wireshark

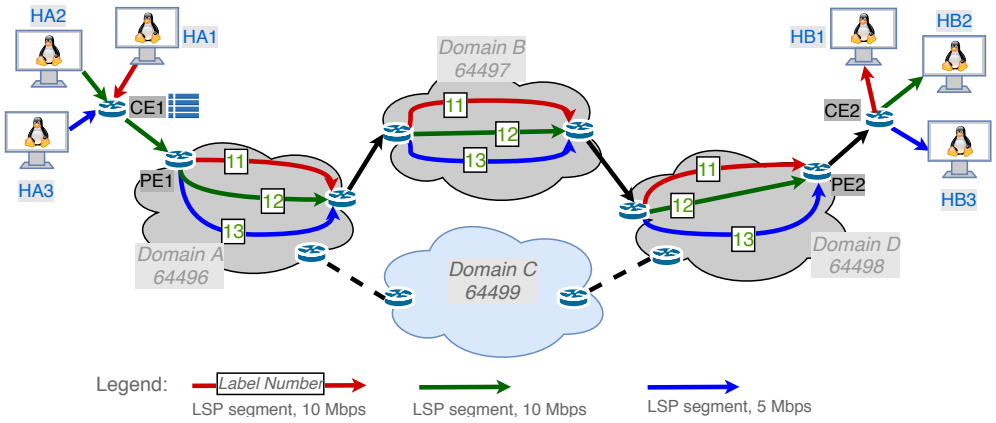


Fig. 5.4 Provisioned LSP segments and end-hosts illustrated on the reference network-topology.

capture of protocol messages shown in Fig. 5.6. The details of these LSP segments are also passed onto the customer application as a *create-lsp response*, shown as it is for this experiment scenario in Figure 5.7. Thereafter, customer (simulated through python script) performs mapping between them flows and Labels in its authority network (*CE1*, Fig. 5.4) with the allocation shown in Table 5.1, flows thus start arriving in the MaaS domains already encapsulated with desired Label headers, and increase in aggregate QoS to 25 Mbps can easily be observed in the graph at about t=25s.

As can also be seen from 25s to 75s, different flows experience different QoS as the LSP segments they travel across have different bandwidth shown in Figure 5.7. Down at the data-plane, bandwidth for the LSP segments is reserved among the *ovs* devices through assigning a *queue* to the Label/LSP, and therefore flow bandwidth traversing the tunnel is limited by the maximum bandwidth allowed on the queue. Our simulated customer application undertakes a simple greedy approach for allocating different LSP segments in each domain to all the candidate flows with the result shown in Tables 5.1. The benefits of this customer delegated mapping are further demonstrated in a separate next section.

IP Flow	Source	Destination	Allocated Label Stack	Resultant BW
Flow 1	HA1 - 10.0.1.11	HB1 - 30.0.3.11	11,11,11	10
Flow 2	HA2 - 10.0.1.12	HB2 - 30.0.3.12	12,12,12	10
Flow 3	HA3 - 10.0.1.13	HB3 - 30.0.3.13	13,13,13	5

Table 5.1 Mapping between IP flows and Label segments of Figure 3.11

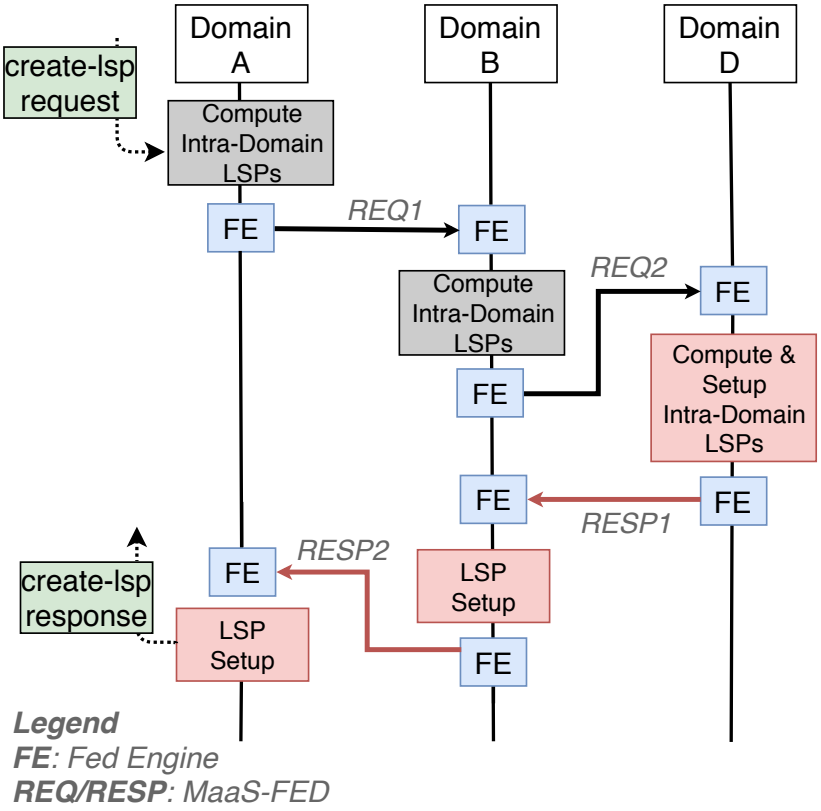


Fig. 5.5 MaaS-Fed protocol message sequence for end-to-end LSP setup related to Fig. 5.1 ideal scenario.

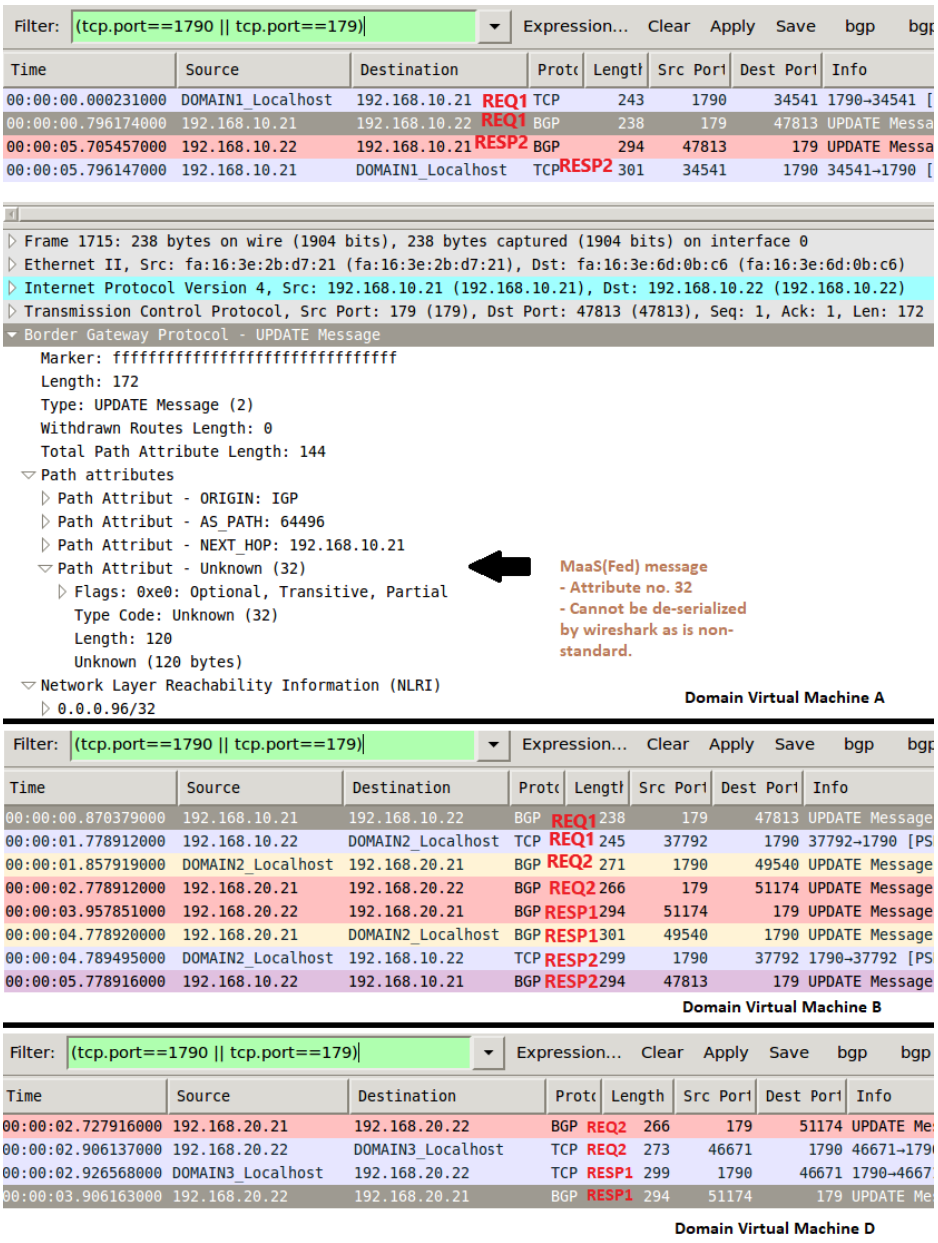


Fig. 5.6 Wireshark traffic capture from different domain virtual machines showing the messages exchanged by involved MaaS plugins for inter-domain LSP setup related to this experimental scenario and above Fig. 5.5. Following information can be validated from this. (Packet data only shown once in VM A as MaaS message cannot be decoded.)

- Inter-domain LSP setup / MaaS-Fed protocol transported as an attribute of BGP update message (*VM A, attribute 32*). Captured packets are therefore decoded as BGP (*proto*) by wireshark, except for some iBGP messages that are sourced/destined from/for non-standard BGP port 1790. Notice the wireshark packet filter, border router (Quagga) BGP speaker is running on standard port 179 while the controller BGP process is running on port 1790, hence the filter.
- Although capture is from different VMs they are time-shifted by a common offset, and thus from the *time* column, chronological order of the messages as in Fig. 5.5 sequence diagram can be validated. Each message is seen twice in a VM as one is an eBGP advertisement and the other is a iBGP advertisement to localhost controller.

```
{
        "output": {
                "sla-status": {
                        "aggregate-status": "success",
                        "domain-result": [
                        {
                                "result": "commissioned",
                                "domain-id": 64496,
                                "sla-flow-detail": [
                                {
                                        "segment-id": 11,
                                        "bw": "10"
                                },
                                {
                                        "segment-id": 12,
                                        "bw": "10"
                                },
                                {
                                        "segment-id": 13,
                                        "bw": "5"
                                }]
                        },
                        {
                                "result": "commissioned",
                                "domain-id": 64497,
                                "sla-flow-detail": [
                                {
                                        "segment-id": 11,
                                        "bw": "10"
                                },
                                {
                                        "segment-id": 12,
                                        "bw": "10"
                                },
                                {
                                        "segment-id": 13,
                                        "bw": "5"
                                }]
                        },
                        {
                                "result": "commissioned",
                                "domain-id": 64498,
                                "sla-flow-detail": [
                                {
                                        "segment-id": 11,
                                        "bw": "10"
                                },
                                {
                                        "segment-id": 12,
                                        "bw": "10"
                                },
                                {
                                        "segment-id": 13,
                                        "bw": "5"
                                }]
                        }
                        ]
                }
        }
}
```


Fig. 5.7 Response to create-lsp request containing Labels segments and their bandwidth information for each of the domain.

5.4 Customer-delegated mapping

In this section, customer delegated mapping of IP flows to LSPs is demonstrated, a feature which was explained previously in section 3.6.4. Through this mechanism, MaaS domains share information on provisioned LSPs as shown in Figure 5.7 with customers, empowering them to perform mapping instead of domains themselves doing it. The benefits are in efficient Traffic Engineering based mapping process and scalability which are demonstrated next.

To demonstrate this feature, no new LSPs need to be created as to enable it from the MaaS perspective, they just have to share the information on Labels and bandwidth with the customer, which they do shown in Figure 5.7. Therefore we take forward the previously provisioned LSPs, and experiment only instead with the mapping of flows to Labels done at customer end (by simulated Python script at CE1 in Figure 5.4) in order to demonstrate efficient TE and scalability.

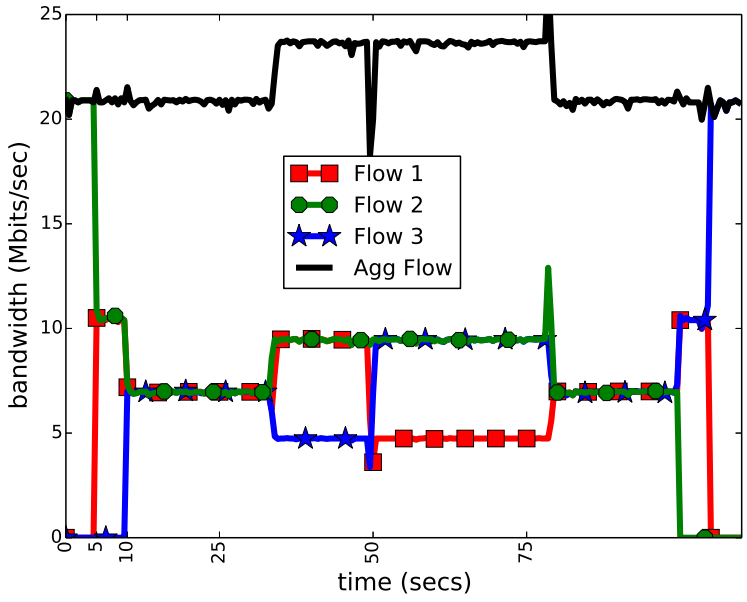


Fig. 5.8 Demonstration of the efficient TE feature of customer-delegated mapping.

- *Efficient Traffic Engineering:* Figure 5.8 brings out the TE advantages of customer-delegated mapping facilitated by MaaS. The first example of TE is evident from *30 to 80s* wherein individual flows are allocated different QoS. This is made possible by customer allocating varied bandwidth tunnels to different flows on the basis of their priority, a knowledge only customer best possesses. MaaS domains in doing such a process would rather load-balance all flows between the LSPs which is not favorable

of TE, or would have to expose APIs for customers to provide them with priority/TE information. But even in that latter case it won't be as dynamic as customer-delegated approach, wherein whenever the priority between flows changes, end-to-end bandwidth can also be instantly altered by simply swapping stack of Labels without the need of network getting involved (50s, Fig. 5.8).

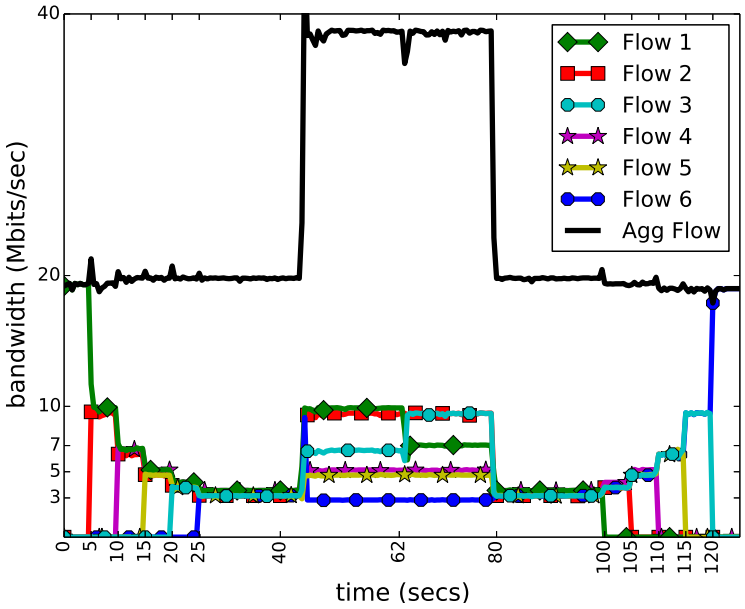


Fig. 5.9 Demonstration of scalability property of MaaS's customer-delegated mapping, wherein aggregate bandwidth (40 Mbps) can be bought for many competing (6) flows.

- *Scalability:* Another advantage of customer-delegated mapping is that it enables coarse-grained scalable operations. Customers can buy aggregate bandwidth, and thereafter finely allocate part of that bandwidth to their many competing flows by simply allocating them (stack of) Labels or LSP segments, mitigating the need for many smaller (*create-lsp*) requests for each or group of flows.

  To experimentally demonstrate that, we upgraded the previous scenario and instead of 25 Mbps, 40 Mbps is now requested by the (simulated) customer application from MaaS domains. And instead of previously considered 3 competing flows, the aggregate 40 Mbps is part divided now among 6 flows as shown in Figure 5.9. As with the previous scenario, these flows are started at an interval of 5s each (0 to 25s), with they being routed in this phase through a common shortest-path and thus maximum link-bandwidth of 20 Mbps gets equally split between them 6 flows (25 to 45s). 40 Mbps

LSPs are provisioned at about t=45s, with the priority of *flow1 and flow3* swapping at t=62s to also again demonstrate the efficient TE process.

Scalability is demonstrated and argued in the respect that large aggregate bandwidth (from 25 Mbps to 40 Mbps) can be instead purchased through 1 request to network instead of 2 smaller requests, with customer-delegated tunnelling empowering efficient finer allocation of those flows to aggregate tunnels (LSPs).

5.5 Service re-negotiation and adaptivity

Section 3.6.3 explained the feature of adaptive service re-negotation by which MaaS domains and customer application make an effort to provision partial bandwidth LSPs for the time being when full bandwidth is not available in the end-to-end multi-domain network.

To demonstrate this feature, the same topology and scenario of Figure 5.1 is considered. However additionally to generate a resource-constraint condition so that service re-negotiation can be triggered, parallel bulky and local LSPs are created in both the intermediate domains B and C (*create-lsp-request 1/2*, Fig. 5.10) which drains majority of the domain resources. Therefore, when the scenario is triggered of requesting 25 Mbps LSPs between inter-domain endpoints (*create-lsp request3*, Fig. 5.10), both the intermediate domains B and C return

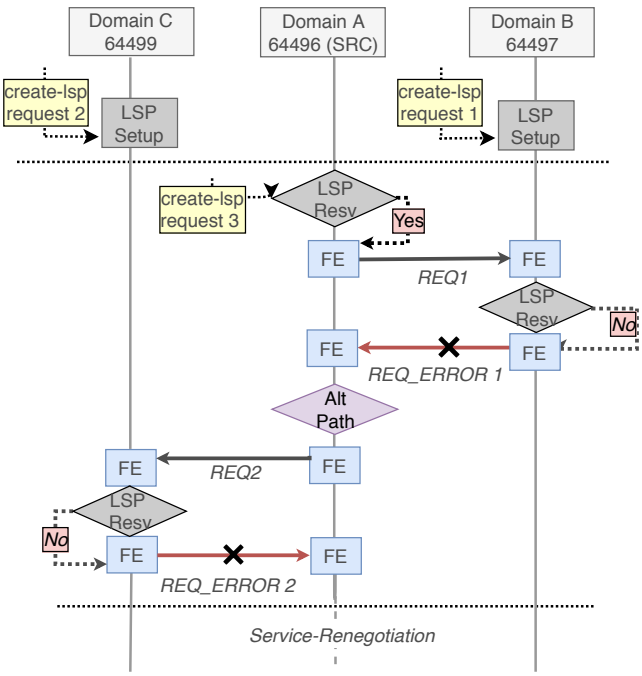


Fig. 5.10 Synthesized scenario to demonstrate service re-negotiation of MaaS.

back a REQ_ERROR message that contains a *cb_support* message mentioning the error details. The final REQ_ERROR2 message is shown in Figure 5.11. Notice the *cb_support* message in there from both the erroneous domains. This object is the enabler of service-renegotiation process as it contains information on partial resource availability and time in future when the full original resources will be available that is used by the source-domain and customer application to decide on whether to go ahead with partial provisioning or cancel the request for time being. Fig. 5.13 shows the wireshark packet capture for this experimental demonstration also related to Fig. 5.10 protocol sequence diagram illustration.


```
{
        "sla-status": {
                "aggregate-status": "success",
                "domain-result": [
                {
                        "result": "resource-unavailable",
                        "domain-id": 64496,
                        "sla-flow-detail": [
                        {
                                "segment-id": 17,
                                "bw": "10"
                        },
                        {
                                "segment-id": 18,
                                "bw": "10"
                        },
                        {
                                "segment-id": 19,
                                "bw": "5"
                        }
                        ]
                }
                ],
                "domain-error-log": [
                {
                        "result": "resource-unavailable",
                        "domain-id": 64497,
                        "cb-support": {
                                "partial-bw": "15",
                                "future-support": "14:00:14.646"
                        }
                },
                {
                        "result": "resource-unavailable",
                        "domain-id": 64499,
                        "cb-support": {
                                "partial-bw": "15",
                                "future-support": "14:00:15.721"
                        }
                }
                ]
        }
}
```


Fig. 5.11 *REQ_ERROR2* message of the Figure 5.10 scenario showing *cb_support* objects that assist in service renegotiation.

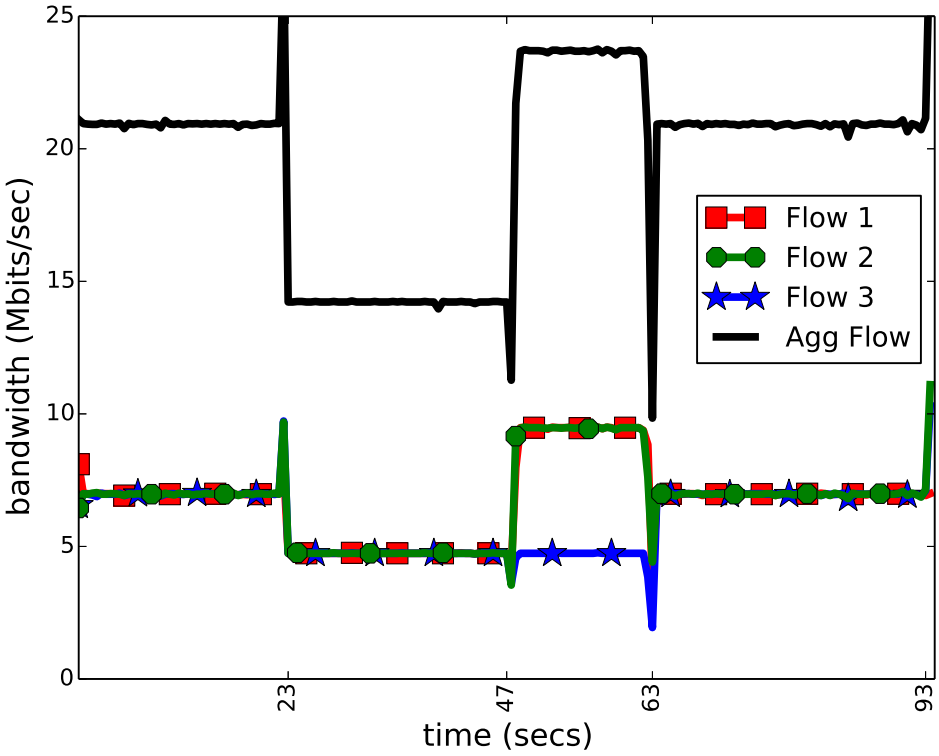


Fig. 5.12 Demonstration of service-renegotiation feature of MaaS.

Thereafter, source-domain A has to go ahead with partial resource provisioning due to unavailability of resources, and initiates a new end-to-end MaaS-Fed REQ/RESP process for the same. Notice the aggregate bandwidth coming down to 15 Mbps at 23s in Figure 5.12 as opposed to the originally requested 25 Mbps. Also when the full resources are available at t=47s, the LSPs are upgraded by all the domains as they were all aware of the time when the full resources were to be available in the erroneous domain as was conveyed through REQ_ERROR's cb_support object. Finally LSPs expire at 63s and flows come back to shortest best-effort route.

This adaptive service re-negotiation process built and envisioned in MaaS goes a long way in making the end-to-end QoS provisioning process dynamic as the scenario of intermittent resource outage is inevitable.

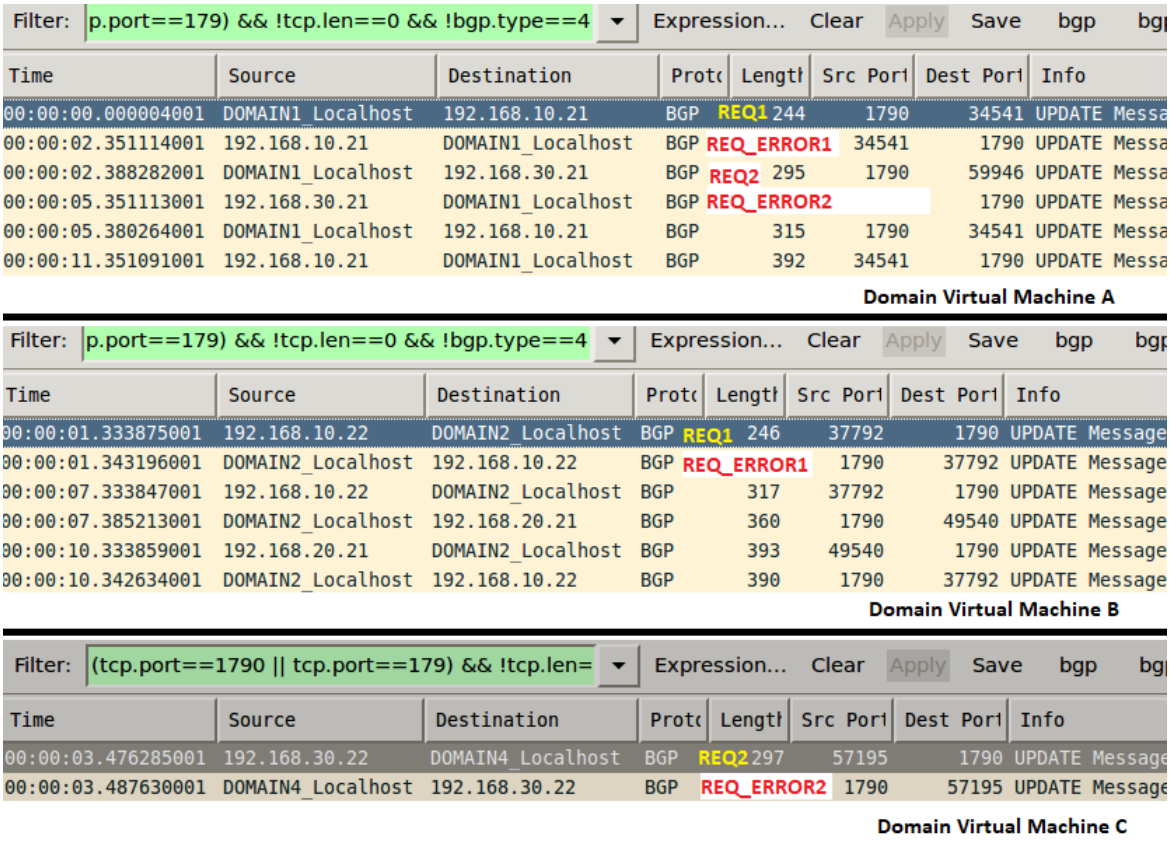


Fig. 5.13 Wireshark traffic capture from different domain VMs showing the messages exchanged by involved MaaS plugins for inter-domain LSP setup leading to crankback error-handling related to this experimental scenario and labelled similarly as above Fig. 5.10. Following validations can be made from this.

- Notice the erroneous REQ_ERORR messages (*1 and 2*) being returned to the source-domain VM A from two different eBGP routers (*10.21 and 30.21*), peered to eBGP processes on both two neighbouring resource-constraint domains B and C respectively.
- The rest of the unlabelled messages therefore continue the re-negotiated end-to-end LSP setup as in Fig. 5.5 via Domain VM B.
- Contrary to previous Fig. 5.6 capture, each message appears only once here as this capture is done only on the local interface reading only the iBGP messages.

Chapter 6

Conclusions, Limitations and Future Work

6.1 Conclusion

- *Dynamic application-centric inter-domain services utilizing SDN Federation:* SDN controllers and Federation of such controllers have emerged as the drivers of next generation Internet network architecture. In this respect, this driver has fueled the development of a framework (MaaS) to provision end-to-end network service dynamically over multiple domains incorporating application requirements.

- *Improved MPLS control-plane:* MPLS has been investigated to be the premier TE framework of Internet. Migrating distributed rigid control-plane of MPLS into SDN controller environment induces dynamicity into its operation helped by centralization and network control abstraction layer. Further collaboration between such per domain SDN-MPLS control-plane or MaaS entities can be made more secure, flexible, adaptive and scalable than legacy inter-domain MPLS operations reliant on inter-domain RSVP-TE.

- *Flexible inter-domain control-plane:* Inter-domain control-plane in an SDN Federation environment is orchestrated by much powerful SDN controllers than border-routers as in legacy networking. This provides grounds to deploy much more flexible inter-domain service-request protocols such as we do with MaaS-Fed building advanced error-handling and service-renegotiation into it.

- *End-to-end network aware applications, and scaling into customer premises:* To bring applications and network closer together is one of the major proposition of SDN control-plane. Improving upon that, even end-to-end multi-domain network state can also be made accessible to external applications as a result of SDN Federation. This property is demonstrated by our MaaS-Fed protocol which propagates LSP state from all involved domains to source-domain, and from there it is shared with the application customer, the utility of which has been proposed in customer-delegated mapping i.e. mapping between IP flows and LSPs in customer-premise equipments in a scalable approach.

- *Innovation through programmability (Generic BGP interface):* Orchestrating network service and protocols from the SDN control-plane (controller) enables their flexible use leveraging the programmability of controller. An example of this is evident from the proposed generic BGP interface wherein BGP is generalized to carry the messages of other inter-domain services than just its specific use-case of routing.

- *Lack and thus proposal of a native in-network inter-domain QoS provisioning framework:* From analysing the state-of-the-art in end-to-end service provisioning frameworks, there was an evident lack of such dynamic services built distributedly into every domain SDN control-plane or controller. Building them natively and closely interacting with rest of the control-plane, inter-domain QoS provisioning services can be made more agile, general-purpose and evolve at the pace of underlying control-plane as we propose with MaaS.

6.2 Learnings

Apart from the major benefit of exposure to research in QoS provisioning mechanism more specifically inter-domain, this project provided a good deal of network programming knowledge attained while developing MaaS and the underlying abstraction layer of SDN-IP-BGP control-plane, as well as hands-on in software architectures suitable for network controller development like MD-SAL and other aspects of working and developing in a production-grade SDN controller that is Opendaylight. Similarly, working towards developing the a virtual network testbed provided exposure to network design and configuring openflow forwarding or data-path. These will certainly be helpful moving further.

6.3 Limitations

This thesis proposes a fairly complete design of an inter-domain QoS framework in the form of MaaS, supported by the development of underlying SDN-IP-BGP plane as well built as the abstraction layer for MaaS. Proposing such an end-to-end solution of a critical Internet component and broadly an entire multi-domain network control-plane (SDN-IP-BGP-MPLS) is a bold statement, and therefore it is of utmost importance to bring out any major shortcoming and limitations.

- *Dynamicity and scalability analysis:* MaaS is a proposal of a dynamic MPLS and therefore QoS framework. However, there's no analysis, quantitative or other, done to reflect its dynamicity such as the latency and time taken to create end-to-end resource reserved LSPs as a function of number of domains, per-domain size and state. However, instead just the utilization of SDN controller and network control abstraction layer therein is proposed the natural enabler for dynamicity.

  A similar analysis on the scalability of the SDN-MPLS control-plane or MaaS framework is required to figure out the number of *create-lsp* requests per-second or minute that can be handled. And also similar scalability analysis of the data-plane, to figure out the number of Label forwarding states that can be stored. While improvement in this regard is proposed through customer-delegated mapping in their premises, it still provides a relatively small improvement.

  Broadly speaking, such dynamicity and scalability analysis of any SDN control-plane or service like MaaS presents a significant challenge and research direction in itself with the benefits in better benchmarking of the solutions.

- *Practical path-computation algorithm and latency sensitive traffic:* Initially focussing on the development of an end-to-end complete framework, the path-computation algorithm for LSP segments was kept simple to a constraint shortest path first (CSPF), where the constraint is the available unreserved bandwidth. However practically, real-time network load, state and management also need to be considered for an optimal path-computation process. Also helpful is the auto-recomputation of LSP segments in case of inevitable change in network state, as is performed by current legacy MPLS Path Computation Elements (PCE). The constraints in the CSPF also need to be extended with end-to-end one-way delay metric so as to cater and create LSPs for latency-sensitive application traffic.

- Understanding implications (security and scalability) of using BGP as a generic interface for communicating inter-domain MPLS service-request or MaaS-Fed messages.

6.4 Future Work

- A natural progression towards future research activity is answering for the major identified limitations listed above of current work - dynamicity and scalability considerations, analysis and enhancements; and optimization of path-computation algorithm.
- The work will be tried to be mapped with other ongoing network rethinking initiatives such as ONAP, P4 and even earlier ones such as Network Service Framework (NSF) standard discussed previously.
- The services of SDN control-plane plugins are not standardized contrary to legacy control protocols like IGP and RSVP. Though flexibility and interoperability are the advantages of SDN control, having open-standards on the data-model and APIs of important SDN control application or plugins like path-computation will help establish common best-practices.
- Different Opendaylight plugins developed during the project importantly *MaaS, Federation Engine and Flow-programmer* as well as the testbed network scripts are planned for open-sourcing after refinements to help other research initiatives in SDN Federation.

The thesis proposes a rather complete and also develops a multi-domain SDN-MPLS-IP-BGP control-plane. This to some extent is analogous to proposing an evolved design of Internet as changes have been made in how each domain operates (SDN control) and also how they interact (controller federation).

Concentrating on the heavy end-to-end design and development of such a framework scoped out working on any concrete academic contribution in the form of optimization or algorithms. However, the thesis acquainted with required state-of-the-art and network programming experience for future more concrete academic research work.

Bibliography

P. Aukia, M. Kodialam, P. V. N. Koppol, T. V. Lakshman, H. Sarin, and B. Suter. Rates: a server for mpls traffic engineering. *IEEE Network*, 14(2):34–41, Mar 2000. ISSN 0890-8044. doi: 10.1109/65.826370.

D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Requirements for traffic engineering over mpls. RFC 2702, RFC Editor, September 1999.

T. Bates, R. Chandra, D. Katz, and Y. Rekhter. Multiprotocol extensions for bgp-4. RFC 4760, RFC Editor, January 2007. URL http://www.rfc-editor.org/rfc/rfc4760.txt. http://www.rfc-editor.org/rfc/rfc4760.txt.

BGP-LS. North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP. RFC 7752, RFC Editor, March 2016.

M. Bjorklund. Yang - a data modeling language for the network configuration protocol (netconf). RFC 6020, RFC Editor, October 2010. URL http://www.rfc-editor.org/rfc/rfc6020.txt. http://www.rfc-editor.org/rfc/rfc6020.txt.

Huaimo Chen, Susan Hares, Yi Yang, Yanhe Fan, Mehmet Toy, Zhenqiang Li, and Lei Liu. Bgp for communications among controllers. Internet-Draft draft-chen-idr-com-cntlrs-02, IETF Secretariat, July 2017. URL http://www.ietf.org/internet-drafts/draft-chen-idr-com-cntlrs-02.txt. http://www.ietf.org/internet-drafts/draft-chen-idr-com-cntlrs-02.txt.

citrix:sd-wan. What is SD-WAN? https://www.citrix.com/glossary/sd-wan.html.

cloudfare:cdn. What is a CDN? https://www.cloudflare.com/learning/cdn/what-is-a-cdn/.

Saurav Das. *PAC. C: A unified control architecture for packet and circuit network convergence, Chapter 5*. PhD thesis, Citeseer, 2012.

DICE. InterDomain Controller Protocol, 2010. URL http://www.controlplane.net/.

Qiang Duan, Chonggang Wang, and Xiaolin Li. End-to-End Service Delivery with QoS Guarantee in Software Defined Networks. *CoRR*, abs/1504.04076, 2015. URL http://arxiv.org/abs/1504.04076.

Olivier Dugeon and Julien Meuric. A Backward Recursive PCE-initiated inter-domain LSP Setup. Internet-Draft draft-dugeon-brpc-stateful-00, IETF Secretariat, March 2017. URL http://www.ietf.org/internet-drafts/draft-dugeon-brpc-stateful-00.txt. http://www.ietf.org/internet-drafts/draft-dugeon-brpc-stateful-00.txt.

A. Elwalid, C. Jin, S. Low, and I. Widjaja. Mate: Mpls adaptive traffic engineering. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 3, pages 1300–1309 vol.3, 2001. doi: 10.1109/INFCOM.2001.916625.

A. Farrel, A. Satyanarayana, A. Iwata, N. Fujita, and G. Ash. Crankback signaling extensions for mpls and gmpls rsvp-te. RFC 4920, RFC Editor, July 2007.

E. Haleplidis, K. Pentikousis, S. Denazis, J. Hadi Salim, D. Meyer, and O. Koufopavlou. Software-defined networking (sdn): Layers and architecture terminology. RFC 7426, RFC Editor, January 2015. URL http://www.rfc-editor.org/rfc/rfc7426.txt. http://www.rfc-editor.org/rfc/rfc7426.txt.

H. Hasan, J. Cosmas, Z. Zaharis, P. Lazaridis, and S. Khwandah. Creating and managing dynamic mpls tunnel by using sdn notion. In *2016 International Conference on Telecommunications and Multimedia (TEMU)*, pages 1–8, July 2016. doi: 10.1109/TEMU.2016.7551923.

HashiCorp. Vagrant Website, 2017. https://www.vagrantup.com/.

Tao Huang, F Richard Yu, Chen Zhang, Jiang Liu, Jiao Zhang, and Junjie Liu. A survey on large-scale software defined networking (sdn) testbeds: Approaches and challenges. PP: 1–1, 11 2016.

IANA. Address Family Numbers, 2018a. URL https://www.iana.org/assignments/address-family-numbers/address-family-numbers.xhtml.

IANA. Border Gateway Protocol (BGP) Parameters , 2018b. URL https://www.iana.org/assignments/bgp-parameters/bgp-parameters.xhtml.

InternetSociety. Networks, Standards and Interoperability, 2018. https://future.internetsociety.org/wp-content/uploads/2017/09/Paths-to-our-Digital-Future-Driver-of-Change-Networks-Standards-Interoperability.pdf.

ISIS-TE. IS-IS Extensions for Traffic Engineering. RFC 5305, RFC Editor, October 2008.

Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 3–14. ACM, 2013.

William Johnston, Chin Guok, and Evangelos Chaniotakis. Motivation, design, deployment and evolution of a guaranteed bandwidth network service. In *Proceedings of the TERENA Networking Conference*, 2011.

Juniper. Learn About Quality of Service (QoS), 2015. URL https://www.juniper.net/documentation/en_US/learn-about/LA_QoS.pdf.

P. Lin, J. Bi, S. Wolff, Y. Wang, A. Xu, Z. Chen, H. Hu, and Y. Lin. A west-east bridge based sdn inter-domain testbed. *IEEE Communications Magazine*, 53(2):190–197, Feb 2015. ISSN 0163-6804. doi: 10.1109/MCOM.2015.7045408.

MEF. SD-WAN Edge CPE + Edge vCPE Use Case, Sep 2017. URL https://wiki.mef.net/pages/viewpage.action?pageId=65376832.

Microsoft. The Skype for Business Software Defined Networking (SDN) Interface, 2016. URL https://blogs.technet.microsoft.com/nexthop/2017/03/08/the-skype-for-business-software-defined-networking-sdn-interface/.

mininet. Mininet Overview, 2018. http://mininet.org/.

I. Monga, C. Guok, W. E. Johnston, and B. Tierney. Hybrid networks: lessons learned and future challenges based on esnet4 experience. *IEEE Communications Magazine*, 49(5):114–121, May 2011. ISSN 0163-6804. doi: 10.1109/MCOM.2011.5762807.

K. Nichols, V. Jacobson, and L. Zhang. A two-bit differentiated services architecture for the internet. RFC 2638, RFC Editor, July 1999.

odl-bgp. Opendaylight Documentation: BGP User Guide, 2018. URL http://docs.opendaylight.org/en/stable-nitrogen/user-guide/bgp-user-guide.html.

odl:carbon. OpenDaylight Carbon. https://www.opendaylight.org/what-we-do/current-release/carbon.

odl:karaf. OpenDaylight Karaf Features, 2018. http://docs.opendaylight.org/en/stable-nitrogen/getting-started-guide/karaf_features.html.

odl:md-sal. OpenDaylight Controller:MD-SAL:MD-SAL App Tutorial, 2018. https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:MD-SAL_App_Tutorial.

odl:overview. Platform Overview, 2018. https://www.opendaylight.org/what-we-do/odl-platform-overview.

odl:web. Opendaylight SDN Controller Platform, 2016. https://www.opendaylight.org/.

OGF. Network Service Framework v2.0, 2014a. URL http://www.ogf.org/documents/GFD.213.pdf.

OGF. NSI Connection Services v2.0, 2014b. URL https://www.ogf.org/documents/GFD.212.pdf.

ONF. The Benefits of Multiple Flow Tables and TTPs, February 2015. URL https://www.opennetworking.org/wp-content/uploads/2014/10/TR_Multiple_Flow_Tables_and_TTPs.pdf.

ONOS. SDN-IP Architecture, 2016. URL https://wiki.onosproject.org/display/ONOS/SDN-IP+Architecture.

OSPF-TE. Traffic Engineering Extensions to OSPF Version 3. RFC 5329, RFC Editor, September 2008.

ovs qos. Open vSwitch- Quality of Service (QoS). URL http://docs.openvswitch.org/en/latest/faq/qos/.

ovsdb-ietf. The open vswitch database management protocol. RFC 7047, RFC Editor, December 2013. URL http://www.rfc-editor.org/rfc/rfc7047.txt. http://www.rfc-editor.org/rfc/rfc7047.txt.

Paul Jakma. Quagga Routing Suite, 2017. https://www.nongnu.org/quagga/.

PCECC. PCECC as SDN Transition Solution, July 2016. URL https://wiki.onosproject.org/display/ONOS/PCECC+as+SDN+Transition+Solution.

G. Petropoulos, F. Sardis, S. Spirou, and T. Mahmoodi. Software-defined inter-networking: Enabling coordinated qos control across the internet. In *2016 23rd International Conference on Telecommunications (ICT)*, pages 1–5, May 2016. doi: 10.1109/ICT.2016.7500361.

Kevin Phemius, Mathieu Bouet, and Jérémie Leguay. Disco: Distributed multi-domain sdn controllers. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–4. IEEE, 2014.

Łukasz Podleski, Eduardo Jacob, José Aznar-Baranda, Xavier Jeannin, Kurt Baumann, and Christos Argyropoulos. Multi-domain software defined network: exploring possibilities in. *TNC 2014*, 2014.

Rui Prior and S. Sargento. Towards inter-domain qos control. In *11th IEEE Symposium on Computers and Communications (ISCC'06)*, pages 731–739, June 2006. doi: 10.1109/ISCC.2006.164.

QoSBlog1. The QOS World . URL http://what-when-how.com/qos-enabled-networks/the-qos-world/.

QoSBlog2. The QOS World. URL http://what-when-how.com/qos-enabled-networks/the-qos-tools-part-1/.

QoSBlog3. The QOS World. URL http://what-when-how.com/qos-enabled-networks/policing-and-shaping-qos-enabled-networks-part-1/.

QoSBlog4. The QOS World. URL http://what-when-how.com/qos-enabled-networks/queuing-and-scheduling-qos-enabled-networks-part-3.

RTM-SDN Group. Automating Unified Communications Quality of Experience (QoE) Use Case Specification, July 2014. URL http://portal.imtc.org/DesktopModules/Inventures_Document/FileDownload.aspx?ContentID=21939.

RTM-SDN Group. Automating Quality of Experience and Diagnostics for Cloud-Hosted Real-Time Media Services, April 2017. URL http://portal.imtc.org/DesktopModules/Inventures_Document/FileDownload.aspx?ContentID=22062.

R. Santos, Z. Bozakov, S. Mangiante, A. Brunstrom, and A. Kassler. A neat framework for application-awareness in sdn environments. In *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–2, June 2017. doi: 10.23919/IFIPNetworking.2017.8264887.

C. Scoglio, T. Anjali, J. C. de Oliveira, I. F. Akyildiz, and G. UhI. Team: A traffic engineering automated manager for diffserv-based mpls networks. *IEEE Communications Magazine*, 42(10):134–145, Oct 2004. ISSN 0163-6804. doi: 10.1109/MCOM.2004.1341272.

Shitanshu Shah, Keyur Patel, Sandeep Bajaj, Luis Tomotaki, and Mohamed Boucadair. Inter-domain sla exchange attribute. Internet-Draft draft-ietf-idr-sla-exchange-10, IETF Secretariat, January 2017. URL http://www.ietf.org/internet-drafts/draft-ietf-idr-sla-exchange-10.txt. http://www.ietf.org/internet-drafts/draft-ietf-idr-sla-exchange-10.txt.

X. Tu, X. Li, J. Zhou, and S. Chen. Splicing mpls and openflow tunnels based on sdn paradigm. In *2014 IEEE International Conference on Cloud Engineering*, pages 489–493, March 2014. doi: 10.1109/IC2E.2014.20.

JP. Vasseur, R. Zhang, N. Bitar, and JL. Le Roux. A backward-recursive pce-based computation (brpc) procedure to compute shortest constrained inter-domain traffic engineering label switched paths. RFC 5441, RFC Editor, April 2009.

velocloud. MPLS Deployment: An Assessment, 2018. URL http://www.velocloud.com/sd-wan/mpls.

Philip Wette. Testing SDN behavior with Mininet. URL http://www.linux-magazine.com/index.php/layout/set/print/Issues/2014/162/Mininet/(tagID)/338.

Quintin Zhao, Zhenbin Li, Boris Khasanov, Zekung Ke, Luyuan Fang, Chao Zhou, Telus Communications, and Artem Rachitskiy. The use cases for using pce as the central controller(pcecc) of lsps. Internet-Draft draft-ietf-teas-pcecc-use-cases-01, IETF Secretariat, May 2017. URL http://www.ietf.org/internet-drafts/draft-ietf-teas-pcecc-use-cases-01.txt.

Appendix A

BGP-SDN control-plane in our Testbed

In this chapter, integration of BGP into our SDN (WaterFed) testbed is explained. Integration of BGP is essential as it the de-facto inter-domain routing protocol of Internet and logically connects different distributed autonomous network domains. Therefore in keeping up with this construct, we similarly have to deploy BGP to control routing between different virtual domains of our testbed, and make the wider network function as a whole. This widespread deployment of BGP as the inter-domain interface has also been leveraged in this Thesis to propose its extension into a generic inter-domain interface. An application of this was explained earlier through *Federation Engine* plugin, which transmits MaaS or any other federation plugin's data to neighbouring domains over this BGP interface.

Deploying BGP is thus essential for our proposed MaaS Framework and Testbed. In this chapter, its integration into our testbed is detailed from 3 perspectives- (a) BGP in an SDN scenario i.e how we BGP peer different domain SDN controllers, (b) device-plane BGP (*Quagga*) configuration, and (c) control-plane (*ODL SDN controller*) BGP configuration.

A.1 BGP in SDN scenario

Two possible approaches for BGP integration into a multi-domain software-defined network and general legacy BGP configuration for reference are illustrated below and in Fig. A.1.

(a) Generally speaking, legacy BGP control-plane consists of an eBGP session between domain border nodes to exchange or propagate IP/multi-protocol endpoint reachability information (Fig. A.1a). Thereafter, external routes from eBGP routers are distributed internally into the network through an iBGP session with other domain nodes (Fig. A.1a).

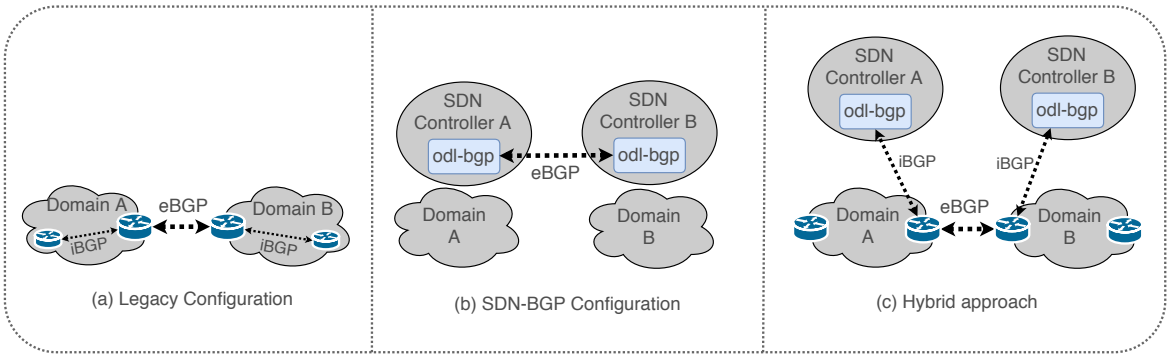


Fig. A.1 General legacy and SDN-BGP configurations, with the hybrid approach being deployed in our Testbed.

(b) SDN controller is a movement towards a consolidated centralized network control-plane. Therefore, it makes sense to also give control of domain BGP process to controller. For this purpose, *odl-bgpcep-bgp (shortened to odl-bgp)* plugin is developed within Opendaylight community and available open-source which provides complete BGP speaker functionality in controller as vendors provide in devices. So quite simply, BGP is SDN scenario can be integrated through a direct eBGP session between different neighbouring domain controllers (Fig. A.1b).

A significant further advantage of deploying BGP control-plane in SDN controller is that other network control application can efficiently access BGP's routing information base (RIB), as well as its inter-domain interface as *Federation Engine* does.

(c) Another approach for SDN-BGP integration is to keep the legacy widely deployed eBGP functionality between data-plane devices intact, and peering device-place eBGP nodes with SDN controller in an iBGP configuration (*hybrid approach*, Fig. A.1c).

We incorporate the former *hybrid* approach for SDN-BGP control-plane in our framework and Testbed. It represents a more sustainable and interoperable solution as the widely deployed inter-domain eBGP peering arrangements are maintained, preventing the need of migrating it to between SDN controllers which would pose a significant administrative task considering the importance of eBGP to security and functioning of a domain. Also, a direct approach (Fig. A.1a) would significantly expose SDN controller to any security risks infected through eBGP.

Since we incorporate a hybrid approach, BGP speaker is thus required in both data-plane and the SDN control-plane. The next two sections explain the BGP speaker used in both these planes.

A.2 Device-plane BGP (Quagga)

Our Testbed data-plane only contains openvSwitch (ovs) devices, which do not support BGP protocol. To mitigate this shortcoming, *Quagga* which is a software routing suite and provides BGP speaker functionality in Linux machines is used. Quagga is fairly well documented, easy to configure and a snapshot of its configuration is shown for reference in Figure A.2. With a single Quagga installation and configuration script in our domain VM, we can instantiate multiple independent BGP speakers and bind them to different network interface or ovs devices. This is essential when we want to have more than one BGP device in our network as in case of Domain B (B-BR1 B-BR2, Fig. A.2). This is done by specifying *multiple-instance* property in Quagga config script with each instance (speaker) identified by a different view number. Notice though the lack of any multiple-instance property in case of Domain A.

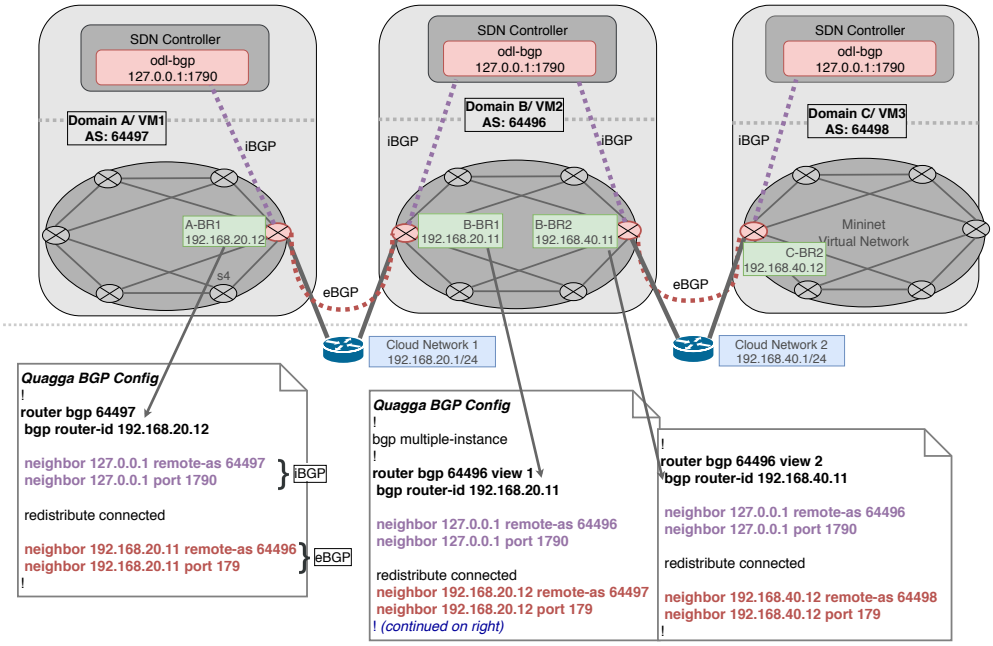


Fig. A.2 BGP control-plane in our multi-domain SDN testbed, with associated device-plane BGP (Quagga) configuration.

To give a quick overview of major Quagga configuration parameters, the network interface onto which BGP speaker has to be tied (an ovs device in our case) is specified by *router-id* and the AS number of the speaker is specified by *router bgp* attribute. Multiple BGP peers can be specified through *neighbour* attributes. If the AS number of *neighbour* is same as the

host BGP speaker's, iBGP peering will be attempted (*purple*, Fig. A.2) and if it's different, eBGP peering will be attempted (*red*, Fig. A.2).

Also recollect as shown in Figure A.2 that the cloud (Openstack) network forms the indirect inter-domain link in our testbed. A Quagga template (parameterized configuration script) is packaged in the base VM image along with Quagga binaries. Later when the automated provisioning is attempted through Vagrant by spinning up a new virtual domain from this base VM image, concrete parameters are passed to the template and Quagga restarted to obtain desired BGP configuration.

A.3 Control-plane BGP configuration (odl-bgp)

Shortened from odl-bgpcep-bgp (opendaylight + bgp + pcep), it is the plugin developed within the Opendaylight community to provide full BGP speaker functionality in SDN controller. Similar to how Quagga provides BGP functionality in data-plane, *odl-bgp* provides such functionality in SDN control-plane. We utilize this plugin to peer with data-plane BGP devices in iBGP mode (*iBGP*, Fig. A.2), resulting which - external routes from eBGP session can be shared into iBGP session and thus received by the controller, and internal routes shared to the same eBGP process to be advertised inter-domain, a process necessary to facilitate end-to-end routing as illustrated in section 4.3.4.

BGP peers of the *odl-bgp* plugin can be specified through a configuration-file at startup or via its REST APIs at runtime. A triad of RIBs (*effective-rib-in, adjacent-rib-in, adjacent-rib-out*) are instantiated by the plugin for every BGP session or peer, apart from another two singleton global RIB (*application-rib, local-rib*) as shown in Figure A.3. These RIBs maintain routing information and are integral part of any general BGP speaker, with *odl-bgp* specifically providing advantage of efficient programmatic access to any of the RIBs to other controller applications interested in routing information or other use-cases such as inter-domain communication like with Federation Engine. These are explained below and also shown in Figure A.3.

(a) *adjacent-rib-in*: This routing table consists of unfiltered BGP routes received from the associated peer (Fig. A.3).

(b) *effective-rib-in*: This routing table consists of filtered routes from the *adjacent-rib-in* above. The filter is standard to BGP protocol, and dependant on the role between the peers – iBGP, eBGP or Route Reflector.

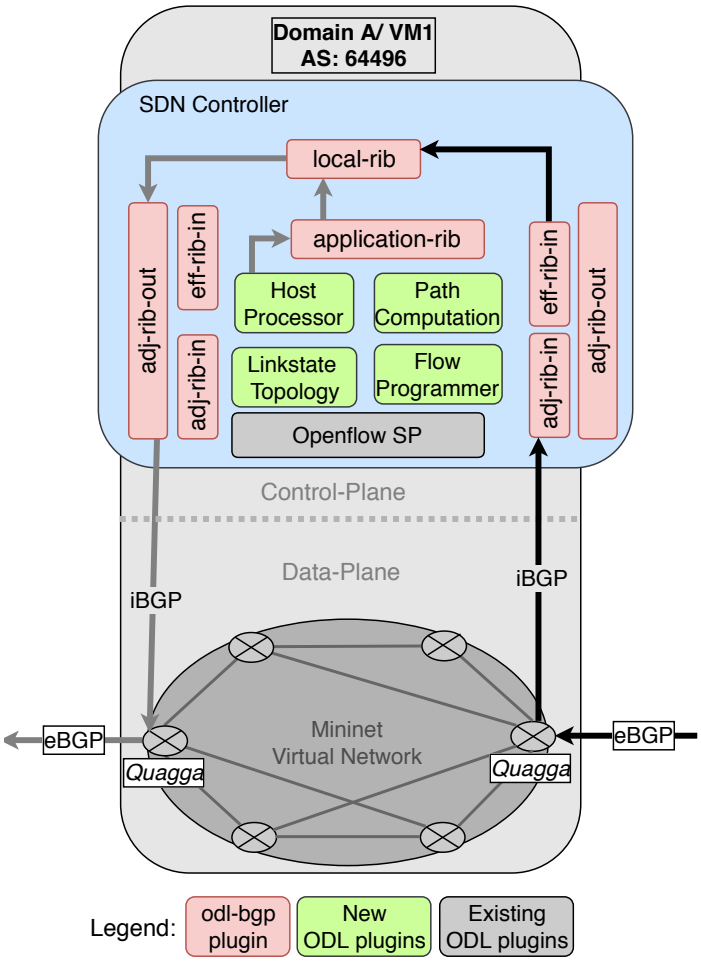


Fig. A.3 Structure of SDN-BGP control-plane implemented by *odl-bgp* plugin showing its different Routing Information Base (RIBs) that store its data.

(c) *adjacent-rib-out*: This routing table consists of routes that are distributed out to the associated BGP peer to be further advertised inter-domain.

(d) *application-rib*: This rib holds information on all domain local routes/prefixes majorly relating to the end-hosts that are tracked by *host-processor* plugin (Fig. A.3). Routes from there spill into *local-rib* and from there into *adjacent-rib-out(s)* to be advertised inter-domain.

(e) *local-rib*: It is a global RIB that maintains consolidated routing information with routes from all the peers received through their *effective-rib-in*, and thereafter employing best-route-selection algorithm for duplicate routes received from multiple peers or *rib-ins*. Apart from external routes, it also contains internal routes primarily received from *application-rib*.

Appendix B

MD-SAL overview

Model-Driven Service Abstraction Layer (MD-SAL) is a set of Opendaylight infrastructure services aimed at providing common and generic support to controller application developers for data-storage, messaging and accessing data across applications. It was born out of the realization that abstracting the entire network in controller natively at once and then offering APIs over it is not a sustainable solution as it limits flexibility and hinders inclusion of newer technologies. Instead any one (application) should be able to provide an abstraction (*producers*), and any one should be able to choose which abstraction APIs they want to use (*consumers*). This abstraction, which is in the form of network control data and service APIs, is required by MD-SAL to be modelled through *yang* domain specific language and thereafter provides a tree database for storing that data. MD-SAL also provides a controller-wide consistent manner for other applications to access any other application's data or services, be it externally through Northbound Rest APIs or internally via Java interfaces/APIs.

The following subsections briefly explain storage and data-access services provided by ODL's MD-SAL layer, and an example scenario to explain it in function.

A. Data-tree store and Data-change events

MD-SAL maintains an in-memory tree data-store, wherein each (*producer*) application's data defined in its yang model is stored as a sub-tree. It automatically generates Java APIs for each of the leaf node or sub-tree of application's data-tree to be accessed, read, written, updated, merged or deleted explicitly by the producer. Importantly, it also generates Java APIs for many (*consumer*) applications to subscribe to a particular change in the tree datastore, and further delivers asynchronous data-change events to them whenever any change is performed by the producer plugin.

B. Notifications

MD-SAL allows ODL applications to *produce* notifications on some critical events. These notifications, similar to data, are defined in their yang model. Whenever a notification is produced, MD-SAL further routes that event to its multiple subscribed *consumer* applications.

C. RPC

While the previous two data-access patterns were routed from *producer* to *consumer* applications via MD-SAL layer, further to them, *consumers* can directly access an application's data or service through Remote Procedure Call (RPC) mechanism which is also similarly defined by *producers* in their yang model.

D. Example Scenario

The Figure B.1 scenario described in this section brings out the crucial role of MD-SAL in the functioning of Opendaylight SDN controller, and also further explains its data-store and access services briefed above. The scenario, as also explained at odl:md-sal (2018), consists of 3 applications or plugins – Openflow Southbound Plugin, Learning Switch and Forwarding Rule Manager (FRM) applications which work together to install an openflow rule for an incoming flow.

1. Generally speaking, an openflow switch forwards incoming packets to the controller if there is no matching forwarding rule for it installed on the switch (1, Packet-In). Thereafter, the packet is received by the controller openflow plugin (2, Fig. B.1) which generates a *notification* for this event. All plugins define the notifications they produce in a yang model, as shown in Figure. B.2 (top) for this particular notification offered by openflow plugin. MD-SAL also generates Java interfaces for other interested applications like *Learning-Switch* in this scenario to implement and receive this notification (bottom, Fig. B.2), and delivers it once produced (3).

2. On receiving the packet-in notification, *learning-switch* application decides on how this packet should be handled on the switch. Therefore, it uses the RPC offered again by openflow plugin to convey this information (4), which is further conveyed to the switch by the plugin (6,7). Again this RPC is routed by the MD-SAL layer (5).

   However, the function explained above is only sufficient for the forwarding of packet just received at the openflow-plugin through *packet-in* event. To ensure that further

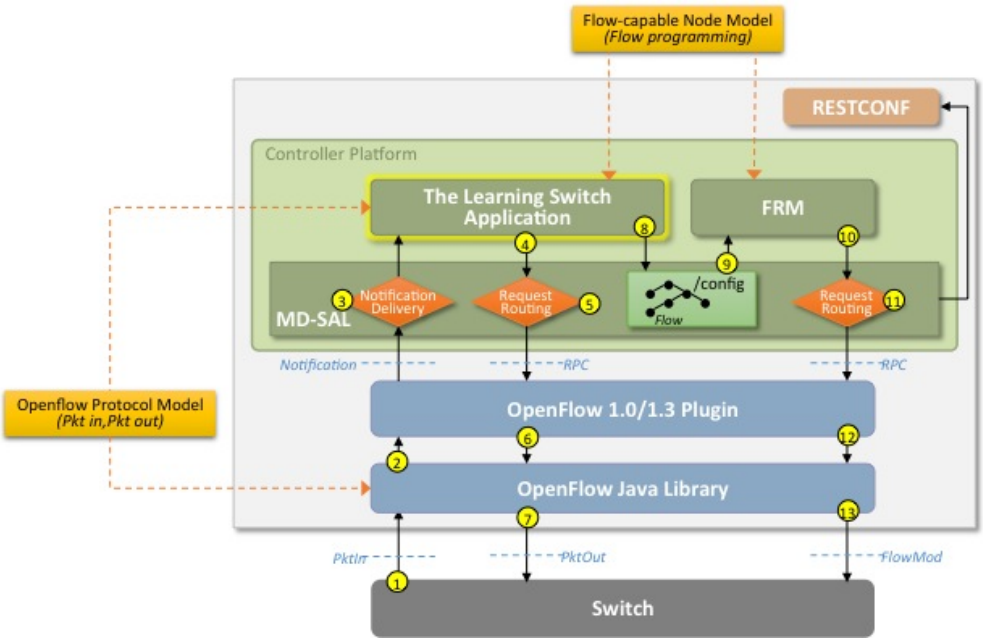


Fig. B.1 Different ODL application plugins, and MD-SAL routing data between them. Image from odl:md-sal (2018).

```
## openflowplugin/packet-processing.yang
module packet-processing {

        notification packet-received {
                description "Delivery of incoming packet";

                leaf connection-cookie {
                        type connection-cookie;
                }
                leaf ingress {
                        type inv:node-connector-ref;
                }
                leaf payload {
                        type binary;
                }
        }
}
```

```
// openflowplugin/PacketProcessingListener.java
/* Interface for receiving YANG notifications defined in module <packet-processing*/
public interface PacketProcessingListener extends NotificationListener
{
        void onPacketReceived(PacketReceived notification);
}
```

Fig. B.2 Openflow plugin modelling the packet-in notification in a yang file (top), and MD-SAL generating interface for other applications to implement and receive this notification (bottom).

packets of this flow are routed by the switch forwarding-plane itself and not forwarded to the SDN control-plane, a permanent openflow rule is needed to be installed which is handled by *Forwarding Rule Manager* (FRM) application.

3. FRM defines a data-model in its *yang* file which maintains information, stored in the MD-SAL datastore, on the openflow rules to be installed on each data-plane openflow switch. MD-SAL also generates Java APIs to write to this data-tree, which *Learning Switch* uses to write an openflow rule (8) that should forward all the further packets of flow. MD-SAL also delivers any change in the data-model to its subscribers, FRM in this scenario (9). Presented with this change, FRM uses openflow-plugin RPC which installs a permanent forwarding rule on the switch (10-13, *FlowMod*).

MD-SAL is thus an important infrastructure component of Opendaylight controller platform which provides network application developers (like FRM) convenient access to *abstraction* (data and services) offered by the *providers* (like Openflow-plugin) which immensely advances the usability, extensibility and flexibility of the SDN platform.

Appendix C

QoS mechanism in network devices

This thesis presented a control-plane architecture in MaaS for end-to-end QoS provisioning. To transfer QoS control information into per-hop data-plane devices (openvswitch), it assumed the use of standard Openflow interface exposed by them without going into the details of how these devices actually guarantee differentiated service. Also while providing a background overview in chapter 2, similar knowledge on the mechanisms switches and routers employ to provide QoS was skipped. This Appendix fills the gap and provides an overview of the common mechanisms followed by the data-plane devices to implement QoS, followed by specific disciplines supported by the device used for experimentation – openvswitch.

C.1 Introduction

With different types of traffic like voice and data now converging onto a common packet-switched network, it's been very essential for network devices to differentiate traffic and ensure according service in terms of delay, bandwidth, priority and other parameters i.e. to support QoS. Each network device independently processes QoS for different traffic types or classes referred to as per-hop behaviour (PHB), and thus is essential to make consistent configuration to achieve end-to-end QoS objectives.

Fig. C.1 shows the different core PHB functions performed by a network device from ingress to egress which collectively provide QoS (Juniper, 2015) (QoSBlog1). When a packet arrives in the device at ingress, it can be *classified* into one of few traffic classes, optionally limited in bandwidth by the *policer* before or after the classification, transmitted into the *queue* of its destination egress interface that confirms to its contracted QoS, with *scheduler* working to service many queues for transmission according to their relative priority. Finally

before the transmission, packets can be *marked* differently to be classified appropriately by the rest of network. These core QoS functions are explained briefly next.

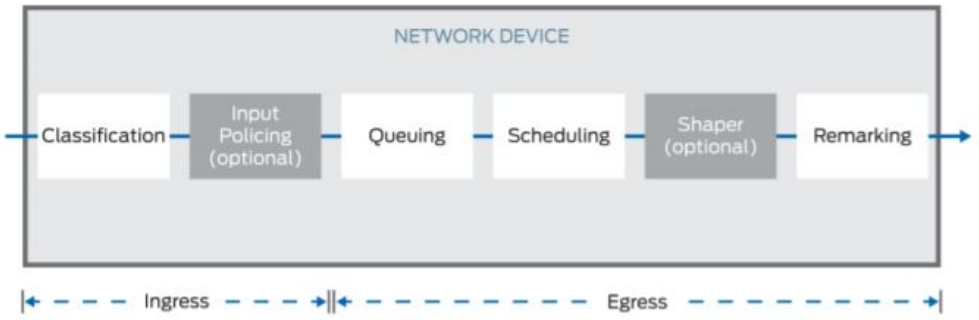


Fig. C.1 Core QoS functions of a general network device. Image from Juniper (2015).

C.2 Core QoS Functions

C.2.1 Classification

QoS generally functions by classifying incoming traffic into different classes or class of services (CoS). Traffic within each class is treated similarly based on set QoS behaviour or PHB performed by the subsequent QoS functions like policers and schedulers.

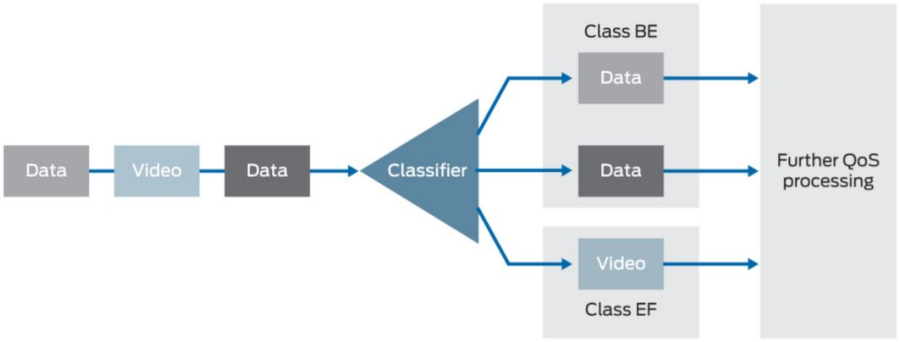


Fig. C.2 An example classification process of packets into two classes - Best Effort and Expedited Forwarding. Image from Juniper (2015).

Classification happens at the ingress when packets enter the network device. Input to classifier are only the packets and the output can be one of multiple defined CoS. Packets go through a set of IF/THEN rules inside the classifier where the match to a CoS is specified on the basis various fields in packet header, like IP ToS in DiffServ enabled networks or

MPLS EXP in similar networks or even on the basis of combination of multiple fields like source/destination addresses and protocol. Fig. C.2 illustrates this classification process.

C.2.2 Policing

Policers limit or control traffic bursts at the ingress to a specified rate and can be applied to entire interface or to varied CoS differently. Due to this function, they act as the first line of defence against the congestion. Apart from rate-limiting, policers generally also allow a configurable amount of bursts to pass through.

Fig. C.3 illustrates the function of a policer. This is an example of hard policer wherein the excess traffic is dropped, whereas the other type of soft policer mark excess traffic to a different, ideally low priority, CoS. They are implemented using the concept of token buckets (QoSBlog3), the configurable parameters to which are the output rate (in bits per second) and the burst size (in bytes).

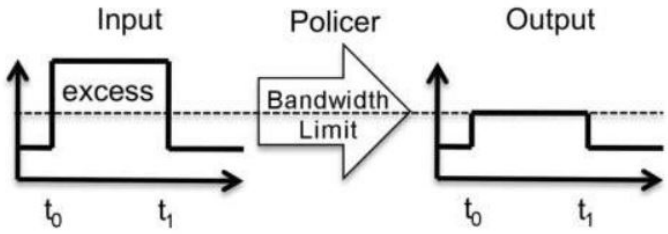


Fig. C.3 Hard policer in action. Image from QoSBlog2.

In many QoS implementations, policer works in conjunction with metering and coloring tool. **Metering** tool measures the incoming traffic rate and compares it against two commonly understood values - Committed Information Rate (CIR) and Peak Information Rate (PIR), therafter **coloring** them differently as shown in Fig. C.4. Green traffic i.e less than CIR is referred to as in-contract, while the other two types are referred to as out-of-contract traffic. These differently colored traffic are input to the policer to be treated differently. While green traffic is provided normal treatment as per the contract, red traffic is generally discarded and yellow is also provided normal treatment if the resources are available with statistical multiplexing providing the ability to do so.

C.2.3 Shaping

Similar to policers, shapers also limit the traffic to a defined rate. However, while the policer generally drops excess traffic, shaper holds it in a queue until it can be transmitted at the

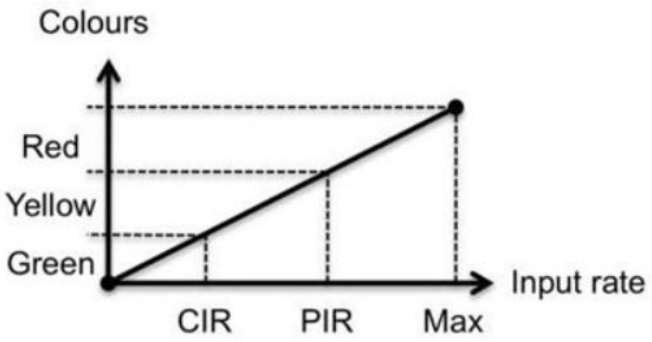


Fig. C.4 Representation of how metering tool compares incoming traffic to CIR and PIR values and colors them. Image from QoSBlog2.

confirmed rate. Shaping function is generally applied on the egress traffic. Fig. C.5 illustrates the shaping function. Since shaping introduces unwarranted queueing delay into the traffic, it leads to the degradation fo service for delay sensitive traffic like VoIP and IPTV.

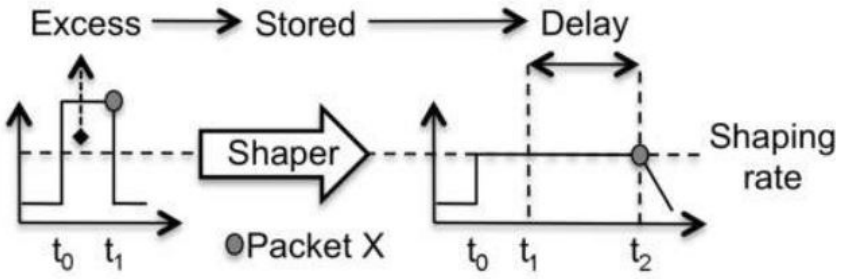


Fig. C.5 QoS shaper in action. Image from QoSBlog2.

C.2.4 Queuing and Scheduling

Queuing and scheduling are the most important QoS functions in a network device. While the previous other functions operated on a single class of traffic, these two combined truly realize QoS by introducing unfairness, and prioritizing some class of traffic over others.

On an egress interface, multiple queues can be created that hold traffic for different classes while waiting for a transmission slot by the scheduler which provides so on the basis of priority of queue and indirectly class. A one-to-one mapping between the CoS and the queue can be considered for simplicity. Queues have a finite buffer length, and when a packet is decided to be transmitted into the queue, its fill level determines whether to enqueue the

packet, or drop it, even in the scenario of queue being not completely full through mechanism like Random Early Detection (RED).

Scheduler services packet from the queues for transmission in a manner that respects the service associated, even in the times of congestion. For instance, an EF queue associated with low-delay VoIP traffic should be scheduled on priority whenever the packet arrives to it. Also queues confirming to different bandwidths should be scheduled for delivery according to their relative weights. While the scheduling discipline employed by a network device can be vendor or model specific, a common basis of modern scheduling algorithms is Priority Based - Weighted Deficit Round Robin (PB-DWRR) (QoSBlog4).

C.3 QoS mechanism in Open vSwitch

Previous section provided an overview of the general QoS functions performed by network devices. Open vSwitch which we used for experimentation however doesn't boast of QoS features comparable to vendor switches. Its QoS support is though reliant on using common Linux tools, most notably Traffic Control (TC) and Hierarchical Token Bucket (HTB) queuing discipline. TC-HTB tool provides a command-line interface to create multiple queues on a Linux interface and thereafter shape or police ingress or egress traffic.

Open vSwitch implementation of QoS however doesn't support all the QoS features of TC-HTB, and specifically provides APIs for following functionality.

```
ovs-vsctl add-br br0
        add-port br0 veth0 --ofport_request=5 -- \
        set port veth0 qos=@newqos -- \
        --id=@newqos create qos type=linux-htb \
                other-config:max-rate=10000000 \
                queues:123=@vifqueue1 queues:234=@vifqueue2 -- \
                --id=@vifqueue1 create queue other-config:max-rate=3000000 --
                --id=@vifqueue2 create queue other-config:max-rate=7000000
```

Fig. C.6 Open vSwitch command to create a bridge, attach an interface and create queues.

- **Creating queues and shaping egress traffic**: For traffic that egresses from a switch, OVS supports traffic shaping (ovs qos). Following Fig. C.6 gives an overview of it. The command creates a new ovs bridge *br0* and attaches an interface *veth0* on to the bridge. Thereafter, a new QoS configuration and therein 2 queues are created on the set interface where the traffic is shaped to 3 Mpbs and 7 Mbps respectively. Fig. C.7 gives

an Openflow example of how an incoming MPLS packet labeled 11 can be pushed into any of those queues thereafter.

- **Ingress traffic policing:** On the other end, a policing policy can be created to drop the excess traffic that enters an interface as shown in Fig C.8 wherein traffic is limited to a maximum of 10 Mbps with excess being dropped and allowing a burst of 8000 Bytes.

```
ovs-ofctl add-flow br0
 ↪  dl_type=0x8847,mpls_label=11,actions=set_queue:123,output:5
```

Fig. C.7 Open vSwitch command to push a MPLS packet into a queue.

```
$ ovs-vsctl set interface veth0 ingress_policing_rate=10000
$ ovs-vsctl set interface veth0 ingress_policing_burst=8000
```

Fig. C.8 Open vSwitch command for traffic policing.

List of Acronyms

A

AQS

Automated QoE Service. 14, 15, 16

ARP

Address Resolution Protocol. 61

B

BGP

Border Gateway Protocol. vii

BRPC

Backward Recursive Path Computation. 24

C

CAL

Control Abstraction Layer. 9, 31

CE

Customer Edge. 35, 36

CoS

Class of Service. 4

CPE

Customer Premises Equipment. 35, 53

CSPF

Constrained Shortest Path First. 33, 77

D

DDS

Document Distribution Service. 23, 43

DSCP

DiffServ Code-Point. 14, 55

E

ECS

Enterprise Collaboration Systems. 3

F

FE

Federation Engine. 42

H

HTB

Hierarchical Token Bucket. 121

I

IDCP

Inter-Domain Controller Protocol. 23, 24

IGP

Interior Gateway Protocol. 20

ISIS-TE

Intermediate System to Intermediate System. 20

IXP

Internet Exchange Points. 2, 3

J

JRE

Java Runtime Environment. 68

L

LDP

Label Distribution Protocol. 20

LFEC

Label Forwarding Equivalence Class. 9, 19, 35, 36, 53

LFIB

Label Forwarding Information Base. 20, 34, 35

LSP

Label Switched Path. vii, 5, 9, 18, 27

M

MaaS

MPLS-as-a-Service. vii, viii, 6, 27

MATE

MPLS Adaptive Traffic Engineering. 21

MD-SAL

Model-Driven Service Abstraction Layer. 71

MPLS

Multi-protocol Label Switching. vii, viii, 5, 27

MPLS-TE

Multi-protocol Label Switching - Traffic Engineering. 18

N

NaaS

Network-as-a-Service. 16, 17

NFV

Network Function Virtualization. 1

NSF

Network Services Framework. 23, 24

NSI-CS

Network Services Interface Connectivity Service. 23

O

ODL

Opendaylight. 9, 29, 30, 33, 60

OSPF-TE

Open Shortest Path First. 20

ovs

openvswitch. 9, 64

P

PC

Path Computation. 19, 31

PCC

PC Client. 19, 21

PCE

PC Element. 19, 21

PCECC

PCE as a Central Controller. 21

PCEP

PCE Protocol. 19

Q

QoS

Quality of Service. vii

R

RATES

Routing and Traffic Engineering Server. 21

RIB

Routing Information Base. 41, 43, 45, 76

RSVP

Resource Reservation Protocol. 4, 5, 20

RTM

Real Time Media. 14

S

SDP

Service Delivery Platform. 16, 17

SD-WAN

Software-Defined Wide Area Network. 2, 3, 8

SDN

Software Defined Networking. 1, 2

SP

Southbound Plugin. xi, 32, 73

T

TC

Traffic Control. 121

TE

Traffic Engineering. vii, 19, 27, 33

TEAM

Traffic Engineering Automated Manager. 21

TED

Traffic Engineering Database. 20

TSSG

Telecommunications Software and System Group. 59

U

UC

Unified Communication. 3, 14

V

VC

Virtual Circuit. 5, 18

VXLAN

Virtual Extensible LAN. 60, 64, 65

W

WAN

Wide Area Network. 4