AUTO-BANDWIDTH CONTROL IN DYNAMICALLY RECONFIGURED HYBRID-SDN MPLS NETWORKS

Prepared by:
Angus Daniel Brandt

Supervised by:
Dr A Murgu (PhD)

THESIS PRESENTED FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN THE DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
FACULTY OF ENGINEERING AND THE BUILT ENVIRONMENT
UNIVERSITY OF CAPE TOWN
SOUTH AFRICA

August 2020

Declaration

I, ... ***Angus Brandt*** .., hereby declare that the work on which this thesis is based is my original work (except where acknowledgements indicate otherwise) and that neither the whole work nor any part of it has been, is being, or is to be submitted for another degree in this or any other university.

I authorise the University to reproduce for the purpose of research either the whole or any portion of the contents in any manner whatsoever.

Signature:..... Signed by candidate Date: 31 August 2020

AUTO-BANDWIDTH CONTROL IN DYNAMICALLY RECONFIGURED HYBRID-SDN MPLS NETWORKS

Summary

The proposition of this work is based on the steady evolution of bandwidth-demanding technology, which currently and more so in future, requires operators to use expensive infrastructure capability smartly to maximise its use in a very competitive environment.

In this thesis, a traffic engineering control loop is proposed that dynamically adjusts the bandwidth and route of flows of *Multi-Protocol Label Switching* (MPLS) tunnels in response to changes in traffic demand. Available bandwidth is shifted to where the demand is, and where the demand requirement has dropped, unused allocated bandwidth is returned to the network.

An MPLS network enhanced with *Software-defined Networking* (SDN) features is implemented. The technology known as hybrid SDN combines the programmability features of SDN with the robust MPLS label switched path features along with traffic engineering enhancements introduced by routing protocols such as *Border Gateway Patrol-Traffic Engineering* (BGP-TE) and *Open Shortest Path First-Traffic Engineering* (OSPF-TE).

The implemented mixed-integer linear programming formulation using the minimisation of maximum link utilisation and minimum link cost objective functions, combined with the programmability of the hybrid SDN network allows for source to destination demand fluctuations.

A key driver to this research is the programmability of the MPLS network, enhanced by the contributions that the SDN controller technology introduced. The centralised view of the network provides the network state information needed to drive the mathematical modelling of the network. The path computation element further enables control of the label switched path's bandwidths, which is adjusted based on current demand and optimisation method used.

The hose model is used to specify a range of traffic conditions. The most important benefit of the hose model is the flexibility that is allowed in how the traffic matrix can change if the aggregate traffic demand does not exceed the hose maximum bandwidth specification.

To this end, reserved hose bandwidth can now be released to the core network to service demands from other sites.

Dedications

To my son:

Xavier Daniel Brandt

So much of our time went into my studies.

Thank you

In loving memory of my family:

Prudence Emdeen Carstens (née Brandt) - Sister

Grace Magdaleen Brandt - Mom

Diedrich Johannes Danie Brandt - Dad

Acknowledgements

To my supervisor, Dr Alexandru Murgu, for his support, advice, guidance and critical input throughout the research project.

To CPUT for their financial support during this time.

To my son Xavier who had to see and experience less of me during my years of study.

To my family and friends for your continued support.

1 Table of Contents

Table of Figures **Page**

List of Tables **Page**

List of Abbreviations

API	Application Programming Interface
AS	Autonomous System
ASIC	Application Specific Integrated Circuit
ATM	Asynchronous Transfer Mode
BGP	Border Gateway Patrol
BGP-LS	Border Gateway Patrol-Link State
CE	Customer Edge
CoS	Class of Service
CPU	Central Processing Unit
CRUD	Create Retrieve Update Delete
CSPF	Constrained-Shortest Path First
DC	Data Centre
DM	Demand Matrix
ECMP	Equal Cost Multi-Path
EIGRP	Enhanced Interior Gateway Routing Protocol
EON	Elastic Optical Network
ERO	Explicit Route Object
FEC	Forward Equivalent Class
FSM	Finite State Machine
GMPLS	Generalised Multiprotocol Label Switching
H-PCE	Hierarchical Path Computation Element
HSDN	Hybrid Software Defined Networking
ICN	Information Content Networking
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IP/MPLS	Internet Protocol / Multi-Protocol Label Switching
IPTV	Internet Protocol Television
IS-IS	Intermediate System to Intermediate System
IS-IS-TE	Intermediate System to Intermediate System Traffic Engineering
ISP	Internet Service Provider
JSON	JavaScript Object Notation
LB	Load Balancing
LER	Label Edge Router
LIB	Label Information Base
LP	Linear Programming
LSA	Link State Address
LSDB	Link-state Database
LSP	Label Switched Path
LSPDB	Label Switched Path Database
LSR	Label Switched Router
MCNF	Multi-commodity Network Flow

MD-SAL	Model-Driven Service Abstraction Layer
MILP	Mixed Integer Linear Programming
MPD	Multi-period Design
MPLS	Multiprotocol Label Switching
MPLS-TE	Multiprotocol Label Switching-Traffic Engineering
NDP	Network Design Problem
NETCONF	Network Configuration Protocol
NFV	Network Function Virtualisation
ODL	OpenDaylight
OS	Operating System
OSI	Open System Interconnection
OSPF	Open Shortest Path First
OSPF-TE	Open Shortest Path First-Traffic Engineering
P	Provider
PCC	Path Communication Client
PCE	Path Computation Element
PCEP	Path Communication Element Protocol
PE	Provider Edge
POP	Point of Presence
QoE	Quality of Experience
QoS	Quality of Service
REST	Representational State Transfer
RFC	Request For Comment
RIP	Routing Information Protocol
RSVP	Resource Reservation Protocol
RSVP-TE	Resource Reservation Protocol-Traffic Engineering
SCNF	Single-commodity Network Flow
SDN	Software-Defined Networking
SNMP	Simple Network Management Protocol
SPF	Shortest Path First
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
TE	Traffic Engineering
TE LSP	Traffic Engineering Label Switched Path
TED	Traffic Engineering Database
TLV	Type/Length/Value
TM	Traffic Matrix
U-ACL	Universal Access Control Lists
VMnet	Virtual Machine Network
VoIP	Voice Over Internet Protocol
VPN	Virtual Private Network

1 Literature Review

1.1 Multiprotocol Label Switching

Multiprotocol Label Switching (MPLS), a protocol-agnostic routing technique, with its support for *Virtual Private Networks* (VPNs), which is programming that creates an encryption network over public infrastructure, is an extensively deployed architecture. The rapid growth of telecommunications networks and new *Internet Protocol* (IP) services such as *Internet Protocol Television* (IPTV) and *Voice Over Internet Protocol* (VoIP) are placing ever-increasing demands for bandwidth on infrastructure [1]. With the steady increase of customer demand and Africa being the fastest-growing region, service providers need to respond to these demands while remaining competitive and increasing profitability. Single service networks are not cost-effective.

Converged, multi-service, multi-domain, single infrastructure MPLS networks have gained adoption in the marketplace [2]. *Request For Comment* (RFC) 3031 [3], specifies the basic concepts of the MPLS architecture as introduced by the *Internet Engineering Task Force* (IETF). The goal of MPLS is to overcome the limitations of IP based forwarding, in that, for traditional IP networks, each router performed a lookup into an IP routing table to determine the next hop. This operational processing is expensive since the entire IP header had to be analysed [2]. The MPLS techniques apply to any network layer protocol, but the specification described in [3] focuses on the use of IP as the network layer protocol.

1.1.1 MPLS Overview

MPLS is a label switching mechanism used to exchange traffic. Labels are assigned on a per traffic basis and distributed through label distribution protocols. Devices distribute and receive locally assigned labels to and from other MPLS devices which allow each device to build a *Label Information Base* (LIB).

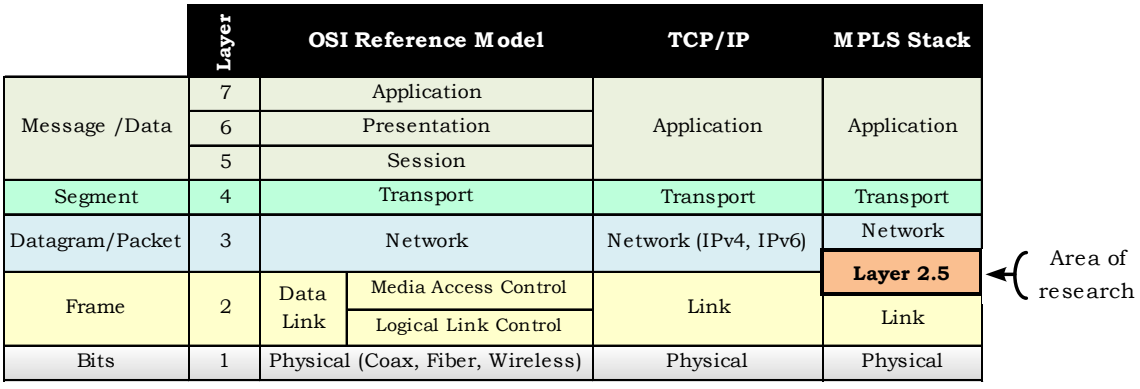

Figure 1.1: MPLS in layer 2.5.

Figure 1.1 shows the MPLS stack with the area of research at layer 2.5 of the *Open System Interconnection* (OSI) reference model. At this layer, data is encapsulated with an MPLS header before sending it to other MPLS devices.

MPLS integrates a label swapping framework with network layer routing and throughout the interior of the MPLS domain, the labels attached to the packets are used to make forwarding decisions [4]. MPLS devices receive MPLS encapsulated traffic and make forwarding decisions based on the MPLS label value in the MPLS encapsulated header. With MPLS encapsulation, a 32-bit MPLS header is inserted between Layer 2 and Layer 3 and leads to MPLS being referred to as a Layer 2.5 protocol [5].

Each MPLS device stores the label as well as the label mapping information for the types of traffic associated with each label. Traffic having the same label is part of the same *Forwarding Equivalent Class* (FEC). MPLS devices form a *Label Switched Path* (LSP) for each FEC. An LSP is a path through an MPLS network [5]. From [3], the sequence of *Label Switching Routers* (LSRs) from the ingress LSR to egress LSR that pushes a label of the same FEC onto a packet, is the LSP for a particular FEC. Each LSP is an end-to-end connection for traffic belonging to the same FEC [5]. All packets of a particular FEC, travelling from the same node, follow the same path [3]. Once a packet is assigned an FEC, subsequent routers do no further header analysis since all forwarding is driven by labels [3]. The outstanding factor of MPLS is its ability to manage and classify the traffic flows to provide better utilisation of resources, hence it forms an effective solution for network integration in dealing with traffic engineering quality demands in carrier networks [2].

1.1.2 IP/MPLS Virtual Private Networks

Bandwidth intensive and time-sensitive applications require IT infrastructure providers, to deliver high throughput while subject to secure and cost-efficient constraints [5].

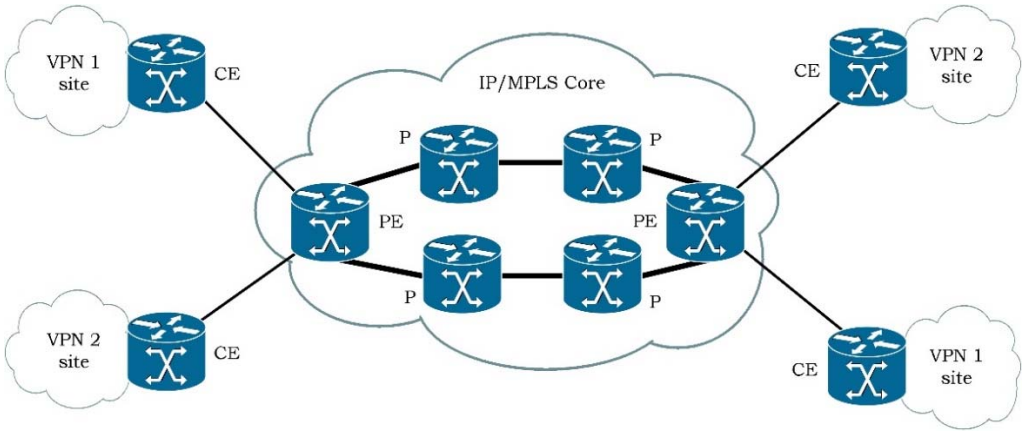

Figure 1.2: VPN in IP/MPLS context.

Figure 1.2 illustrates a VPN implementation using an *Internet Protocol/ Multiprotocol Label Switching* (IP/MPLS) core [6]. VPNs provide a cost-effective solution allowing service providers to deliver services to different customers using the same delivery backbone network while isolating each customer using vertical service instances to ensure privacy and security [6].

There has been widespread adoption of the IP/MPLS VPN technology by service providers for use in backbone networks as shown in Figure 1.3 [5].

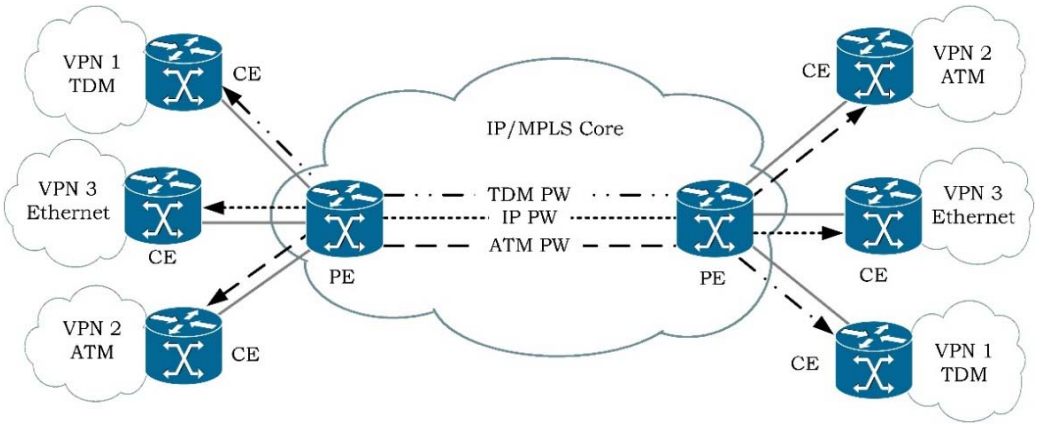

Figure 1.3: VPN service-oriented network.

IP/MPLS VPNs enable service providers to build service-oriented networks with multiple services in a single converged network. This converged network provides high availability, reliability and performance. The service-oriented network decouples the roles of the *Provider Edge* (PE) and *Provider* (P) routers by creating service instances at the edge of the network in the customer-facing PE routers. Multiple service instances on different PE routers belonging to the same service are connected using MPLS pseudowires [5].

1.2 MPLS Traffic Engineering

Traffic Engineering (TE) concerns the performance optimisation of operational networks. TE can address congestion caused by inefficient resource allocation. Minimising congestion through efficient resource allocation can decrease packet loss, decrease transit delay and increase the aggregate throughput [7].

In many large *Autonomous Systems* (ASs), TE is an indispensable function. TE facilitates the efficient and reliable operation of the network while optimising network resources and the performance of traffic [4]. Congestion problems that arise because of the inefficient allocation of resources can generally be addressed through TE, with the objective of such strategies being to minimize maximum congestion through efficient resource allocation [7].

In an AS, *Interior Gateway Protocols* (IGPs) using the *Shortest Path First* (SPF) algorithm contribute significantly to the congestion problems in that the SPF algorithm generally optimises based on a simple additive metric. The algorithm considers only the topology. Bandwidth availability and traffic characteristics are not considered in routing decision making [4]. The traffic engineering systems consist of two components namely network nodes and traffic engineering tools. The nodes must be capable of establishing and maintaining traffic-engineering tunnels. The system must be capable of establishing, modifying and deleting traffic-engineered tunnels.

The traffic-engineering life cycle, Figure 1.4 [8], allows the system to control traffic engineering functions [8]. Network performance optimisation operates within this controlled environment. The optimisation process should not be too reactive to the dynamics in the MPLS implementation, but at the same time must be fast enough for the efficient use of resources [4].

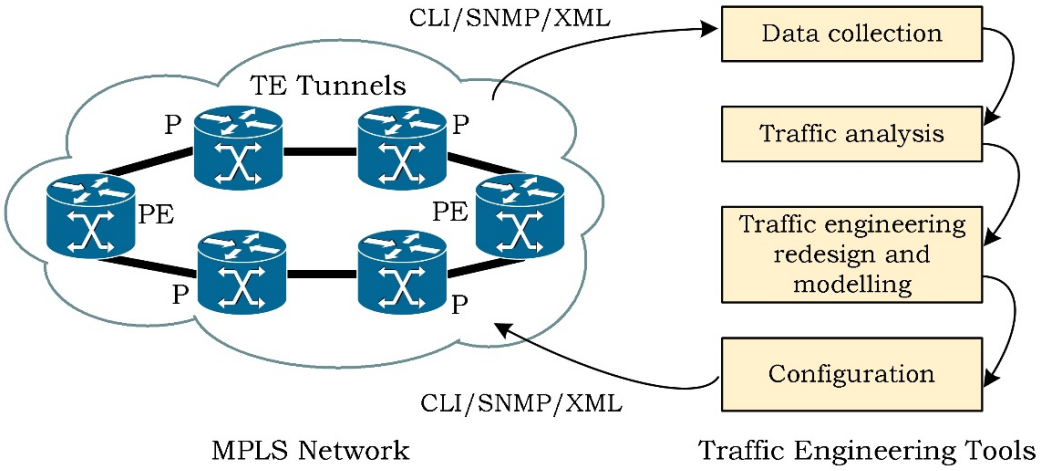

Figure 1.4: Traffic engineering life cycle.

Applications control traffic to minimise power consumption, maximising aggregate network utilisation and providing optimised load balancing as well as other generic traffic optimisation techniques ([9], [10]).

TE places the traffic where the bandwidth is available [5]. Without TE, congestion scenarios frequently manifest even when feasible alternate paths with excess capacity exist.

An overlay model such as IP over *Asynchronous Transfer Mode* (ATM) or IP over Frame Relay is generally used to extend current shortcoming in IGPs where the additional services to support traffic and resource control such as constraint-based routing are provided by the overlay model. MPLS can provide most of the functionality available from the overlay model [4].

Extension to the *Resource Reservation Protocol* (RSVP) protocol, a feature of TE over MPLS, supports the instantiation of explicitly routed LSPs, as well as the rerouting of LSPs. These LSPs also referred to as LSP tunnels allow for the implementation of a variety of policies related to network performance optimisation [11].

Through explicit routing, an ingress node of an MPLS domain can control the path through which traffic traverses from the ingress node, through the MPLS network, to an egress node [11]. Intradomain routing, which is the focus of this research, optimises traffic routing between AS border routers within a single domain. Intradomain routing is different from interdomain routing where traffic flow is optimised between ASs [12].

Figure 1.5, shows a scenario before and after the application of TE [5]. TE allows for traffic to follow a path other than the best IGP path which uses the SPF algorithm for routing. With the SPF algorithm, if two or more equal-cost shortest paths exist to a given destination, the algorithm will choose only one of the paths to transport the traffic. *Equal Cost Multi-Path* (ECMP) modifies the algorithm slightly in that if two or more equal-cost shortest paths exist, the traffic is distributed amongst the paths uniformly [13].

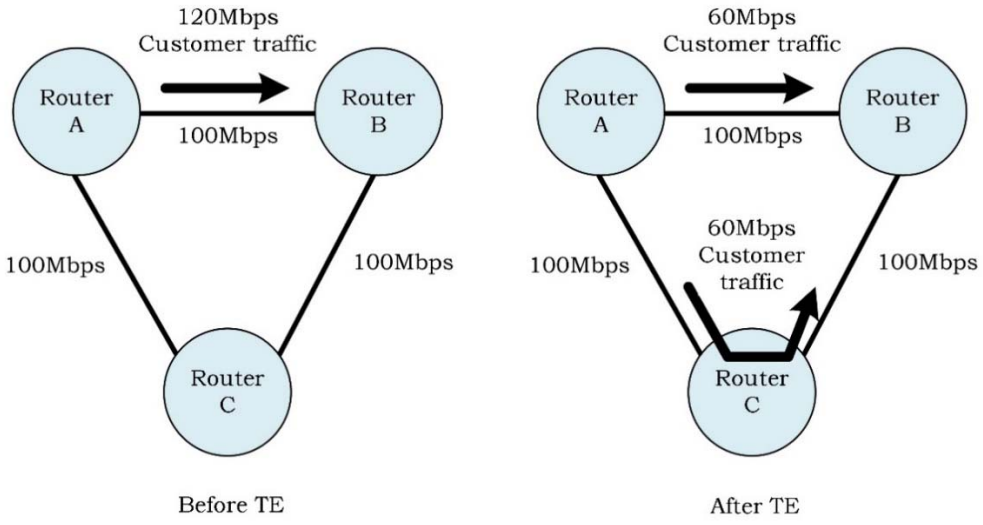

Figure 1.5: TE load placement.

During resource contention periods in the MPLS network, the priorities determine the treatment of an LSP. *Multiprotocol Label Switching-Traffic Engineering* (MPLS-TE) uses eight LSP priority levels, with zero as highest and seven as lowest, to indicate the importance of certain LSPs over others. This guarantees that less important LSPs can only reserve resources in the absence of important LSPs. It also guarantees that important LSPs are always established along the most optimal path that fits the constraint. Rerouted important LSPs, therefore, have a better chance of finding an alternative path [14].

Resource Reservation Protocol-Traffic Engineering (RSVP-TE) enables MPLS to provide TE capabilities to control the data-forwarding path in the network. With RSVP-TE, MPLS routers can signal an explicitly routed LSP. The exact path and hops for the LSP can be specified end-to-end allowing manipulation of the data traffic paths in the network. MPLS-based TE can arbitrarily split traffic, which is highly flexible for both routing and forwarding optimisation purposes [12].

With MPLS-TE, an LSP can be specified that is different from the normal path taken by the IP packets. The packets on IP-only networks travel from source to destination node using a computed path determined by the individual IP routers. The SPF algorithm determines these paths in each router, and therefore IP-only networks offer little flexibility for providing alternative paths for traffic flow [13].

1.2.1 TE Performance Objectives and Techniques

Table 1.1 summarises previously work and provides a comprehensive list of performance objectives and techniques used to optimise these objectives [9]. The work in this thesis will focus on congestion minimisation through intradomain routing, which affects delay, jitter and packet loss. The problem is formulated as a multicommodity flow problem, to minimise the maximum utilisation of flows assigned to the links. Flow is the bandwidth resources allocated to service demand.

Table 1.1: Traffic engineering performance objectives.

Performance Objective	**Impacts**	**Optimisation Technique**
Congestion Minimisation	Delay, jitter, packet loss	Minimum cost multi-commodity flow problem
End to End (E2E) Delay Minimisation	*Quality of Service* (QoS) and *Quality of Experience* (QoE).	Constrained-Shortest Path First (CSPF)
Packet Loss Minimisation	Congestion	Over-provisioning
Energy Consumption Minimisation	Environmental impact	Adapt the rate of network operation to offered workload. Reduce the number of active devices.
QoE Maximisation	QoS and subjective psychological parameters	Optimisation of network performance.
Resource Utilisation Optimisation	Computation, buffer space, bandwidth.	Schedule well-characterised data transfers, i.e. transmit backup traffic at night. Use bandwidth on demand.

Limitations of TE solutions include the following:

- **Unrealistic traffic splitting ratios:** Leads to excessive packet reordering at the destination resulting from multiple paths is undesirable.

- **Suboptimal path computation algorithms:** Latency inflation results from both the CSPF algorithm continuously computing different paths and the auto-bandwidth algorithm automatically adjusting the reserved LSP bandwidth resulting in continuous path changes.

- **Traffic engineering database does not reflect the real-time network state:** The *Traffic Engineering Database* (TED), used by the *Path Computation Element* (PCE) for path computation does not always reflect the real-time network state.

- **Long convergence times of distributed protocols:** The establishment of paths can take more time because RSVP-TE is used to inform other nodes once the PCE responds with the computed path information.

The above points identified limitations focussed on the current MPLS technology as it is widely used and the limitations it presents in the PCE based architecture. New TE features such as *Border Gateway Patrol-Link State* (BGP-LS), and *Path Communication Element Protocol* (PCEP) extends the TE capability of an MPLS network in terms of network programmability.

1.3 Evolutions in MPLS Technology

1.3.1 Software-Defined Networking Extensions

In the conventional networks, specialised algorithms are implemented on dedicated devices to control and monitor the flow of data in the network and to manage the routing paths and determine the interconnection of different network devices. In general, dedicated hardware such as *Application Specific Integrated Circuits* (ASICs) is used to implement such rules and algorithms [15].

Each router analyses the header of the packet while running a network layer routing algorithm [3]. This operation takes place using routers that are very expensive compared to switches. Under high network traffic, this methodology is severely limited by the network device capability and results in severe limitations on network performance. The network backbone cannot adapt quickly enough to changes in demand without being hugely impacted by processing-intensive hardware or software adjustments [16].

A solution is to implement data handling rules as software modules rather than implementing them in hardware. This idea is realised by a technology called SDN. The concept was originally explored by researchers at Stanford University [17].

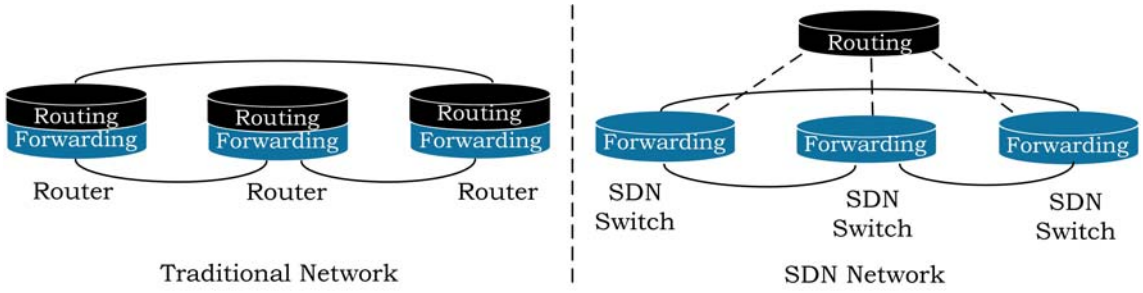

Figure 1.6: Evolution of the traditional network towards an SDN architecture [18].

In the SDN network of Figure 1.6, a traditional network is shown, where the routing and forwarding functions are performed by the same device. In the SDN network, the forwarding function of transferring incoming packets to an outgoing interface has instead dedicated hardware for the forwarding function. The selection of the outgoing interface pointing toward the best path which is determined by the routing function is no longer performed by the router, but by the SDN controller [18].

The data plane consists of a set of one or more network elements each containing a set of traffic forwarding processing capabilities as abstractions of underlying physical capabilities. Network elements in the data plane expose their resources to the SDN controller in the control plane [19]. SDN aims to provide open interfaces, supporting the development of software that can control network element connectivity and the flow of network traffic through these network elements [19].

1.3.2 Hybrid SDN Deployment

Networking paradigms can be divided into three categories namely traditional networks, hybrid networks and pure SDN as shown in Figure 1.7 [15].

The full implementation of SDN, where the network control layer is decoupled and separated from the forwarding layer has not yet received widespread adoption by service providers. It has been deployed mainly on university campus networks and data centres. The transition from the traditional network to an SDN implementation is called hybrid SDN. A hybrid approach supports both distributed and centralised control planes ([24],[29]).

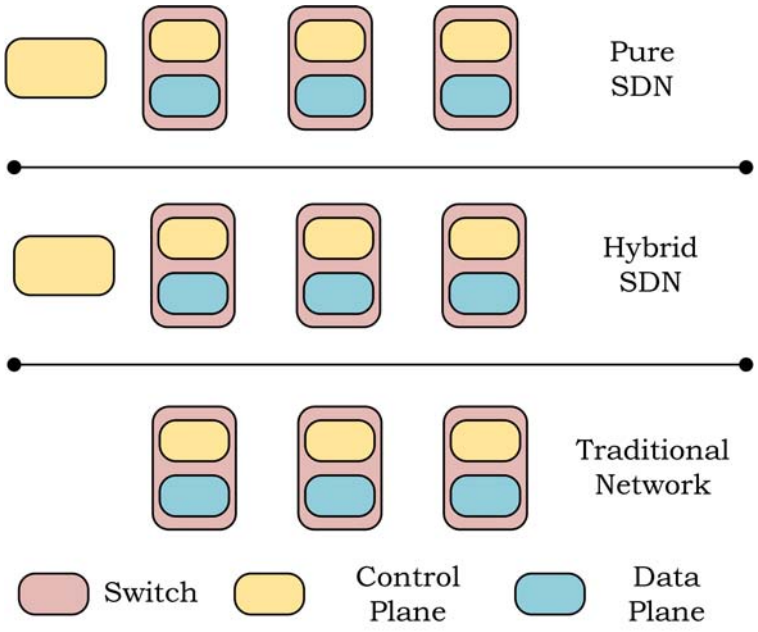

Figure 1.7: Networking paradigm categories.

Partial SDN deployment named HSDN refers to the implementation of the MPLS network capabilities while using an SDN controller for its orchestration capabilities. Since the network operators are reluctant to migrate to full SDN due to cost concerns and technology constraints, the HSDN would be implemented progressively ([21]–[23]).

SDN switches are introduced into the network alongside traditional MPLS switches at strategic points. SDN switch placement planning becomes a very important feature of network design. The goal is to develop solutions that can be deployed in the existing network while offering SDN advantages, such as reduced network complexity and centralised control ([21]–[23]).

HSDN combines the flexibility of SDN with the robustness of OSPF and IS-IS. MPLS data forwarding capabilities with SDN's orchestration capabilities bring multiple advantages such as the routing optimisation of traffic and improvement of traffic-engineering performance [24].

HSDN retains the existing distributed router control planes but offers new *Application Programming Interfaces* (APIs) that allow bidirectional interaction of the applications with the network.

The controller acts as a broker between the application and the network element [25]. HSDN, shown in Figure 1.8, can be implemented using the SDN controllers' southbound PCE and *Border Gateway Patrol* (BGP) interface [26].

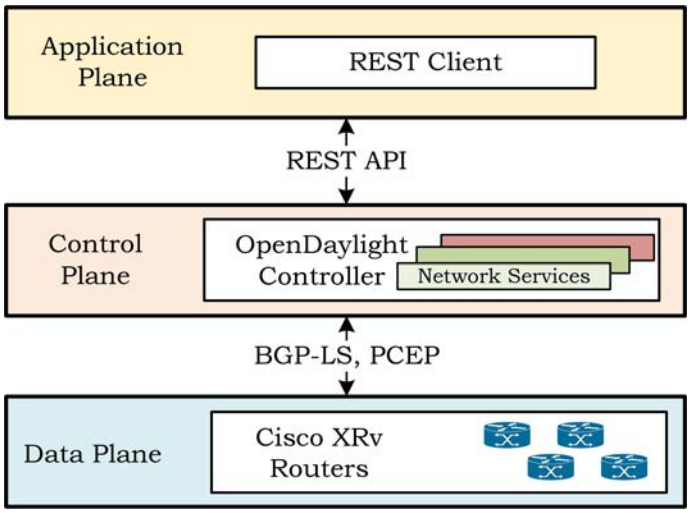

Figure 1.8: Simplified view of a hybrid SDN network.

A *Path Communications Client* (PCC) on the MPLS node would then establish communications with the controller using the PCEP [24].

OSPF or IS-IS with the TE extensions will still be responsible for populating the local routing tables on the MPLS routers, while BGP shares the routing table information with the BGP application programming interface of the controller [24].

1.3.3 Path Computation in HSDN

The PCE architecture comprises a PCE server, a PCC and to support communication between server and client the PCEP protocol. Given a network graph, a PCE is an application that is capable of computing a path through the network while observing computational constraints. A PCC is a client application, usually located in access routers which initiate a path computation request to be performed by the PCE. The PCE modules keep synchronisation with the network topology, link capacities and established LSPs [26].

A PCE supports requests from a PCC network node for path computation [27]. RFC4655 defines the model for a PCE for computing MPLS based *Traffic Engineering Label Switched Paths* (TE LSPs) [28]. A PCE operates on a network graph as input information and calculates a network path while applying the imposed computational constraints. The PCE uses the TED to compute a TE LSP path. The TED holds information such as bandwidth and applies the constraint to the TE LSP service request [24].

PCE responds with the computed paths or the reason for a failed computation in case the demand exceeds the network capabilities. The TED collects information about the network topology and current TE information. The TED is maintained through routing protocols such is OSPF-TE and IS-IS-TE [24].

There are various PCE architectures as shown in Table 1.2 that can be used in either single or multiple domains [29]. Path computation applies to intra-domain, inter-domain and inter-layer contexts.

Table 1.2: PCE architectures for various network domains.

Architecture	**PCE Application**	**Domains**
Single PCE path computation	Single PCE is used	One domain
Multiple PCE path computations	Multiple PCE's	One domain
Centralised PCE computational model	Single centralised PCE	One or more domains
Distributed PCE computational model	Multiple PCE's	One or more domains

Computing an LSP can be done by a single PCE or multiple PCEs. For the single PCE case, the full end-to-end path is computed. When more than one PCE is involved in the path computation, each PCE computes a path segment resulting in a collaborative end-to-end path calculation, that is made up of the individual path segments calculated by the individual PCEs [24].

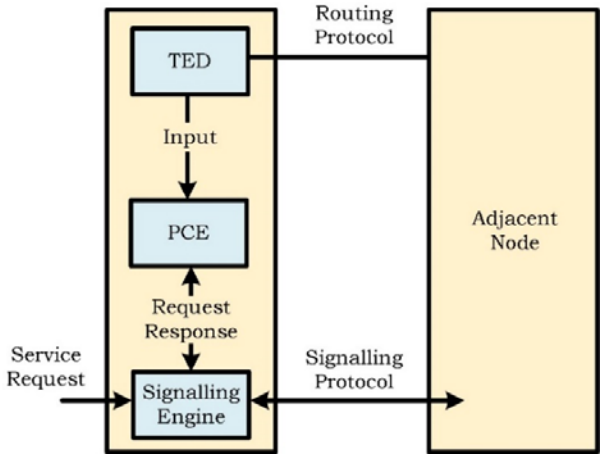

Figure 1.9: Composite PCE node.

Figure 1.9 shows the architecture of a composite PCE node for the typical case where a router implements the PCE functionality for path computation [28]. The PCE can be co-located in the network element, or it can be a separate dedicated device. The former is known as an internal PCE with the latter an external PCE. A PCE can be implemented on a router, an LSR or as a dedicated network server.

The routing protocol exchanges TE information to populate the TED. TE LSP provisioning service requests are received by the node and converted into signalling requests.

Figure 1.10 shows the architecture of an external PCE to the requesting network element [28]. The service request is received via the head-end node, which in turn requests the external PCE for path computation.

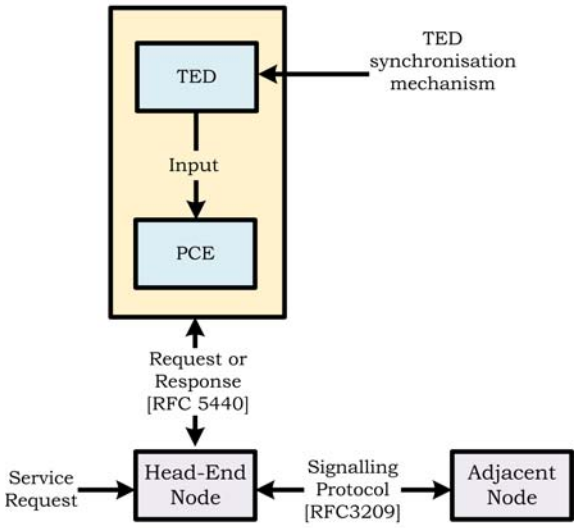

Figure 1.10: External PCE node.

The PCEP enables communications between PCC and PCE. The RFC5440 specifies the PCEP for PCC to PCE communications [30]. The protocol consists of a client-server interaction between PCC and PCE. PCEP is designed to be flexible and extensible to allow for additional future message object should it be required. PCEP operates over *Transmission Control Protocol* (TCP) which provides reliable messaging and flow control.

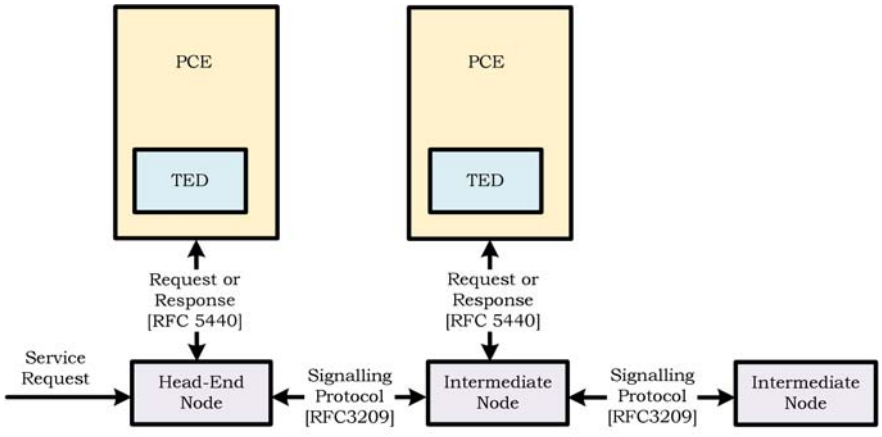

Figure 1.11: Multiple PCE path computations.

For the architecture in Figure 1.11, the head-end PCC requests an external PCE for path calculation [28]. In this case, the path returned is a partial or loose path that requires further computation by the next PCE.

Each PCC must know the location of the PCE. PCC to PCE interaction is done through the exchange of PCEP messages over TCP/IP for reliable communications. Path computation is performed with the PCC opening a session with the PCE within a TCP session [24].

The PCC requests a path computation from the PCE, and the PCE responds to the PCC with the path that has been computed via the PCEP communications path ([35, 36]).

A stateful PCE is aware of both the network state and the already computed paths and reserved resources in the network while a stateless PCE knows the network state only. In a stateless PCE, each request is processed independently without considering resource allocation of the previous request. This results in a much simpler path computation [28].

By further introducing inter-PCE communications, Figure 1.12, a PCE consulted by the head-end node can request computational help directly from another PCE [28]. In the end, a full path for the requested service is returned to the head-end node.

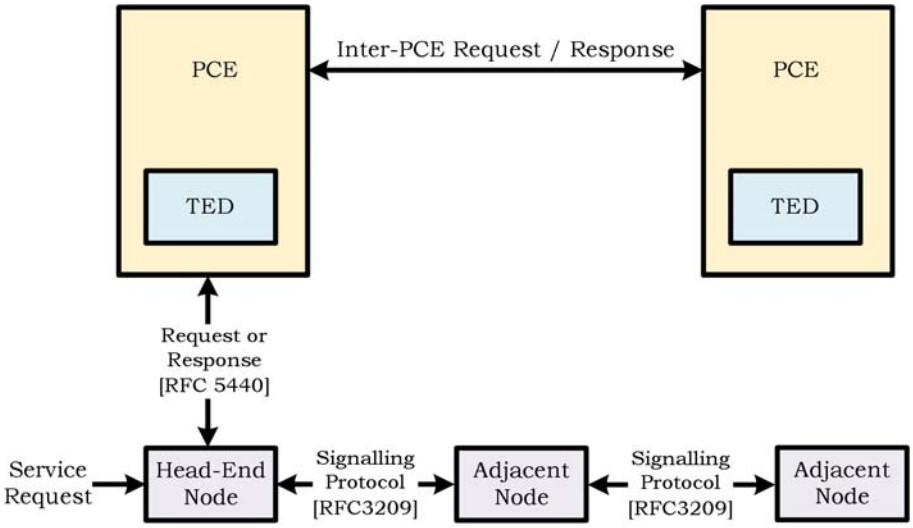

Figure 1.12: Multiple PCE with inter-PCE communications.

1.3.4 BGP Link-State Routing Protocol

From RFC7752, the BGP-LS protocol is used to collect and share information about the link-state and TE using a set of extensions added to the BGP routing protocol [31]. The attribute is used to carry link, node and prefix parameters and attributes and is described as a set of *Type/Length/Value* (TLV) triplets.

The *Link-state Database* (LSDB) and IGP's TED describes the links and nodes within an IGP area. PCE needs to gather information about the topologies and capabilities of the network to perform the requested calculations. Link-state and TE information can be collected from the network with an external component using the BGP routing protocol.

Figure 1.13 shows the distribution of link-state information to consumers [40]. A BGP speaker exchanges the network reachability information with other BGP speakers, including the intermediate ASs that the traffic must transit to reach the destinations [9].

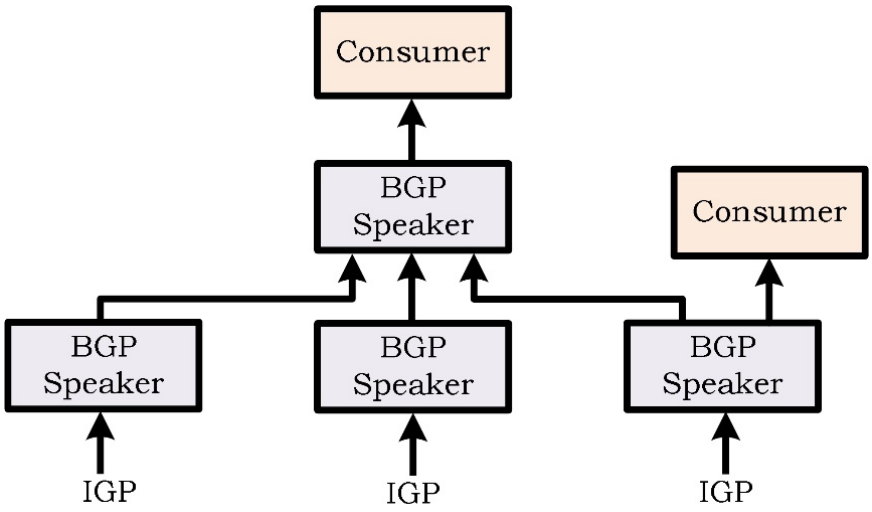

Figure 1.13: Collection of link-state and TE information.

Through IGPs, routers maintain the link-state information about nodes and links for a given area in their LSDBs. The BGP process then retrieves this topology information from the LSDBs and distribute it to consumers directly or via a BGP peer speaker.

The extension from BGP to BGP-LS proposed by the IETF are [31]:

- Local and remote IP address
- Local and remote interface identifier
- Link and TE metric
- Link bandwidth
- Reservable bandwidth
- Per *Class-of-Service* (CoS) class reservation state
- Pre-emption
- Shared risk link groups

PCE offers enhanced computational power that may not be available to individual routers. Routers cannot compute the MPLS-TE path across IGP areas since the router TED lacks visibility of the complete topology. PCE is a computational server that can have visibility into more than one IGP area. This can be achieved through cooperation with other PCEs [31].

A PCE architecture survey is provided by [24]. PCE architectures are categorised into four groups namely single-domain, multi-layer, multi-domain and multi-carrier. A single administrative entity owns and controls single domain networks.

For the single-domain case, the routing protocol can exchange the entire network topology without any restrictions. There are benefits even for the single domain case in that complex path computation algorithms satisfying multiple constraints as well as global re-optimisation of provisioned paths can now be performed by the PCE [24].

In multi-domain networks, the flooding of TE information is inhibited because of control plane scalability and confidentiality considerations across domains in the multi-carrier case. Due to scalability problems, multi-domain routing protocols only exchange reachability information without detailing network resources [24]. Figure 1.14 shows how the PCE populates its own TED using the BGP speaker. The TED information is distributed to the PCE via BGP [28].

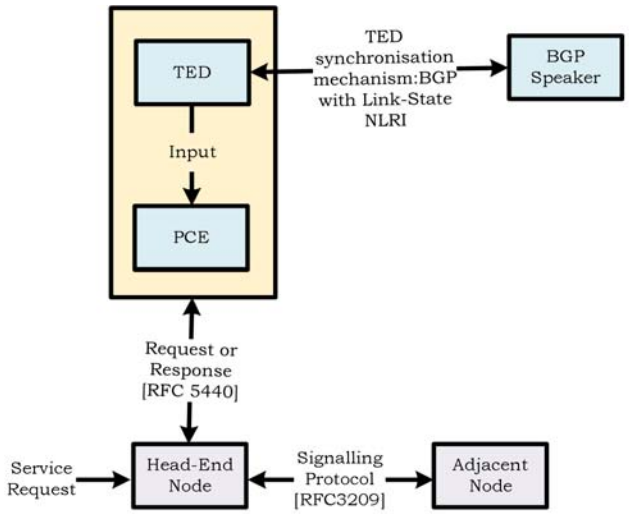

Figure 1.14: External PCE using a TED synchronisation mechanism.

Using the PCEP, the PCE sends explicit paths to the PCCs specified using *Explicit Route Objects* (ERO). The ERO's are used for the establishment of LSPs through RSVP-TE in MPLS and GMPLS networks.

MPLS provides the TE capability to route traffic flows along explicit routes. Routing protocols such as OSPF-TE collect topology information allowing source nodes to perform path computations subject to additional QoS constraints. The path computation process is a crucial TE step to optimise resource utilisation [24].

Heavier processing is required when computing paths based on multiple constraints. Restricted topology visibility becomes prominent when moving from a single-domain single-layer network to multi-domain multi-layer networks due to scalability reasons [24]. The PCE collects link-state information and provides the additional advantage of unloading the CPU-intensive path computation process from the network node to the PCE [24].

1.4 SDN Controller Architecture

Several SDN control plane implementations have been developed such as [32]:

- NOX (C++/Python by Nicira)
- POX (Python by Nicira)
- Maestro (Java by Rice University)

- Beacon (Java by Standford University)
- Floodlight (Java by BigSwitch) [15] and
- OpenDaylight (Java by OpenDaylight project).

These are but a few of the available SDN architecture implementations that are all network operating systems having a single controller [20]. Several surveys ([15], [33], [34]) give full details and comparisons of the differences between these controller implementations.

The controller provides a northbound interface enabling the development of network management applications and a southbound interface to give access to network devices through a vendor-independent protocol ([35], [36]).

The SDN controller, shown in Figure 1.15 interacts with three layers through open interfaces ([20], [36]).

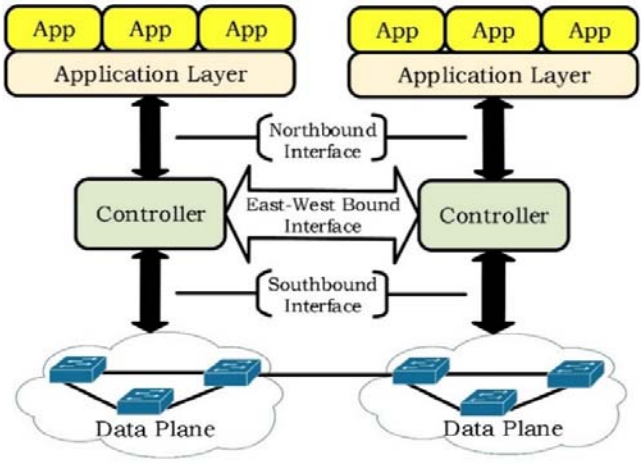

Figure 1.15: SDN interfaces.

These three layers can be described as follows:

- **Infrastructure Layer**: The data plane consists mainly of traffic forwarding elements.
- **Control Layer**: Consists of software-based SDN controller implementations. This allows for control functionality through open APIs. Four communications interfaces are identified namely [37]:

  1. *Southbound protocols:* Allows the controller to interact with the forwarding elements in the infrastructure layer. Examples are OpenFlow and NetConf.
  2. *Northbound Application Interface:* Enables programmability of the controller by exposing the network abstraction data model and other functionality within the controller to the applications at the application layer.

3. *Eastbound protocols:* Allow interconnection of conventional IP networks with SDN networks. Acts as a translation module between SDN and legacy technology. One such example is the PCE module for the MPLS case.[37]

4. *Westbound protocols:* To enable communications between multiple SDN control planes of different SDN domains. Help controllers to synchronise the state for high availability.

- **Application Layer:** These applications consume SDN communications and network services. SDN applications interact with the controllers to achieve specific network functionality and operational requirements. These applications can operate in functional domains such as QoS, TE, *universal access control lists* (U-ACL) management and load balancing.

The principal drivers behind SDN are to separate the control plane from the data plane and to centralise network intelligence and state. In doing so, it provides programmability of network devices. The decoupling of the control plane from the data plane provides a simpler programmable environment and the capability for external software to define the behaviour of a network ([25], [47]).

The decoupling of forwarding hardware from the control plane simplifies network management and enables innovation and evolution. The intelligence is logically centralised in software-based controllers known as the control plane, and network devices, known as the data plane, become simple packet forwarding devices [34].

SDN has been reported as one of the most disruptive and interesting technologies in networking. Manufacturers like Cisco, HP and NEC and novel manufacturers like Corsa are commercialising SDN products [9].

The management plane can be seen as the control platform that accommodates traditional network management services and protocols such as *Simple Network Management Protocol* (SNMP), BGP, PCEP and *Network Configuration Protocol* (NETCONF) ([48], [34]).

1.5 QoS Routing in HSDN

1.5.1 Distributed Systems

In distributed systems routing is determined by the individual control modules in individual IP routers. Distributed algorithms have been studied by various authors. In [39] an algorithm is defined to establish routing tables in individual nodes of a network. The algorithm works well with slowly changing network input statistics where nodes fail occasionally, and link changes are occasional.

Gallager [39] cites that the usual approach to solving this problem would be to have a central node that receives periodic updates from the network and then solve the current routing problem. He further cites that such a strategy is not simple in that routing protocols and control node protocols for executing its decisions were required at the time.

In [40] it is presented a then-new, distributed algorithm for dynamically determining the weighted shortest path in a network. The algorithm focussed on obtaining loop-free paths when link weights change. In [41], Vutukury and Garcia-Luna-Aceves present a distributed, loop-free, shortest path, multipath routing algorithm that can be used for traffic load-balancing for minimising congestion and delays. The work in [42] presents a method for bandwidth reallocation in a path-oriented network without a centralised control system coordinator.

The emergence of SDN has shifted the research focus from distributed systems to centralised network control and routing computation. Centralised network control, as well as the execution of QoS routing algorithms, are achievable because of protocols such is PCEP, BGP-TE, IS-IS-TE and OSPF-TE [43].

1.5.2 Centralised Systems

In and SDN centralised systems, the routing process is carried out by a centralised controller. Here the emphasis is placed on a logically centralised model which does not suggest that the system is physically centralised. Production SDN design resort to physically distributed control planes [44] with a logically centralise TE implementation described in [45]. In a comparative controller study done in [25], it can be observed that most controllers are centralised and multithreaded. Centralised control simplifies the management since control logic can be constructed from a complete view of the network. Such a system decreases convergence time and coordination complexity. Having a complete view of the network facilitates routing decisions, traffic engineering and fault localisation [35].

1.5.3 Transmission Paths in IP Networks

Multicast routing involves finding routes from a source node to multiple destination nodes. In contrast, this study focus on unicast routing, from a single source node to a single destination node. While unicast routing is concerned with solving the problem of finding a route between a single source node and destination node resulting in a single path, multipath routing involves finding multiple parallel paths between the single source and destination node as would be deployed in this research.

1.5.4 Routing Paths

1.5.4.1 Unicast Single Path Routing

Single path routing algorithms are not optimal in responding to congestion and a temporary traffic burst in a network. Routing algorithms like *Routing Information Protocol* (RIP) and *Enhanced Interior Gateway Routing Protocol* (EIGRP) are not optimal since these algorithms provide only a single path between the source and destination node pairs. OSPF allows multiple paths only when multiple paths of minimum cost exist [41].

1.5.4.2 Unicast Multipath Routing Based on PCE Architecture

In [46] the authors provide a standardised view of the SDN controller interfaces, navigating the multitude of ideas and concepts that make up the SDN technology.

Four basic principles of SDN is defined, namely:

- Open interfaces
- Programmability
- Separation of control and data plane
- Logically centralised control

SDN consolidates the control plane so that a single programme control multiple data-plane devices. The open interface allows the non-SDN domain to be able to react to controller inputs via the PCEP messages. PCEP is a protocol between the *Path* PCE and the PCC. The PCE will be the southbound interface, and the PCC will be the client implemented on the MPLS equipment side [47].

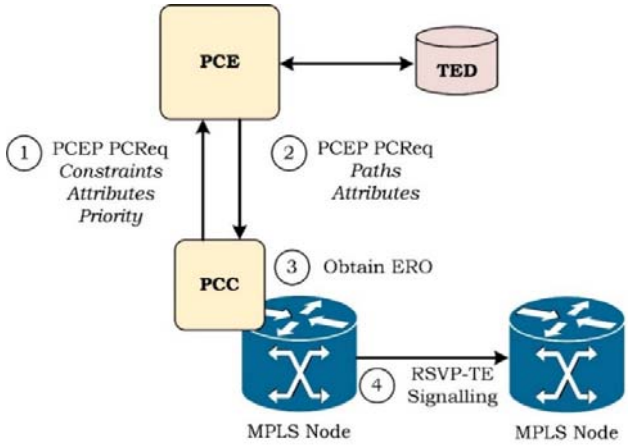

Figure 1.16: PCE architecture using PCEP to communicate with PCC.

In Figure 1.16 a PCC requests a path. The PCE then computes a path using the TED. PCEP and RSVP-TE are then used to fulfil the request. PCEP governs communications between the PCC and PCE [9].

Using the PCEP the legacy MPLS domain now can react to *Create Retrieve Update Delete* (CRUD) instructions from the controller on the LSP. Also, the legacy MPLS domain is now enabled for programmable control.

1.5.5 Multipath Routing in MPLS Networks

1.5.5.1 Induced MPLS Graph

An induced MPLS graph maps logically onto the physical network through the selection of the LSPs for traffic trunks. The LSRs comprises the set of nodes of the graph, with a set of LSPs providing logical point-to-point connection between the LSRs. The induced MPLS graph is important since the basic problem of bandwidth management in an MPLS domain revolves around the efficient mapping of the induced MPLS graph onto the physical network topology [7].

The induced MPLS graph shown in Figure 1.17, leads to the following formalised abstraction. Let $G = (V, E, C)$ be a capacitated graph representing the physical network topology where V and E are the set of nodes and links respectively and C the link capacity set.

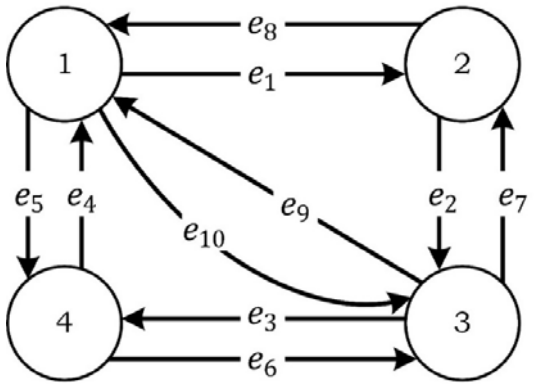

Figure 1.17: Induced graph.

1.5.5.2 Bidirectional Traffic Trunk

The union of two unidirectional trunks form a bidirectional trunk. The two logically coupled trunks carry traffic in both directions, one in the forward direction and the other in the backward direction. Trunks are bidirectional if both traffic trunks are instantiated through one action at one LSR. Neither composite traffic trunks can exist without the other. Both are instantiated and destroyed at the same time [4].

1.5.5.3 Pre-Emption Attribute

The attribute assures that high priority traffic trunks are routed through favourable paths within a differentiated services environment. The attribute determines whether a traffic trunk can pre-empt another traffic trunk from a given path. When LSPs need to reroute, important LSPs have a better chance of finding an alternate path.

1.6 Virtual Private Networks

1.6.1 Hose Model

In the context of VPNs, there are two popular models for providing QoS, namely the Pipe- and Hose model ([48],[49]). The Hose model provides advantages to a VPN customer in that traffic can be sent to a set of endpoints without specifying the detailed traffic matrix. It furthermore provides a reduction in the size of the access links through multiplexing gains, obtained through the aggregation of flows between endpoints ([50],[51]).

In contrast, a complete traffic matrix is required for the Pipe model detailing the load between every pair of endpoints. With the number of endpoints per VPN constantly increasing and communication patterns between endpoints increasingly unpredictable, knowing the complete traffic matrix becomes almost impossible.

1.6.2 VPN Routing Structure

VPN customers specify QoS requirements per VPN endpoint and not every pair of path endpoints. For these endpoints in the Hose model, a pair of bandwidths, ingress- and egress bandwidths need only be specified. Efficient algorithms provision these hoses, setting up paths between every pair of VPN endpoints such that the aggregate bandwidth reserved on the path is minimised [49].

The cost of the reservation is strongly influenced by the routing schemes used between the VPN end nodes.

Three common routing schemes for implementing the Hose model are available:

1. ***Tree routing:*** In tree routing, VPN endpoints are connected by a Steiner tree ([49],[51]). All VPN traffic is sent along the unique path in this tree from source to destination.

2. ***Single-path routing:*** For single-path routing, every (u, v) pair of VPN endpoint is assigned a single path γ_{uv} for routing the traffic from u to v, but the union of all paths need not be a tree.

3. ***Multi-path routing:*** In multi-path routing, every pair (u, v) is assigned a collection of paths from u to v along with a specification of which fraction of traffic from u to v that should be sent along the path.

In [52] it is shown that optimal bandwidth reservation can be computed in polynomial time for multi-path routing. The study shows that even for very small networks, multi-path routing reduces the reservation cost as compared to single-path and tree routing. Furthermore, the running time of optimal algorithms for multipath routing is better than tree routing and single-path routing.

In [48] a comparative study is made between tree- and multipath routing regarding blocking performance and bandwidth efficiency using sub-provisioning which is defined as providing reserved bandwidth at a percentage of the full provisioning bandwidth. By using the sub-provisioning method, they show that the blocking performance of multipath routing is much better than tree routing.

VPN multipath routing splits the traffic between a (u, v) pair among multiple paths. In the event of a single path failure, the pair (u, v) will not be disconnected. Furthermore, there are known polynomial algorithms for optimal routing computation in general cases [48].

1.6.3 VPN Based on Hose Model

In Figure 1.18, the customer endpoints aggregate on the link between the *Provider Edge* (PE) *Label Edge Router* (LER) and the LSR.

For each VPN endpoint j, the maximum total bandwidth $b^+(j)$ of traffic that j will send into the network at any time is specified along with the maximum total

bandwidth $b^-(j)$ of traffic that j will ever receive from the network at any time. Provisioning bandwidth based on $b^+(j)$ and $b^-(j)$ values allow all traffic not violating $b^+(j)$ and $b^-(j)$ values to be accepted. This type of guarantee can only be met if the bandwidth reserved is based on the worst-case traffic split.

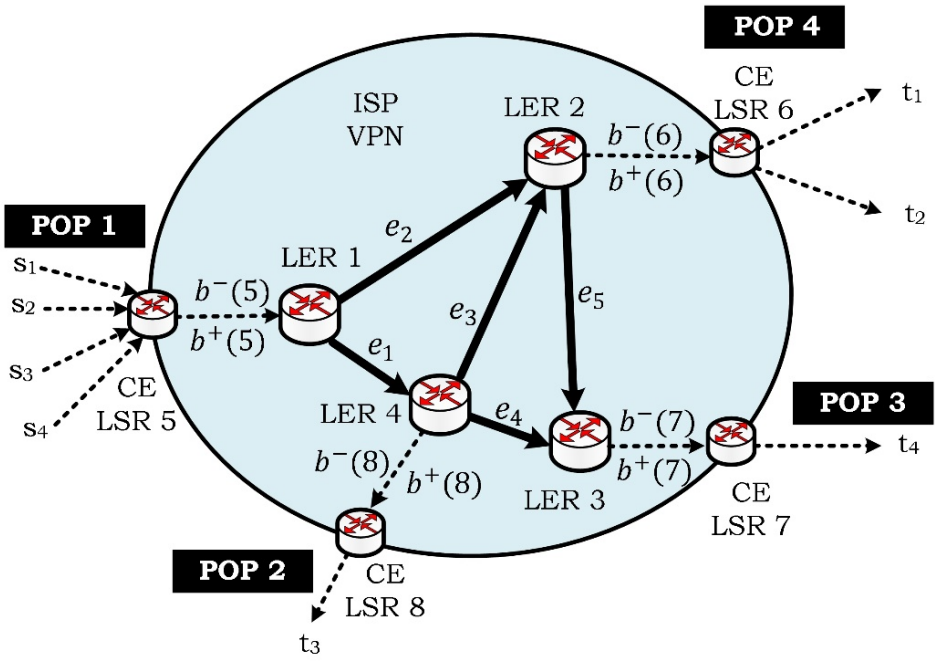

Figure 1.18: VPN based on hose model.

This would typically be the case where traffic from each hose in the network is directed to one endpoint. The Hose model requires at least twice as much bandwidth than for tree routing. Worse case traffic split is rare, resulting in network resources being mostly underutilised.

Work done in ([48],[50]) is an attempt to address the overprovisioning problem. Based on the online measurement and traffic prediction, the bandwidth of internal network links is dynamically changed to reduce the amount of overprovisioning.

At the time, dynamic resizing was complex, unproven and not as reliable as the static provisioning. Sub-provision is another method used where only a certain percentage of the full provisioned bandwidth is reserved. Full provisioning would be to provide the bandwidth according to $b^+(j)$ and $b^-(j)$ bounds.

1.7 Timers Based Bandwidth Management

An important aspect of the routing protocols is the convergence. When there are changes in the network, nodes will have an inconsistent view of the network and need to update their view of the network to arrive at some consistency level. This process is referred to as routing convergence. Timers are activated for routing protocols to update their local database to achieve convergence.

1.7.1 BGP Timers

BGP is used to communicate information about networks in an autonomous system, to other autonomous systems. The network here refers to an IP prefix defined network where IP prefix has the form *A.B.C.D/n*, with */n* referring to the network mask. A TCP session is set up between bordering ASs using port number 179. As long as the session is active, both sides can exchange periodic network information updates. Several BPG timers are defined for the BGP *Finite State Machine* (FSM). BGP has five mandatory timers and two optional timers, where a time parameter is defined for each timer [53].

Table 1.3: BGP timers.

BGP Timers State	**Value is Seconds**
Connect Retry	120
Hold	90/240 (Cisco: 180)
Keep-Alive	0.3333 * Hold (Cisco: 60)
Min Route Advertisement Interval	30 EBGP, 5 IBGP
Min AS Origination Interval	15

The time values in Table 1.3, indicate the initial value of the mandatory timers. Of these timers, the hold and keep alive timers are the primary BGP session timers. The other two optional timers not shown in the table are “delay open” and “idle hold” timers.

1.7.2 OSPF Timers

Table 1.4: OSPF timers.

OSPF Timers State	**Value in Seconds**
Hello Interval	10
Poll Interval	120
LSA retransmission Interval	5
Dead Interval	40
Transit Delay	1

The default OSPF timers, shown in Table 1.4 [54] settings are optimal for most networks. The timers may, however, be changed depending on network requirements. A description of these timers can be found in [55].

1.8 Related Work by Other Researchers

MPLS traffic engineering lacks the global view of the network whereas the *Software-Defined Network* (SDN) has a global network view [56]. This feature along with network programmability makes *Hybrid Software Defined Networking* (HSDN) a very attractive topology to implement TE features. This section reviews TE approaches and is a look at some of the earlier work as well as more recent work done to solve TE problems related to HSDN.

1.8.1 TE-Related Review on Hybrid SDN

Table 1.5 is a summary of work done in the MPLS research area where an SDN architecture has been implemented in a hybrid context. The hybrid context, known as hybrid SDN, is an architecture where the legacy MPLS network is extended by implementing an SDN controller in the MPLS network.

Table 1.5: TE-related review on hybrid SDN.

Feature	Description	Network Type	Optimisation
Load Balancing for Multiple Traffic Matrices Using SDN Hybrid Routing [57].	The objective is to load balance traffic on all links. Can respond to sudden traffic changes. Changing traffic is represented by multiple matrices. A hybrid of explicit- and destination-based routing is used. Explicit based routing relies on *Ternary Content Addressable Memory* (TCAM), which is expensive per Mb and in power requirements. Hybrid routing relieves the TCAM requirements using a blend of explicit and destination-based.	SDN using controller	Yes –Heuristic
Traffic Engineering in SDN/OSPF Hybrid Network [56].	The objective is to minimise the maximum link utilisation of the network by adjusting the OSPF weight settings of the entire network. Also how to split flows that aggregate at the SDN nodes so that max link utilisation is minimised. Work is original in doing TE in the hybrid network scenario in that the OSPF weights and network splitting ratio of the SDN nodes can both be changed at the same time.	SDN/OSPF hybrid network Testbed**:** Simulation experiments	Yes - Heuristic. Weight optimisation.
Optimal Load Balancing Method for Symmetrically Routed Hybrid SDN Networks [58].	Load balancing method that optimises link cost and path selection simultaneously under symmetrically routed condition. With symmetrically routed flows, the same path is used for flows in both forward and reverse direction. The method minimises the Maximum Link Utilisation by considering link cost and path selection probability as simultaneous equations.	SDN/OSPF Hybrid Network Testbed**:** Simulation experiment	Yes Min MLU
Highly Available SDN Control of Flexi-Grid Networks With NFV-Enabled Replication. [59]	*Network Function Virtualisation* (NFV) relates to the dynamic deployment and operation of common network functions stored and executed in virtual computing instances running on commodity hardware and deals with the virtualisation of PCE functions. Shows how PCEP and BGP-LS protocols are used to achieve synchronisation.	NFV of PCE functionality. Testbed**:** Virtualised environment	No NFV architecture is described.
Active Stateful PCE high-availability for the control of Flexi-grid Networks with NFV enabled replication.[60]	Similar architecture as [59], but here the focus in on the synchronisation of the distributed databases. TED and *Label Switched Path Database* (LSPDB) must be synchronised when PCE replicas are dynamically instantiated. The synchronisation is achieved through PCEP and BGP-LS.	NFV of PCE functionality. Testbed**:** Emulated hardware. Control Plane	No NFV architecture is described.
Multipartner Demonstration of BGP-LS-Enabled multidomain EON control and instantiation with H-PCE [61].	An extended *Hierarchical Path Computation Element* (H-PCE) architecture is considered to control multi-domain *Elastic Optical Network* (EON) based on GMPLS. Inter-domain EON lightpath provisioning enabled by OSPF-TE, BGP-LS, PCEP and RSVP-TE protocols. The architecture is evaluated in a multi-partner control plane testbed with multi-platform and multi-vendor network controllers. Stateless and stateful H-PCE are considered.	EON network with H-PCE. Testbed**:** Multipartner network of four European research institutions.	No. Orchestration architecture is described and tested.
Analysis of PCE-based path optimisation in multi-domain SDN/ MPLS/BGP-LS network. [26]	Reports on an investigation with PCE-based path computation employing TE. The PCE module is equipped with the IBM CPLEX LP solver. The model describes the minimisation of the total cost of flows deployed within the network.	PCE module with IBM CPLEX LP solver. Testbed**:** Virtual routers, PCEP and SDN.	Yes The minimisation of the total cost

1.8.2 Traffic Engineering on MPLS networks

Review of TE approaches used on MPLS networks.

A comparative TE study by Sengezer *et al.* [62] between single-layer and multi-layer TE approaches is described. Single-layer TE is described as TE interventions where only the upper layer is affected, with no changes to the underlying logical layer. When the logical layer is reconfigured in response to a TE intervention, the resulting approach is called multi-layer TE. The study compares three TE approaches;

- MPLS layer TE on a fixed logical topology,
- MPLS multi-layer TE with periodic logical topology updates,
- Label switched path rerouting in the wavelength division multiplexing layer.

The results show that when traffic is unpredictable, then multi-layer TE outperforms single-layer TE. However, single layer TE outperforms both multi-layer approaches when traffic is predictable but requires greater network resources than multi-layer. This finding is useful in that it supports the TE approach used in this research regarding the testbed design. With reduced resources, multi-layer TE with period logical topology updates is shown to be the better approach. Furthermore, in this work, a wavelength division multiplexing layer will not be implemented, due to resource constraints.

Awais *et al.* [63] show that applying TE on an MPLS network shows an improved end to end delay benefit when compared to an MPLS network where no TE is applied. Sridhar *et al.* [64] demonstrate that MPLS TE has an improved performance over IP TE and MPLS in terms of packet size, round trip time, throughput and latency.

Agrawal *et al.* [65] propose a solution by combining proactive and reactive TE algorithms. Proactive algorithms are prediction-based and perform efficiently when traffic is predictable but cannot respond to unpredictable traffic spikes. Reactive algorithms can respond to changes in traffic patterns, but issues of stability and convergence may arise. A robust version of the minimisation problem under demand uncertainty is implemented called the *Robust Routing TE* (RRT) algorithm. The uncertain demand can be static or multihour traffic and is a routing solution to TE for dynamic traffic demand. Though the RRT algorithm performs better than OSPF-TE, the work, however, is implemented in simulation, where the focus appears to be on testing the RRT algorithm rather than in real-world network implementation.

Otoshi *et al.* [66] propose a framework that handles the uncertainty of traffic information inspired by how the human brain makes decisions in uncertain environments. The Bayesian decision-making model by Bitzer. *et al.* [67] that proposes a probabilistic framework influencing human brain decision making, is used to handle this network traffic information uncertainty. The paper, however, illustrates only the usefulness of the algorithm where uncertainty is caused by lacks and lags of network information but leaves the implementation of such a framework as a remaining challenge.

In [68], Taruk *et al.* develop a traffic engineering method that uses the first shortest path constraint algorithm, implemented on the router, to regulate the flow of data in the network aimed at reducing the excessive use of bandwidth in an MPLS model network. The OSPF routing protocol is used. Traffic is moved from high congestion paths to paths with lower congestion levels. The results show a reduction in queue length at each LSR. Very little detail is given on the actual algorithm used. Furthermore, the full implementation is carried out in a simulation, with no detail given on the network implementation. Because of the full simulation implementation, it brings to question how much of such an implementation can be implemented in a real network scenario.

A load-balancing algorithm *Load Balancing Algorithm using Deviation Path* (LBDP) is proposed by Li *et al.* [69], as a method to establish an LSP on an MPLS network that avoid the hotspots. The NS2 simulation results show advantages in bandwidth utilisation, throughput and packet loss over two other algorithms used in the study namely *Load Balance by Sideway Algorithm* (LBAR) and *Shortest Path First* (SPF). LBDP switches the LSP to a newly available path when congestion is detected. The downside of the method is that for traffic bursts, too many switch requests will be generated in response to the short time bursts affecting network convergence time.

The Dijkstra algorithm is used to find the shortest path between source and destination on an MPLS network. By implementing MPLS TE, a modified Dijkstra algorithm as shown by Fang *et al.* [70], where the algorithm presents the next uncongested shortest path instead of denying the path request, the results show improved bandwidth utilisation on the links of the MPLS network when compared to the unmodified Dijkstra algorithm implementation. The implementation, however, only finds the shortest paths through the network and does not implement any optimisation approach.

1.8.3 Traffic Engineering on MPLS networks using SDN

This research focuses on hybrid SDN in an MPLS/OSPF context. This section reviews TE work done on hybrid-SDNs in recent years where OSPF or IS-IS was used as the internal routing protocols.

Legacy MPLS networks have gained high programmability capability as well as the concentration of control functions and openness, through the unified BGP-LS/PCEP interface and the centralised software-defined networking control plane [71]. This largely simplifies network management and enables innovation [72].

Zhang *et al.* [57] achieve load balancing through a combination of destination-based routing complemented with explicit routing, hence hybrid routing. Traffic fluctuation is accounted for through multiple traffic matrices. Here traffic-fluctuations refers to the change in user demand for bandwidth over time. The optimal route is a combination of destination-based routing and explicit routing that accommodates multiple traffic matrices. The authors do not describe the controller architecture but simply states that the controller translates the hybrid routing solution to forwarding entries which are then installed in the routers.

The objective in [56], Guo *et al.*, is to minimise the maximum link utilisation of the network by adjusting the OSPF weight settings of the entire network for flows from regular nodes as well as split flows that aggregate at the SDN nodes, so that max link utilisation is minimised. The work is original in doing TE in the hybrid network scenario in that the OSPF weights and network splitting ratio of the SDN nodes can both be changed at the same time. An *SDN/OSPF Traffic Engineering* (SOTE) algorithm is proposed.

In [58] Yasunaga *et al.*, a load balancing method is described that optimises link cost and path selection simultaneously under symmetrically routed conditions is described. With symmetrically routed flows, the same path is used for flows in both forward and reverse direction. The main contribution of the method is that it minimises the Maximum Link Utilisation by considering link cost and path selection probability as simultaneous equations. Simulation is used to implement the mathematical models, while no description is given of any controller implementation.

An extended *Hierarchical Path Computation Element* (H-PCE) architecture is considered in [61], González de Dios *et al.*, to control multi-domain *Elastic Optical Networks* (EON) based on GMPLS. Inter-domain EON Lightpath provisioning is enabled by OSPF-TE, BGP-LS, PCEP and RSVP-TE protocols. The architecture is evaluated in a multi-partner control plane testbed with multi-platform and multi-vendor network controllers. Stateless and stateful H-PCE are considered. The main focus of this study was to implement and validate H-PCE and did not focus on strategies for optimised path computation throughout the network.

Rzym *et al.* [73] report on an investigation into PCE-based path computation deployment in a hybrid-SDN multi autonomous system architecture. The impact of the number of simultaneous demands on computation time and memory usage was shown. The model describes the minimisation of the total cost of flows deployed within the network and shows that the computation time, and memory usage increase linearly with an increase in simultaneous demands.

The survey by Abbasi *et al.* [72] on classic traffic engineering techniques finds that for optimum TE on the network, the TE implementation must be able to implement multi-path diversity. Routing decisions must also be made based on the global view of the network and network flow must be considered. The architecture used in this research, meet the described characteristics, in that the global view of the network is maintained by the router routing protocols. The implemented optimisation uses multi-paths to distribute the demand throughout the network, while the stateful implementation of the PCE keeps track of the LSPs.

The work by Martinez *et al.* [74] reports on the implementation of an integrated transport SDN (T-SDN) controller. The systems auto-configures degraded optical paths based on an OSNR measurement on each node's outgoing link. The measured OSNR data is analysed and values below an expected threshold value are declared signal-degraded and needs to be restored. This work is different in that the focus is on the optical transport layer and the reconfiguration of degraded optical paths.

Morin *et al.* [75] implemented a STEM (SDN Traffic Engineering Management) module, to provide on the fly bandwidth allocation to users without requiring the operator's intervention. The implementation uses the OpenFlow 1.3 protocol to communicate with the data plane. While OpenFlow 1.3 is an established protocol in a purely SDN environment, it does not form part of the supported protocols on legacy equipment and would require the introduction of new hardware into the network. In the implementation, the authors demonstrated how difficult it is to find off-the-shelf equipment that can support their STEM implementation.

Kadiyala *et al.* [76] seek to find an optimal set of egress routers, nearest to the source, to route inter-domain transit traffic through a hybrid SDN ISP network, such that the maximum link utilisation of the egress links is minimised. This will ease bottlenecks at peering border routers between ISPs. The authors introduce a small number of SDN nodes into the network. This hybrid realisation of SDN, which is the coexistence of legacy equipment with a small number of SDN nodes requires the ISP to introduce new hardware into their network. Furthermore, the implementation is evaluated through simulation on an ISP topology.

Similar to [76], the hybrid SDN implementation of Chen *et al.* [77] also include a limited number of SDN switches replacing some of the legacy switches in the network. While this hybrid SDN implementation certainly saves on cost since only a few of the legacy switches are replaced with commercial SDN devices, there is still an initial prohibitive cost to this type of implementation. Given a fixed amount of SDN switches with which to upgrade a network, the authors demonstrate a solution to solve the hybrid SDN switch placement problem as an optimisation problem.

Husni *et al.* [78] implement an SDN controller application. Shortest path LSP creation, LSP deletion and LSP modification were tested to verify the functional operation of the application with additional performance testing on the time needed to create several LSPs. While there are similarities in the testbed design to the work done in this thesis, the authors did however not implement any algorithm to solve any TE issue.

The *SDN Traffic Engineering Management* (STEM) module, Morin *et al.* [79] creates LSPs based on a path calculated by the PCE. The PCE can take into consideration various constraints while calculating the LSP. However, the LSP is created based on a request and does not respond dynamically to demand. Furthermore, the implementation uses the OpenFlow protocol, which is not supported by legacy equipment, to the extent that the authors had to use an L2/L3 switch that implements the OpenFlow protocol along with the traditional switching functionality. The solution does not fit well with legacy equipment of ISPs and will require additional hardware for a real-world implementation.

Sinha *et al.* [80] demonstrate a prototype hybrid SDN implementation. The model uses a combination of MPLS and OpenFlow protocols. This hybrid SDN implementation requires SDN compatible nodes at the edges. This approach is costly and requires ISPs to investment into new hardware. Similar network topologies are implemented by Caria *et al.* [81], and Guo *et al.* [82].

Seremet *et al.* [83], does a study on the reconvergence time after a link failure while using BGP-LS and PCEP as the southbound protocols. Timers of the BGP protocol is adjusted to minimise the time needed for complete reconvergence.

1.9 Concluding Remarks

MPLS has been introduced as a ubiquitously deployed networking technology with widespread VPN architectural implementations. The adoption of the IP/MPLS virtual private network technology by service providers for backbone networks has enabled multiple service-oriented network deployments.

MPLS-TE allows the placement of LSPs that is different from the path calculated by the IGP SPF algorithm that a packet would usually take to reach its destination.

The introduction of SDN provides programmability of the network devices. The decoupling of the control plane from the data plane provides a simpler programmable environment, where the external software influences the behaviour of the network.

The following motivations are drawn from the literature review:

- The objective of extending networking protocols is to:
    1. Be current and is broadly deployed by internet service providers.
    2. Have the capability to create, read, update and delete paths.
    3. Track the resources and resource utilisation.
- MPLS meets all these requirements. The choice of MPLS as technology is further supported by a comparative study performed by Mahesh *et al.* in [84], where conventional routing as compared to MPLS routing it was found that MPLS routing outperformed IP routing in bandwidth utilisation.
- VPN provides a cost-effective service delivery mechanism and has seen widespread adoption by service providers for use in backbone networks. IP/MPLS VPN architecture combined with the advantages inherent in SDN allows the automatic bandwidth adjustment during changing traffic demands.
- The benefits of MPLS includes scalability, effective management and it is standards-based. Compared to IP based networks, MPLS scales asynchronous transfer mode and virtual private networks very well [2].
- Benefits of MPLS are extended through the new SDN networking paradigm, with some very useful features such as:
    1. Separation of the control plane.
    2. The centralised architecture provides a global network view. The controller stores the entire network information including the network topology and dynamic changes of network status.

3. Network programmability of the data forwarding layer devices allows the optimisation of network resources allocation.

- HSDN and PCE allow the implementation of an optimisation algorithm in applications external to the networking environment.

- External PCE node is of importance in this research because it enables a single PCE implementation for the IP/MPLS virtual private network architecture.

- Single domain PCE architecture provides all the topological information needed without applying any policies as it is the case for multi-domain.

- PCE architecture can compute label switched paths separately from the actual packet forwarding in MPLS networks.

- Multipath routing is superior to single-path and tree-path routing. LSPs are therefore configured along multi-paths to accommodate the demand for bandwidth.

- SDN centralised controller and the BGP application programming interface allows a global network view. The BGP speaker exchanges the network reachability information with the controller.

- SDN controller performs an intermediary function between the network on the southbound side and the network application which interacts with the controller via the *Representation State Transfer* (REST) interface.

- Through PCEP, the network has a mechanism by which CRUD actions can be performed allowing the optimised decisions to be fed back into the network.

- BPG-LS along with interior gateway protocols such as OSPF-TE and IS-IS-TE tracks network changes and broadcasts all local TED information to the BGP-LS database of the path computation element.

- The induced MPLS graph maps logically onto the physical network and lead to a completely formalised capacitated graph representation $G = (V, E, C)$.

- This graph representation and the formulation of the bandwidth modelling problem to be discussed in chapter 3 provides the environment needed to address the objectives of this research.

- The combined benefits of IP/MPLS, virtual private network and SDN with its BGP-LS and PCEP applications programming interface provide a rich enabling environment to address the bandwidth management requirements of this thesis.

In the next chapter, the research hypothesis, objectives and key research questions are presented.

2 Research Objectives and Plan

An adaptation of the proposed taxonomy of SDN issues and research directions presented in [33] is shown in Figure 2.1. This taxonomy presents a survey of the most relevant research initiatives raised by SDN researchers towards the support and adoption of SDN concepts in current networks.

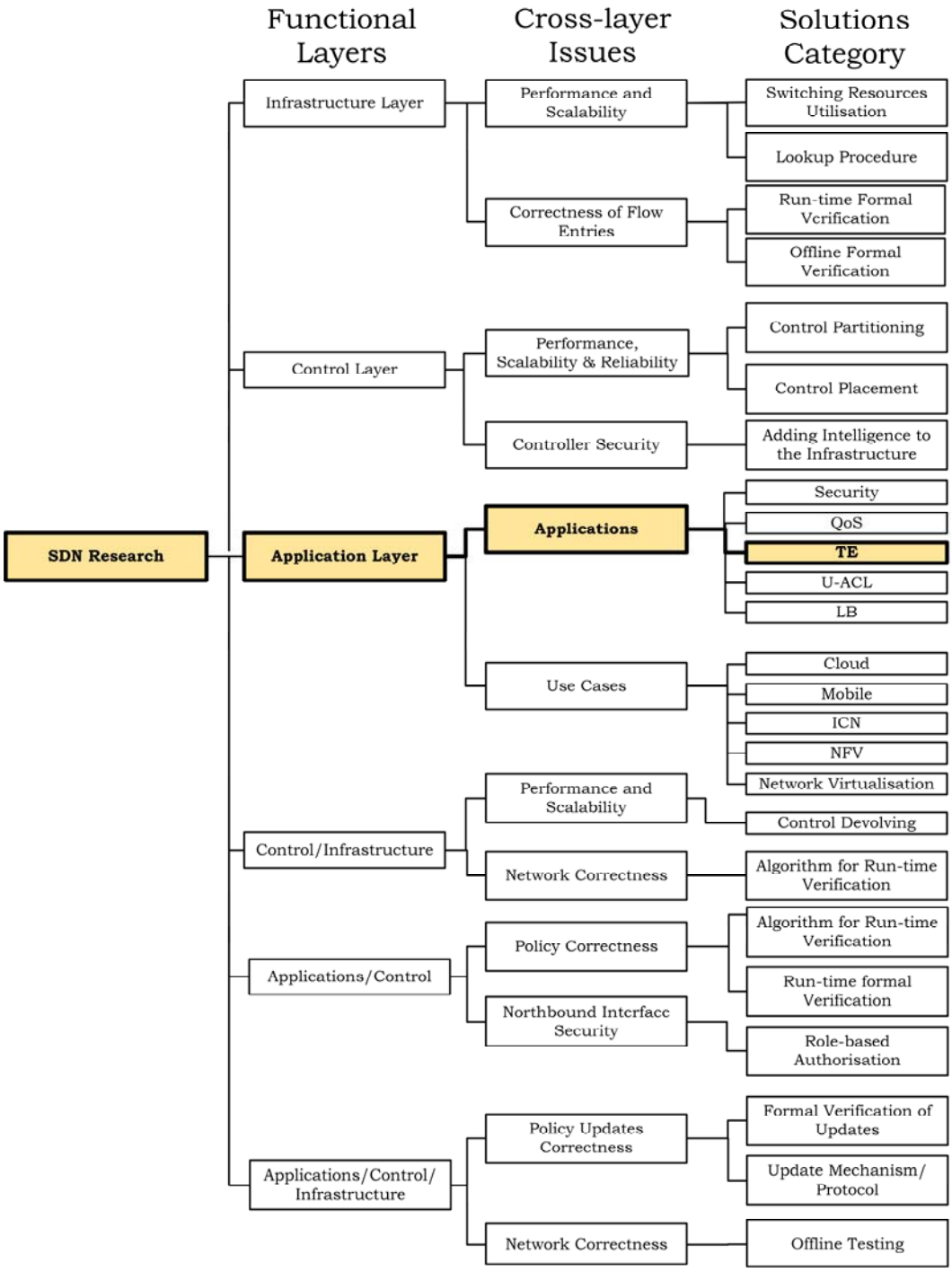

Figure 2.1: Overview of surveyed taxonomy.

In this taxonomy, three broad SDN research categories are identified namely the functional layers, cross-layer issues and solutions. These categories appear as vertical columns in Figure 2.1.

The taxonomy is useful to identify the focus area of this work in the SDN space. The functional layer of interest for this research is the application layer. Associated with each of the functional layers are the cross-layer research issues. The solutions categories represent the research attempting to solve the cross-layer issues.

Of interest is the SDN research related to the application layer, with applications being the identified concern. At this layer, the SDN applications interact with the controller to achieve a networking function based on given objectives.

Applications that are written for this purpose implement network functionality which falls into one of the following categories:

- Security
- QoS
- TE
- Universal access control lists management
- Load balancing

The path highlighted in Figure 2.1, indicates the area of research for the work done in this thesis, with TE being the solutions category.

2.1 Problem Formulation

- Bandwidth consuming devices are on the increase ([86], [87]), with each generation of devices placing bigger loads and strains on the internet service provider data network. The dominating traffic being real-time entertainment from bandwidth-intensive web applications (YouTube, Netflix, Twitch, VoD, etc.), Aouini [85].

- In 3-4 years, technology will place a bigger load on internet service providers, with most applications being online and the demand for video and audio streaming increasing [1]. Internet traffic growth is increasing exponentially every year, with the number of AS increasing linearly by 238.6 per month ([88], [89]).

- MPLS is the technology that will carry these ideas into the future [2], and auto-bandwidth adjustment will mean that bandwidth can be shifted around in the network, in a dynamic way to respond to peaks in traffic demand. MPLS has been widely adopted as a means of providing a Quality of Service to customers on a common platform ([93],[94]).

2.2 Research Motivation

- The proposition for this work is based on the steady evolution of bandwidth-demanding technology, which currently and more so in future, requires operators to use expensive infrastructure smartly to maximise the use of their equipment in a very competitive environment. The internet ISP market is very competitive and network equipment is expensive making infrastructure expansions costly ([91], [92]).

- The replacement of legacy devices with SDN devices is not an economically viable, thus extending the capabilities of current infrastructure by adopting evolutions in SDN ensures a networking environment that can respond to dynamic changes in bandwidth demand while optimising the placement of LSPs in the network.

- Hybrid SDN provides an environment where both legacy devices and SDN can co-exist ([77], [90]). The PCE-based inter-domain MPLS network architecture is still in a research phase. Single domain PCE-based LSP computation poses less of a challenge since the lack of network visibility is greatly reduced. The TE database used to compute the LSP monitors all network resources within a domain.

- Implementing PCEP, BGP-LS and OSPF-TE protocols on a centralised SDN controller allows for a network-aware PCE implementation to compute LSPs and extend the programmability capability of the network.

- Using PCEP, the bandwidth of LSPs can be adjusted based on demand. Dynamically shifting the bandwidth demand within the physical infrastructure in response to current user traffic profile goes a long way in addressing the future bandwidth needs.

- Automatic bandwidth adjustment reduces the need for human intervention in the re-optimisation of the network.

2.3 Research Questions

2.3.1 Question 1

By integrating hybrid-SDN; can the current legacy MPLS network devices adapt its logical topology dynamically in response to varying demand by automatically adjusting label-switched path bandwidths and routes?

2.3.2 Question 2

Can the logical topology of current legacy MPLS data networks be optimised continuously, in response to time-varying demand by the integration of hybrid-SDN and the path computation element?

2.3.3 Question 3

For the hybrid-SDN network; how is network throughput affected by varying the label-switched path bandwidth, network topology and optimisation method?

2.4 Hypotheses

Based on the problem formulation, research motivation and research questions, two hypotheses were formulated.

2.4.1 Hypothesis 1

By implementing hybrid SDN on legacy MPLS networks, the *path computation element communications protocol* and *border gateway protocol* with *link-state* will allow the MPLS network to respond dynamically to time-varying demand.

2.4.2 Hypothesis 2

By implementing a bandwidth optimisation algorithm in the path computation element, the unused but reserved label switched path bandwidth in the MPLS network can be returned to the MPLS network.

2.5 Hypothesis 1 Objectives

The work in this section is focused on testing the plausibility of hypothesis 1 (2.4.1) as well as provide test cases to answer research questions 2.3.1 and 0.

2.5.1 Hypothesis 1 Development Work

- Construct a legacy IP/MPLS network, which should then be extended to an HSDN network. Southbound protocols PCEP and BGP-LS will be implemented.
- Construct one partial mesh and one full mesh topology. This will provide variation in source-destination routes when the optimisation objective functions are implemented.

2.5.2 Hypothesis 1 Plausibility Testing

The following test cases have been developed and attempt to test the plausibility of hypothesis I. The tests aim to demonstrate two main claims:

1. That the legacy HSDN network can adapt dynamically.
2. That the legacy HSDN network can reorganise the LSPs in response to time-varying demand.

The test cases must explore answers to research questions 2.3.1 and 0.

2.5.3 Test Case 1 – Optimisation Objective Functions

Using the PCE, this test will implement two optimisation objective functions namely *minimising the maximum link utilisation* and *minimum routing cost*.

Purpose:

i. From hypothesis 1: To evaluate if the legacy HSDN network can adapt dynamically to varying demand.

ii. From hypothesis 1: To evaluate if the HSDN network can reorganise the LSPs in response to time-varying demand.

iii. This test will also evaluate questions 2.3.1.

Variables:

- Independent variables for this test case are the number of paths *k, links, Min-Max, Min-Cost* and *demand.* Here *k* is the number of candidate paths per demand pair. *Links* are the connections between the nodes and *Min-Max* and *Min-Cost* are the optimisation process.

- The variable *k* will have the values 2 and 5. For *k*=2, a maximum of two diverse paths will be presented to the optimiser. Likewise, for *k*=5, a maximum of five diverse paths will be presented to the optimiser if five such paths exist.

- The dependent variable for the test case is *Link utilisation.*

Expected outcome:

- For the Min-Max case, the expectation is that for small values of *k*, the demand will be spread over some of the links, but as the value of *k* is increased, the demand should be spread over more of the links. Ideally, the demand should be spread evenly over all the links.

- For the Min-Cost case, the expectation is that for small values of *k*, the demand will be concentrated on the shortest source-destination path. As the value of *k* is increased, the demand should be spread over more of the links. This will only happen if the link is at full capacity since only the route with the lowest cost is considered first.

2.5.4 Test Case 2 – Multi-Time Window Hose Model Optimisation

Implement multi-time window demand optimisation using the Hose model. The Hose model was discussed in 1.6.1 and 1.6.2.

For the HSDN and using the PCE, the test aims to optimise the LSP routing through the network based on the demand and observe how demand is distributed over the links. The hose model is used to ensure that there are always enough network resources.

Min-Max optimisation is used as opposed to Min-Cost, since the latter will use least cost routing, overloading certain edges while underutilising other edges. The multi-time window represents the change in traffic demand over four time periods. This scenario is typical of fluctuating demand over 24 hours, with morning, midday, evening and midnight traffic as an example of four possible periods, where the fluctuation in traffic is defined a priori.

Purpose:

i. From hypothesis 1: To evaluate if the legacy HSDN network can adapt dynamically to varying demand.

ii. From hypothesis 1: To evaluate if the HSDN network can reorganise the LSPs in response to time-varying demand.

iii. This test will also evaluate research questions 0.

iv. This test aims to demonstrate how the HSDN network, interfaces with the PCE to operate as part of a control loop that can automatically adjust to the time-varying demand. This result will demonstrate the flexibility of the system to dynamically adjust to demand at different time intervals.

Variables:

- Independent variables for this test case are *Time window,* the number of paths *k* and *Network links.* Here *k* is the number of candidate paths and *Network links* is all the direct connections between nodes.

- The dependent variable for the test case is the percentage of *Link utilisation.*

- The HSDN full mesh topology has 12 links each with 1Gbps capacity. The total utilisation of all 12 links is observed for every time window at which time LSPs are configured between source-destination pairs based on the demand.

- Eight *source-destination pairs* are set up, with demands that vary over four time-windows.

Expected outcome:

- The expected outcome is that; as the source-destination demand varies, the source-destination LSP bandwidth also varies, indicating that the LSP bandwidth adjusts dynamically. For this to happen, the PCE must optimise the LSP routes after each time window based on the available bandwidth in the physical topology. This result gives assurance that:

  i. The PCE is performing the optimisation process.

  ii. The PCEP reconfigured the LSPs in the network either by adjusting the LSP bandwidth or by reconfiguring an optimised path based on demand.

- With Min-Max optimisation there is an expectation that the demand will be spread over more edges as opposed to overloading the shorted path edges. The link utilisation is observed for every time window. Where possible, the optimiser is presented with a maximum of 5 source-destination paths.

2.6 Hypothesis 2 Objectives

The work in this section is aimed at testing the plausibility of hypothesis 2 (2.4.2).

2.6.1 Hypothesis 2 Development Work

- Implement a control loop that will dynamically respond to changes in demand and release overprovisioned LSP bandwidth back to the network. The control loop is essential for automatic dynamic control.

- Implement the Min-Max path optimisation algorithm.

2.6.2 Hypothesis 2 Plausibility Testing

The following test case has been developed and attempt to test the plausibility of hypothesis 2. The test aims to demonstrate the claim stemming from hypothesis 2:

- To evaluate if reserved but unused LSP bandwidth in the IP/MPLS network can be released back to the network

2.6.2.1 Test Case 3 - Total Bandwidth Reservation

The test observes the total bandwidth reservation over the entire network for every time window. The bandwidth of all the configured LSPs for that time window is summed and expressed as an instantaneous value.

The hose implementation allows for bandwidth to be released to the core of the network from nodes experiencing lower demand requirements. This is done by adjusting the reserved symmetric BW_{out} and BW_{in} hose limitation. The demand is then redistributed to nodes where the demand requirement is higher.

Purpose:

- From hypothesis 2: To evaluate if unused LSP bandwidth in the IP/MPLS network is released back to the network.

Variables:

- Independent variables for this test case are *Time window,* where each *Time window* has different source-destination demand and a set of *source-destination pairs*.

- The dependent variable for the test case is *Network Reservation* in Mbps. This is a measure of how much LSP bandwidth has been reserved on the HSDN network for each time *window*.

- For each time window, the total reserved bandwidth for all source to destination LSPs is summed.

Expected outcome:

- The expected results are that for each time window the total bandwidth reservation should vary during each time window. This result would indicate that:

  i. The bandwidths of the configured LSPs is changing dynamically for each *Time window* in the HSDN.

  ii. That unused bandwidth is returned to the network. This is one of the key requirements of the network, in that LSP bandwidths should not be overprovisioned when there is no or little demand for bandwidth.

iii. The result will attempt to show how bandwidth is shifted based on the demand at the time.

iv. Bandwidth optimisation within the network reduces the incidence of bottlenecks that are caused by increases in bandwidth demand. The process releases the unused bandwidth back to the network for paths where there is reduced demand for bandwidth.

2.7 Additional Objectives

This section addresses research question 2.3.3 and is addressed separately since it was not addressed in any of the prior plausibility test cases.

2.7.1 Research Question 2.3.3

These test cases look at network throughput as a function of *topology*, *optimisation method* and varying *LSP bandwidth*.

2.7.2 Test Case 4 – Increasing Path Diversity

Source-destination path diversity is a count represented by variable *k* of the number of distinct paths that exist between the source and destination nodes. The optimisation processes use these paths to optimise the demand placed on the edges making up the paths.

Purpose:

The objective of the test is to evaluate at which point the variable *k*, the number of candidate paths, no longer has any impact on the optimisation process.

Variables:

- Independent variables paths *k* between (*s, t*) *MinMax* and *MinCost*.
- The dependent variable is *Throughput*.

Expected outcome:

- The result will illustrate the relationship between path diversity and throughput for the full mesh topology.

2.7.3 Test Case 5 – Varying LSP Bandwidth, Topology and Optimisation

A full mesh topology is used to compare the optimisation processes as the demand pairs are increased. The demand pairs are the source-destinations pairs. These source-destination pairs are increased while observing the throughput of each optimisation process.

Purpose:

The test compares the overall network throughput when the two optimisation processes are performed on a full mesh topology.

Variables:

- Independent variable for this test case is *Demand Pairs*. This is the number of source-destination pairs with demands.
- The dependent variable for the test case is network *Overall Throughput* in Bps.

Expected outcome:

- There is an expectation that the minimisation of maximum link utilisation method will produce higher overall throughput since the optimiser spreads all demands evenly over several links.

The LSP bandwidth is increased incrementally while observing the throughput on both the full mesh and partial mesh topologies. Both optimisation processes are compared.

Purpose:

This test looks at the number of LSP that can be routed through the network as the LSP BW is increased.

Variables:

- Independent variables for this test case are *k, LSP BW, MinMax, MinCost, Partial Mesh* and *FullMesh*.
- The dependent variable for the test case is network *Maximum Throughput* in Mbps.

Expected outcome:

- The result will illustrate the relationship between LSP bandwidth, the optimisation process and the topology.

2.8 Research Constraints

The following section describes the research constraints.

2.8.1 Test Bed Constraints

- In an extensive survey, Amin *et al.* [96] provides a taxonomy on Hybrid SDN networks and refers to the test network implemented in this thesis as an SDN-BGP based network. Such networks only implement protocols BGP, OSPF and OpenFlow. For an ISP to implement OpenFlow, supporting hardware must be installed in the network. Since this is costly for ISPs, only the BGP and OSPF protocols are implemented in the testbed design for the research, since these two protocols are supported by most legacy networking equipment.
- A virtualised environment is used. In this case, the Cisco router OS is implemented but in a virtualised environment. Because of the virtual

implementation, connections between Gigabyte interfaces are soft links implemented using the host virtualiser software.

- The Cisco router OS has reduced data plane functionality, meaning that the OS cannot be used to transport real data as would be the case if the OS was operating in the manufacturer's router hardware.

2.8.2 Single Autonomous System

- An even more challenging task is the computation of an LSP, through multiple domains. Network visibility, in these cases, is greatly reduced because of boundary points between these domains [95].

- Single autonomous system route optimisation is implemented.

2.8.3 Real-Time Demand Measurement

- There is no real-time demand measurement. Demand is modelled using time windows which adjusts the demand at various source locations.

- The demand is assumed to be well behaved and does not take into consideration traffic bursts.

2.8.4 Convergence Time

- LSP reconfiguration does not take into consideration convergence time. For the small network implemented, convergence time is small. Since well-behaved traffic is considered, there is enough time between time windows for the network to converge. The importance of full convergence latency of traffic is discussed by Šeremet *et al.* in [83], but will not be considered in this study.

- Research assumes that all links are available and that the controller BGP-DB is fully aware of all network devices.

2.9 Research Plan

The following section gives a short overview of the research plan for this work.

2.9.1 Develop the IP/MPLS Test Network.

- Two topologies will be considered, i.e. full mesh and partial mesh.

- The two topologies provide an opportunity to compare LSP routing through two networks where source-destination routing choices are different. The full mesh topology offers more path diversity through the network than the partial mesh topology.

- For both topologies, MPLS will be configured.

2.9.2 Incorporate the SDN Controller Into the IP/MPLS Network

- Model the network mathematically and develop a control loop for the bandwidth adjustment process. The control loop must have the following features:

  i. Mathematically modelling the current network state.

  ii. Have a defined multi-time demand matrix.

  iii. Apply the network optimisation algorithm to the model based on the demand matrix.

  iv. A method to configure LSP paths and adjust LSP bandwidths based on the optimisation algorithm output.

2.9.3 Implement the Test Cases

- All test cases described in this chapter must be implemented and the results collected.

- Perform an analysis of the results.

2.10 Concluding Remarks

An attempt is made in this section to contextualise the research using the taxonomy proposed by [33] and to address the questions of why this research was undertaken, what work must be done and how the work will be implemented.

Several test cases were discussed which aims to test the plausibility of the hypotheses of this thesis as well as evaluate the operation of the implemented system.

In the next chapter, the bandwidth adjustment modelling problem is tackled. A mathematical model of the network is considered.

The multi-commodity network flow problem is discussed along with its formulation notation and solution features.

3 Bandwidth Modelling Problem

The auto-bandwidth adjust problem arises from the variation in bandwidth demand, where demand is the bandwidth required from the network. In the next section, two optimisation techniques are described namely, minimum cost routing and the minimisation of maximum link utilisation.

3.1 Graph Models

This section introduces the graph models of a network. A directed graph is used to describe the vertices and links of the network. A formulation for the network design problem is then described which details the multi-commodity network flow problem using link-path formulation descriptors.

3.1.1 Terminology

A flow is any demand that is carried on a path. The term flow is used in the general network flow modelling sense. The route between two endpoints is referred to as a path. A path may or may not carry non-null flows. A capacitated network is a network where the capacities are known. Traffic engineering seeks to find the optimum way to carry a demand in a capacitated network.

3.1.2 Directed Graphs and Networks

Directed and undirected graphs are elementary discrete mathematical models used to model networking problems. The structures consist of vertices and links connecting some of these vertices [97].

Graphs have many applications in various disciplines. For this study, graphs will be used to model a communications network. Communications links between nodes are represented by links while communications nodes, e.g. routers, are represented by vertices.

For the graph $G = (V, E)$, the primary distinction between a directed and undirected graph is that a directed graph, Figure 3.1 consists of a set of link elements E, where every link $e \in E$ is a two-element ordered subset of the set of vertices V, for example, $e = (i, j)$.

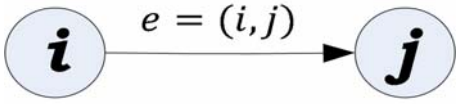

Figure 3.1: Directed node pair.

A link can therefore also be expressed in terms of its vertices, $(i, j) \in E$ which is different from $(j, i) \in E$. For the link $(i, j) \in E$, node i is considered the tail node with j the head node. The link (i, j) emanates from or is outgoing from node i and terminates at or is incoming to node j [97].

For the undirected graph, shown in Figure 3.2, $G = (V, E)$ the link pair $e \in E$ is unordered. The link joining the node pair i and j is either (i, j) or (j, i) where $(i, j) \in V$. The tail and head nodes are therefore undefined. An undirected link can permit the flow of traffic in both directions. This is different from a directed link (i, j), only permitting traffic flow from node i to node j, with i the source and j the sink. The undirected links are emulated by two opposing directed links.

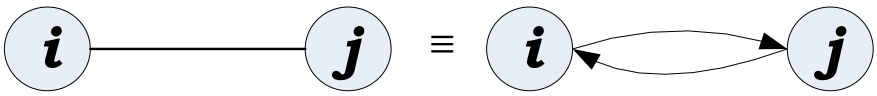

Figure 3.2: Undirected node pair.

There are various other graph attributes such as the graph degree, multi-arcs, subgraphs etc. These topics are well documented in the literature [71, 72], and will not be expanded further in this work. Other important aspects about the graph models used in this work are that the graphs considered are not multigraphs, that is, no graphs with multiple links connecting the same vertices will be considered. Only the simple graph case is considered so that the graph will contain no self-loops and thus, is not a multigraph.

3.2 Link-Path Formulation

In describing the *Network Design Problem* (NDP), two types of formulations are used to describe the multi-commodity formulation, namely the link-path formulation and the node path formulation. The link-path formulation is generally more effective for practical NDPs than the node-link formulation because there are fewer variables and constraints to consider. Also, in the node-link formulation, we do not control the paths and have to use all the paths, whereas in link-path formulation the paths can be controlled ([74],[75]).

The link-path formulation requires predetermined sets of candidate paths. In this work, the paths are calculated using the *k-shortest paths* (KSP) algorithm. Variable k is the number of shortest paths for an origin-destination pair. For moderate to large networks, generating 5-10 candidate paths per demand pair is sufficient in most of the cases [101]. The demand represents the traffic volume or the required bandwidth between an origin-destination pair. The demand is assumed to be bidirectional. For a network $G(V, E)$ with D demand pairs having positive demand volumes, each with candidate paths indexed by $p = 1,2, \dots, P_d$, the demand constraints can be written as:

$$x_{d1} + x_{d2} + \cdots + x_{dP_d} = h_d\,, \qquad d = 1,2, \dots, D \tag{3.1}$$

where x_{dp} denotes the unknown flow allocation variable for demand d over the paths p and h_d the demand volume variable for $d = 1,2, \dots, D$. Here, P_d is the total number of candidate paths for demand d. For each demand d, the demand volume h_d is split over the paths $1,2,..,P_d$. $\mathbb{P}_d$ is the set of KSP between the origin and destination pair.

To solve the KSP problem, a graph $G(V,E)$ with non-negative link weights, source s and destination t are given. The shortest path between s and t is determined, then the second, then the third shortest path and so on until the k^{th}-shortest path has been determined [100].

The k^{th}-shortest link-disjoint paths can also be found by temporarily deleting all the links on the shortest path before running the Dijkstra's algorithm again on the reduced graph. The Dijkstra's algorithm is an algorithm for finding the shortest path between nodes in a graph [100].

The link-path formulation fits better with the practical design problems but requires that a set of pre-computed paths be entered. The demand flow allocated to different KSP for a demand pair adds up to the total demand volume h_d, for the demand pair. Path flows are non-negative for all paths. Thus, for each demand d, the following flow conservation rules apply.

$$\sum_{p=1}^{P_d} x_{dp} = h_d \ , \quad d = 1,2,\dots,D. \tag{3.2}$$

If links have capacity constraints, one can define the following binary allocation variable

$$\delta_{dpl} := \begin{cases} 1, \text{if path } p \text{ for demand pair } d \text{ uses the link } l; \\ 0, \text{otherwise.} \end{cases} \tag{3.3}$$

For a specific demand pair d and link l, the term

$$\sum_{p=1}^{P_d} \delta_{dpl} x_{dp} \tag{3.4}$$

represents the flow on link l serving demand d. The capacity c_l of a link l must not be exceeded by the flow allocated to that link. If the summation is done over all demands $d = 1,2,\dots,D$ then the above term becomes,

$$\sum_{d=1}^{D} \sum_{p=1}^{P_d} \delta_{dpl} x_{dp} \tag{3.5}$$

and represent the flow carried by link l.

If the capacity of link l is already specified, c_l is used to denote the capacity of the link otherwise the variable y_l is used to denote the capacity needed to be planned for link l to carry the traffic flow. Thus, for link l variables y_l is defined as:

$$\sum_{d=1}^{D} \sum_{p=1}^{P_d} \delta_{dpl} x_{dp} = y_l \ , \qquad l = 1,2,\dots,L \tag{3.6}$$

which is the total flow on link l for all demand paths $p = 1,2,\dots,P_d$ and demands $d = 1,2,\dots,D$. Here, L is the total number of links in the network.

The link flow must not exceed the capacity of the link, and so

$$\sum_{d=1}^{D}\sum_{p=1}^{P_d}\delta_{dpl}x_{dp} = y_l \leq c_l\,, \qquad l = 1,2,\dots,L\,. \tag{3.7}$$

If ξ_{dp} is the nonnegative unit cost of flow x_{dp} on path p serving demand pair d, then the total cost for all the demands $d = 1,2,\dots,D$ can be expressed as

$$J_{TC} = \sum_{d=1}^{D}\sum_{p=1}^{P_d}\xi_{dp}x_{dp}\,, \tag{3.8}$$

where the unit cost for path p and demand d is calculated as

$$\xi_{dp} = \sum_{l=1}^{L}\delta_{dpl}\hat{\xi}_l\,, \tag{3.9}$$

where $l = 1,2,..L$, are the links forming the path p for demand d. Here, $\hat{\xi}$ is the cost of the link l.

3.3 Multi-Commodity Network Flow

The *multi-commodity network flow* (MCNF) problem is defined over a network, $G(V,E)$ where multiple commodities, also termed demands, must be accommodated from a specific origin node (s) to a destination node (t). The demands must be accommodated within the capacity constraints of the individual arcs in the network.

Network design goals such as minimum cost, minimisation of maximum flow, delay and disjoint paths can be imposed on the network, which is expressed through minimisation or maximisation of objective functions subject to capacity constraints [101].

In this work, the commercial LP solver package, IBM CPLEX optimiser, is used. The CPLEX solver implements optimisers based on the simplex algorithms, as well as primal-dual logarithmic barrier algorithms and a sifting algorithm [102]. CPLEX optimiser offers C, C++, Java, .NET and Python libraries that solve LP problems. It specifically solves linear or quadratic constrains optimisation problems where the objective to be optimised is a linear function ([78],[79]).

3.3.1 Notation

Consider a capacitated network defined as a directed graph $G = (V,E)$, where V denotes the set of nodes and E the set of arcs. $|V|$ denotes the number of nodes in the set V. For $|V|$ nodes, there are $|V|(|V|-1)$ unidirectional possible demand pairs. Node pairs with positive demand are listed by the index $d = 1,2,\dots,D$, where D is the

total number of demand pairs. For the origin-destination pair (s_d, t_d), let $\mathbb{P}_d^{s,t}$ be the set of all the simple paths connecting s to t.

Figure 3.3 illustrates a single commodity, $(d = 1)$, with origin-destination pair $(1, 4)$, with some nodes being transit nodes and the set of all simple paths $P^{1,4} = 3$.

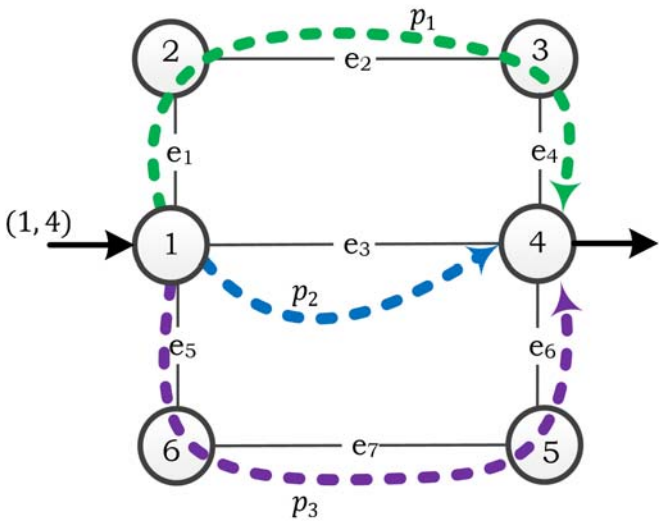

Figure 3.3: Three paths connecting s to t.

Mathematically Figure 3.3 can be expressed as;

$$\widetilde{\mathbb{P}}_{1,4} = \{p_1, p_2, p_3\} = \widetilde{\mathbb{P}}_d \tag{3.10}$$

where $\widetilde{\mathbb{P}}_{1,4}$ is the set of paths between demand pair (1,4)

$$\widetilde{\mathbb{P}}_1 = \{e_1, e_2, e_4\} \tag{3.11}$$

is the first path constructed from the set of links e_1, e_2 and e_3

$$\widetilde{\mathbb{P}}_2 = \{e_3\} \tag{3.12}$$

is the second path consisting of a link e_3 only and,

$$\widetilde{\mathbb{P}}_3 = \{e_5, e_7, e_6\} \tag{3.13}$$

the third path constructed from the set of links e_5, e_7 and e_6, with source node (s) and destination node (t) servicing demand 1 as;

$$s_1 = \{1\} \tag{3.14}$$

$$t_1 = \{4\} \tag{3.15}$$

Candidate paths can be indexed by $p = 1,2,..,P_d$ for each demand identifier d, where $P_d = 3$ for the network in Figure 3.3, is the total number of candidate paths for demand d and x_{dp} is the flow variable for demand d using path p.

For the multi-commodity case in Figure 3.4, $D = 2$ demands exist between origin-destination pair (s_d, t_d). The set of simple paths for $P^{1,4} = 3$ and $P^{2,5} = 3$.

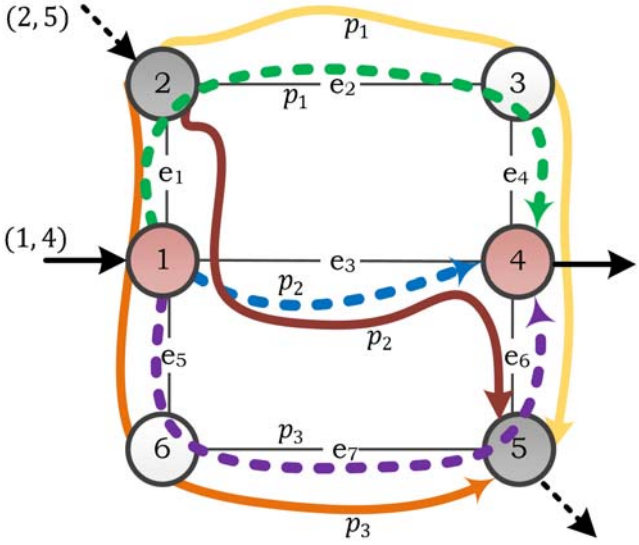

Figure 3.4: Multicommodity flow.

For Figure 3.4, $\widetilde{\mathbb{P}}_{1,4}$ can be expressed as in equations $\widetilde{\mathbb{P}}_{1,4} = \{p_1, p_2, p_3\} = \widetilde{\mathbb{P}}_d$ (3.10) to$t_1 = \{4\}$ (3.15).
For demand pair (2,5), the paths are expressed as;

$$\widetilde{\mathbb{P}}_{2,5} = \{p_1, p_2, p_3\} = \mathbb{P}_d \tag{3.16}$$

where $p_1, p_2, p_3$are the three paths between node pair (2,5).

$$\mathbb{P}_1 = \{e_2, e_4, e_6\}, \tag{3.17}$$

where e_2, e_4 and e_6 are the links forming path 1. Similarly,

$$\mathbb{P}_2 = \{e_1, e_3, e_6\} \tag{3.18}$$

and

$$\mathbb{P}_3 = \{e_1, e_5, e_7\} \tag{3.19}$$

have links e_1, e_3, e_6 and e_1, e_5, e_7 forming paths 2 and 3 respectively, with source node (s) and destination node (t) servicing demand 2 as;

$$s_2 = \{2\} \tag{3.20}$$

$$t_2 = \{5\} \tag{3.21}$$

In the case of splittable flows, the demand d uses multiple paths to split the flow x_{dp}. Non-splittable flows only use one path from source to destination to service the demand d. The links in $G(V, E)$ can also be indexed from 1 to $|E|$, and each path can be written as a combination of these links, by variable l ranging with the index as a reference. Hence, x_{lp} is the flow variable for the path p, using link l.

3.4 Variations on Objective Function

For this work, the notation shown in Table 3.1 is used.

Table 3.1: Notation used in the link-path formulation [101].

Notation	Description
***Given**:*	
D	Number of demand pairs with positive demand volume
L	Number of links
h_d	Demand volume of demand index d = 1,2,...,D
c_l	Capacity of link l= 1,2,...L
P_d	Number of candidate paths for demand d, d=1,2,..., D
δ_{dpl}	Link-path indicator. 1 if path p for demand k uses the link l; 0 otherwise
ξ_{dp}	The nonnegative unit cost of flow on path p for demand d
$\widehat{\xi_l}$	Nonnegative cost of flow per link l
***Variables**:*	
x_{dp}	Flow amount on path p for demand d
y_l	Link-flow variable for link l

3.4.1 Minimum Routing Cost

A nonnegative cost per unit of flow ξ_{dp} is defined for each path p and demand d. One method of defining a unit cost is to base it on the links in the path and to assign one unit per link. A flow on a path crossing two links would have $\xi_{dp} = 2$. The objective function based on the total cost to be minimised is expressed as [101]:

$$\min_{x_{dp}} J_{RC} = \sum_{d=1}^{D} \sum_{p=1}^{P_d} \xi_{dp} x_{dp} \tag{3.22}$$

subject to

$$\sum_{p=1}^{P_d} x_{dp} = h_d, \qquad d = 1,2,\dots,D \tag{3.23}$$

$$\sum_{d=1}^{D} \sum_{p=1}^{P_d} \delta_{dpl} x_{dp} \leq u_l\,, \qquad l = 1,2,\dots,L \tag{3.24}$$

$$x_{dp} \geq 0, \quad p = 1,2,\dots,P_d, \qquad d = 1,2,\dots,D \tag{3.25}$$

The unit path cost ξ_{dp} is the summation of the unit flow cost on links that make up the path. If we denote the unit link-flow cost as $\widehat{\xi_l}$ on link $l = \{1,2,\dots,L\}$, then ξ_{dp} for path p and demand k can be written as

$$\xi_{dp} = \sum_{l=1}^{L} \delta_{dpl}\, \widehat{\xi_l} \tag{3.26}$$

The simplex algorithm can be used to solve optimisation problems of the above type but requires that the problem is formulated in the standard form involving only non-negative variables and equality constants.

3.4.2 Minimisation of Maximum Link Utilisation

If the link flow on a link from node 1 to node 2, link 1-2, is denoted by y_{12}, then y_{12} is the sum of all individual flows passing through that link.

The link utilisation is expressed by the ratio y_{12}/c_{12} where c_{12} represents the capacity of link 1-2. Finding the link where the utilisation is maximum in a three-link network is now a case of choosing the ratio with the highest utilisation value.

Utilisation rate

$$r = max\{y_{12}/c_{12}\,, y_{13}/c_{13}\,, y_{23}/c_{23}\} \tag{3.27}$$

The maximum link utilisation is a value that is in the interval [0, 1] for the feasible case where link flow is less than the capacity of the link. It follows that r is greater than or equal to each of its components, therefore:

$$r \geq y_{12}/c_{12}\,, \qquad r \geq y_{13}/c_{13}\,, \qquad r \geq y_{23}/c_{23} \tag{3.28}$$

define the cost $J_{MLU} = r$ to be minimised

$$\min_{(x,y,r)} J_{MLU} = r \tag{3.29}$$

subject to

$$\sum_{p=1}^{P_d} x_{dp} = h_d, \qquad d = 1,2,..,D \tag{3.30}$$

$$\sum_{d=1}^{D}\sum_{p=1}^{P_d} \delta_{dpl} x_{dp} = y_l\,, \qquad l = 1,2,\dots,L \tag{3.31}$$

$$y_l \leq u_e r, \qquad l = 1,2,\dots,L \tag{3.32}$$

$$x_{dp} \geq 0, \qquad p = 1,2,\dots,P_d, \quad d = 1,2,\dots,D \tag{3.33}$$

$$y_l \geq 0, \qquad l = 1,2,\dots,L \tag{3.34}$$

$$r \geq 0 \tag{3.35}$$

A consequence of this model for non-uniform link capacities is that utilisation will be balanced over the links, but not necessarily flows. The multi-commodity network flow problem attends two goals namely:

1. Minimisation of cost.

2. Minimisation of maximum link utilisation.

Each flow problem produces a different optimal solution depending on the goal. Of importance is the observation in that for cases where the difference between the demand volume and the link capacity is very small, and where the demand volume and link capacity is very close, that the optimal solution in both these cases shows very little difference irrespective of the difference in goal [101].

3.4.3 Capacitated Flow Network

For the network with known link capacities, the type of problem lends itself to a capacitated design problem, where a feasible flow allocation vector that satisfies the demand constraints and the capacity constraints must be found.

Each demand $d \in \{1,2,\dots,D\}$ is associated with a positive demand volume h_d, expressed in general in the demand volume unit of Mbps. The link capacity unit is expressed in Gbps.

Each demand has associated with it a list of candidate paths $p \in \{1,2,..,P_d\}$, where P_d is the total number of paths for demand d. The general form for the list of candidate paths can be expressed as $\mathbb{P}_d = \{\mathcal{P}_{d1}, \mathcal{P}_{d2}, \dots, \mathcal{P}_{dP_d}\}$. $\mathbb{P}_d$ is the list of candidate paths for demand d. $\mathcal{P}_{d1}$ is the 1st path for demand d. Flows are assigned to demand volumes such that the flows for demand d on paths in the set $\mathbb{P}_d$ is denoted by x_{dp} where $p \in \{1,2,..,P_d\}$. The capacitated problem can be stated as:

$$\text{Demand constraints:} \quad \sum_{p} x_{dp} = h_d \,, d = 1,2,\dots,D \tag{3.36}$$

$$\text{Capacity constraints:} \quad \sum_{d}\sum_{p} \delta_{edp} x_{dp} \leq c_e \,, \quad e = 1,2,\dots,E \tag{3.37}$$

$$\text{Positivity constraints on variables:} \quad x \geq 0 \tag{3.38}$$

The flow allocation problem arises in the case where the capacity of a network is fixed and installed, but the network demand is changing in time. All feasible solutions to the capacitated problem are necessarily bifurcated. When non-bifurcated solutions are required, additional constraints must be imposed to force the single-path solution.

3.5 Multi-Time Window Flow Networks

For the multi-time window case, the demand volume for a time window exists only for that window. Multi-hour demand volumes must be addressed when dimensioning the network. Multi-hour demand is a demand that exists for a time window. The demand for the previous time window is not added to the current time window. This scenario is more applicable to IP networks, where demand volume can vary greatly in time.

This is different from *Multi-Period Design* (MPD) that spans over a time window period of weeks, months, even years. The demand volume for the MPD case is in addition to the demand volume that was present in prior periods also understood as incremental demand.

The capacitated multi-time dimensioning case is when the network already has the capacity, such as an already installed capacity, but multiple busy hours of demand needs to be considered for routing or flow optimisation and virtual tunnel establishment and reconfiguration if the problem is feasible

3.5.1 Multi-Time Window Capacitated Design

The case considered in this section is when the routes and flows are rearrangeable from one time planning period to the next, where the goal is to minimise the routing cost. Table 3.2 presents the notation used in the linear programming models for multi-time demand with rearrangeable flows:

Table 3.2: Notation used for multi-time demand.

Notation	**Description**
$\delta_{edp} = 1$	If e belongs to p realising d; 0 otherwise
h_{dt}	The volume of demand d at time t.
φ_{dpt}	Unit routing cost on path p for demand d in time window t
C_e	The capacity of link e.
x_{dpt}	Flow allocation to path p of demand d at time t

The objective function is defined as

$$\min_x J_{TWCD} = \sum_d \sum_p \varphi_{dpt}\, x_{dpt} \tag{3.39}$$

subject to

$$\sum_p x_{dpt} = h_{dt}, \quad d = 1,2,\dots,D \quad t = 1,2,\dots,T \tag{3.40}$$

$$\sum_d \sum_p \delta_{edp}\, x_{dpt} \le C_e, \quad e = 1,2,\dots,E \quad t = 1,2,\dots,T \tag{3.41}$$

where T is the time planning period. In the above, it is assumed that the problem is solvable. Due to rearangeability, the above problem can be solved separately for each time window since there are no coupling constraints between time windows once the capacity is given. For the case where the network does not have enough capacity to carry the demand and to ensure feasibility, the following inequality, $\sum_p x_{dpt} \le h_{dt}$ can be introduced instead of $\sum_p x_{dpt} = h_{dt}$. This will result in a feasible solution [100].

3.5.2 Link Cost Structure

One capacity unit on link e is assigned a cost ξ_e. Generally, the unit cost for every $e \in E$ is known. Then for the links that a path is composed of, the total unit cost is:

$$\zeta_{dp} = \sum_e \delta_{edp}\, \xi_e \qquad d = 1,2,\dots,D; \qquad p = 1,2,\dots,P_d \tag{3.42}$$

The link capacity granularity for all test cases is one unit, corresponding to 1 Gbps. The cost of 1 Gbps capacity unit on a link consists of termination cost and distance-based cost. If 10 is used as the cost for each of the two termination points per link and the distance cost per km is taken as 0.1, then $\xi_e = 2x10 + 0.1x distance_e$ where $distance_e$ is the distance in km per 1Gbps of link e.

3.6 Traffic Demand Time Patterns

The traffic demand in time windowed networks is represented by a three-dimensional non-negative hyper-matrix $\boldsymbol{D}(t)$, with the $(i,j)-th$ entry $\boldsymbol{D}_{i,j}(t)$ being a time function. Each entry represents the demand in terms of bytes or packets from source i to destination j in the time interval $[t, t+\Delta t] \subset \mathcal{T}$. The time planning interval is denoted by $\mathcal{T}$. The *traffic matrix* (TM) is distinct from the *demand matrix* (DM) in that TM is the carried load, whereas DM is the offered load.

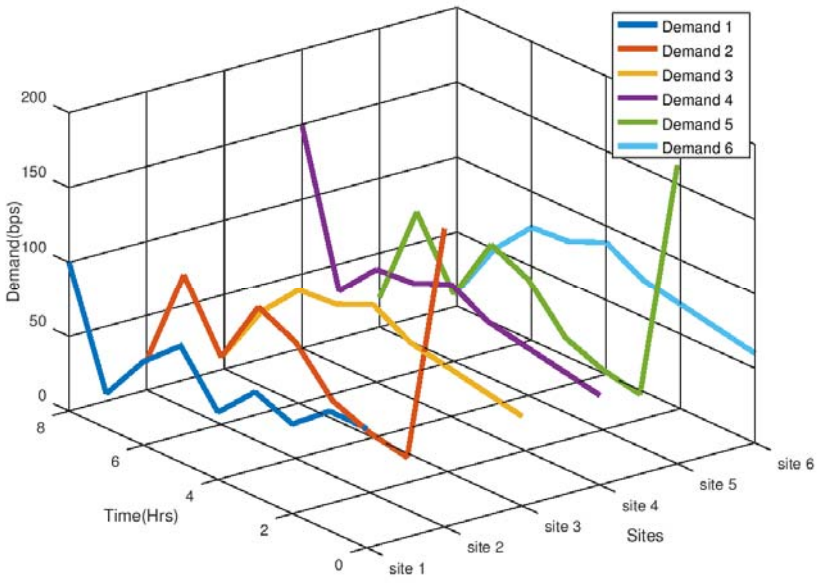

Figure 3.5: Multi-time demand matrix.

The demand matrix shown in Figure 3.5 is represented by a three dimensional, non-negative hyper-matrix $\boldsymbol{D}(t)$. Finding flow routes in a general network while not exceeding the capacity of any link is called the multi-commodity flow problem

3.7 Traffic Demand Uncertainty Models

This section looks at the models for specifying $\boldsymbol{\mathcal{D}}$, the set of demand matrices that the network should be able to support. The concept of demand uncertainty is one of the challenges this thesis needs to address in solving the automatic bandwidth adjust requirement based on shifting demand.

Two uncertainty models of interest are:

1. Hose model.
2. Polyhedral model.

3.7.1 Hose Model

In the hose model, only two parameters need to be specified per node, that is, the egress bandwidth and the ingress bandwidth. These parameters can be denoted by t_i^{out} and t_i^{in}, respectively. Each parameter specifies the sum of all traffic entering and leaving the site i [50].

Let S be the set of sites, then

$$\sum_{j \in S \setminus \{i\}} t_{ij} \leq t_i^{out}, \ \forall i \in S \tag{3.43}$$

and

$$\sum_{j \in S \setminus \{i\}} t_{ji} \leq t_i^{in}, \ \forall i \in S \tag{3.44}$$

The traffic demand matrix $\mathbf{t} = (t_{ij})_{i \in S, j \in S \setminus \{i\}}$ must satisfy all these equalities [103]. The traffic demand between sites i and j can be further restricted by an inequality t_{max}, if for that path, the demand never exceeds t_{max}. Thus $t_{ij} \leq t_{max}$ can be added to the polytope $\mathcal{D}$. The polytope $\mathcal{D}$ needs to be sufficiently large to allow the traffic to vary. If the polytope is too large, network resources will be wasted.

3.7.2 Traffic Uncertainty Polyhedral Model

D denotes a set of demands where $d \in D$ is characterized by a source s, a destination t and a demand volume h_d. By understanding the traffic demand behaviour between various origin-destination pairs, a set of linear inequalities can be used to express a bounded set containing all relevant traffic demands configurations. This set defines the polyhedral model by

$$D = \left\{ t \in \mathbb{R}_+^{|V| \times (|V|-1)} : \boldsymbol{At} \leq b \right\} \tag{3.45}$$

where V is the set of network nodes and t_{ij} the traffic demand from node i to node j. The demand is polyhedral if the vector of traffic demands $(t_{ij})_{i \in V, j \in V \setminus \{i\}}$ can be any point of a given polytope. $\mathbf{A}$ is a real-valued matrix and $\boldsymbol{b}$ is a real-valued vector. D is the traffic demand polytope. A model that permits a polyhedral traffic demand is polyhedral. The polyhedral model can be applied to the VPN context as well as other network contexts [103].

In this model, traffic belongs to a polytope. As network endpoints increase, traffic becomes difficult to estimate. This traffic uncertainty is modelled by a vector of traffic demands that satisfies some linear inequality and is bounded. In this model, the multiple traffic paths and how traffic demand is split over these paths are independent of the current traffic demand volume. As traffic demand varies, the routing remains stable. Routing based on this model is not adaptive.

3.8 Multipath Flow Transport

In the hose model, VPN endpoints specify the maximum bandwidth needed for sending or receiving data. To support this service, predetermined paths have to be allocated to connect the VPN endpoints. Finding appropriate VPN endpoint paths and bandwidth reservation while minimising total bandwidth used is an important problem to solve. VPN using the hose model was first introduced in. The hose model provides a means to describe traffic demand on VPN, providing a context for TE, routing and provisioning [104].

3.8.1 Symmetric Hose

A VPN must be provisioned so that bandwidth can be reserved on these paths to support the quality of service. VPN is represented by an undirected graph $G = (V, E)$, where V is a set of nodes and E is a set of links, also denoted by $\{i, j\}$, to emphasize the two end-nodes i, j in V. The order of the pair indicates that i is the head node and j is the tail node.

Let $Q \subset V$, denote the set of link nodes also referred to as terminals, that need to communicate with each other. A pair of terminals (s, t), where $s, t \in Q$ has d_{st} as the amount of traffic that has to be routed from s to t. We denote by S the set of an ordered pair of terminals $S = \{(s, t) \in Q \times Q : d_{st} > 0\}$. For a pair of terminals, $s, t \in S$ the convention is that $d_{st} \geq 0$ and $d_{ss} = 0$.

In the hose model, there is an upper bound $b(s)$ on the cumulative amount of traffic that can be sent or received by s at any point in time. For a node $s \in Q$, denoted by $b^+(s)$, the ingress bandwidth and $b^-(s)$ the egress bandwidth. For a link $e \in E$ there is an installed capacity $c_e \in \mathbb{R}_+$. For a set of K flows, define the source node of flow K as $s(k) \in Q$ and the destination node $t(k) \in Q$. Associated to a flow there is an uncertain bandwidth requirement $d_{s(k),t(k)}$. A valid traffic matrix $\mathbf{D} \in \boldsymbol{\mathcal{D}}$ is an assignment of a demand $d_{st} \geq 0$ to each unordered pair s, t of terminals that meets the cumulative upper bounds [105], that is,

$$\sum_t d_{st} \leq b(s) \tag{3.46}$$

Thus, the hose uncertainty model is the polytope $\mathcal{D}$ of traffic matrices satisfying the following linear constraints:

$$\sum_{t:t \neq s} d_{s,t} \leq b^+(s) \ , \forall s \in Q, \tag{3.47}$$

$$\sum_{t:t \neq s} d_{t,s} \leq b^-(s) \ , \forall s \in Q. \tag{3.48}$$

$$d_{st} \geq 0 \quad \forall (s, t) \in S \tag{3.49}$$

Bandwidth must be reserved and node-to-node path $s \leftarrow V$ must be selected such that any demand satisfying the upper bound written above 3.46 can be feasibly

routed. The routing process is described by a vector $\left[\mathbf{f}_{s,t}^{e}\right]_{s,t\in Q,e\in E}$, where $\mathbf{f}_{s,t}^{e}$ is the fraction of traffic from $s \in Q$ to $t \in Q$, that is routed through link $e \in E$ [106].

A path P_{st} is then selected for each $s,t \in Q$, and reserving the bandwidth $x_e \geq 0$ on each link e. For every valid $\mathbf{D}$, one must have $\sum_{st:e\in P_{st}} d_{st} \leq x_e$. The traffic matrix $\mathbf{D} = (d_{st})_{s,t\in Q}$ is assumed to be symmetric $d_{st} = d_{ts}$ for all pairs of terminals s,t and $b^{+}(s) = b^{-}(t)$, for all terminals $s \in Q$.

3.8.2 Bandwidth Reservation for Splittable Flows

The maximum amount of traffic sent through a link e for any valid traffic matrix $\mathbf{D} \in \mathcal{D}$ that should be reserved on each arc is:

$$B_e = max_d \sum_{s,t\in Q} f_{s,t}^{e} . d_{s,t} \tag{3.50}$$

such that

$$\sum_{t\in Q} d_{s,t} \leq b^{+}(s), \qquad s \in Q \tag{3.51}$$

$$\sum_{t\in Q} d_{t,s} \leq b^{-}(s), \qquad s \in Q \tag{3.52}$$

$$d_{s,s} = 0, \qquad s \in Q \tag{3.53}$$

$$d_{s,t} > 0, \qquad s,t \in Q, s \neq t \tag{3.54}$$

A routing scheme in a VPN must ensure that the bandwidth reserved does not exceed the link capacity, such that $B_e \leq C_e$ for all $e \in E$. Associated with each link $e \in E$, there is an installed capacity $c_e \in \mathbb{R}_+$ and a routing cost w_e.

The goal is to minimise the total cost of the reserved bandwidth, $\sum_e w_e x_e$.

3.8.3 Node Splitting

The node-splitting technique is used to transform a VPN endpoint v into two nodes v and v', which corresponds to the nodes outbound and inbound function. The original arc is also transformed into two links of the same cost and capacity. For every VPN endpoint artificial v and v' nodes together with two artificial directional links as shown in Figure 3.6, with capacity $b^{+}(v)$ and $b^{-}(v)$ are introduced [98].

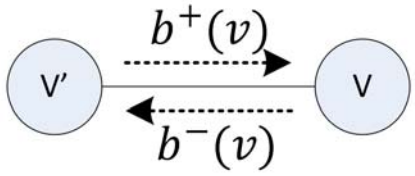

Figure 3.6: Node splitting.

Node v' becomes the source and destination of a connection while v acts as an internal node that does not contribute or consume any traffic. This transformation method guarantees that the hose ingress and egress capacities are incorporated into the analytical model and not violate the $b^+(s)$ and $b^-(s)$ bounds.

The topology of Figure 3.7, describes the test network which is used to evaluate the dynamic routing process in a VPN context using the hose model.

The topology forms a graph $G = (V, E)$, with the set of nodes $V = \{1, 2, 3, 4\}$ and the set of links $E = \{e_1, e_2, \dots, e_5\}$ representing the core network.

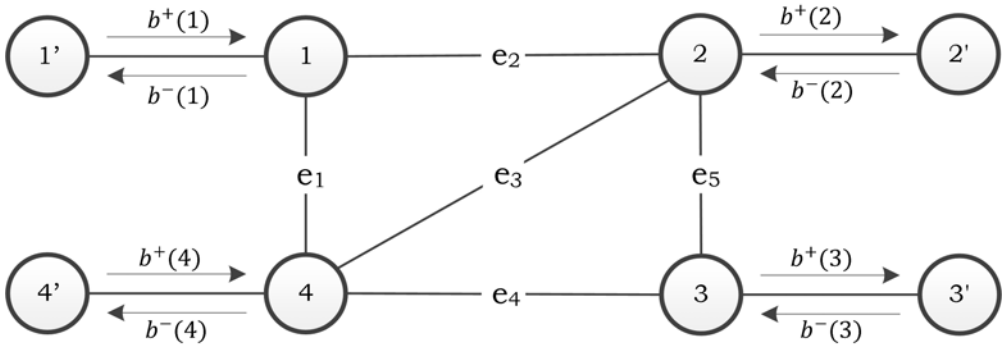

Figure 3.7: Network test model.

Nodes 1, 2, 3, 4 represents the backbone network. The special instance of the polyhedral set called the hose model is considered, where each node $v \in V$ has a total outgoing and incoming bandwidth that is denoted $b^+(v)$ and $b^-(v)$ respectively.

Let $H = \{(s,t): s,t \in V, t \neq s\}$ be the set of commodities where a source s has the demand d_{st} to be carried out for destination t.

The demand matrix is $\mathbf{D} \in \mathbb{R}_+^{(|H|-1)\times(|H|-1)}$ has the element $d_{s,t}$ governed by the polytope equation 3.48.

For the test topology shown in Figure 3.7, the set of demand commodities is described by $\mathfrak{D} = \{(1,2),(2,1),(1,3),(3,1),(1,4),(4,1)\}$. The inequality $d_{12} + d_{13} + d_{14} \leq b^+(1)$ then describes the total outgoing demand from node 1, which is not to exceed $b^+(1)$.

Similarly, $d_{21} + d_{21} + d_{41} \leq b^-(1)$ describes the total ingress demand for node 1.

The overall formulation can then be described as:

$$\sum_{j \neq i, (i,j) \in D} d_{ij} \leq b^+(i), \forall\, i \in V \tag{3.55}$$

$$\sum_{i \neq j, (i,j) \in D} d_{ij} \leq b^-(j), \forall\, j \in V \tag{3.56}$$

3.9 Concluding Remarks

This chapter describes the intricacies of the bandwidth modelling problem for flows that can be split. Various research objectives for this thesis were defined in Section 2.1.3. Bandwidth optimisation and automatic bandwidth adjustment are two of the key outputs of this work.

3.9.1 Graph Model

- A graph model $G(V,E)$ is used to represent the physical network topology. Topological information is collected from the routers using BGP-LS, OSPF-TE and PCEP protocols and the controller discussed in Chapter 1.

- Having the graph model, and a set of demands between nodes, the link path-formulation with its notation discussed in Section 3.2 is used to find the best route to minimise or maximise a particular objective.

3.9.2 Optimisation

- MPLS provides the capability for LSPs of varying bandwidth to be configured along predetermined source to destination paths.

- To ensure an even consumption of bandwidth per link, the LSPs traversing the network are optimised to minimise the maximum link utilisation.

- Also, the least cost routing is discussed as an additional attribute of the optimised LSP routing scheme using cost functions to be minimised.

3.9.3 Multi-Time Window Demand

- To address the varying demand concern for which the bandwidth of the paths in the network needs to automatically adjust, Section 3.5 introduces the concept of multi-time window demand. The traffic demand matrix introduced in Section 3.6, illustrates the varying nature of multiple demands over an 8-hour planning period. This leads to the discussing of two demand uncertainty models prescribed in Section 3.7, for dealing with the time-varying demands.

3.9.4 Multipath Routing Using the Hose Model.

- The network test model introduced in Section 3.8, explains the hose model in a VPN context. The hose model provides a convenient means of dealing with the demand uncertainty while optimising bandwidth usage.

In Chapter 4, the test network illustrated in Figure 3.7 is implemented together with the protocols and hardware architecture details to create a rich environment that allows for network programmability and ultimately for the TE deployment as described in the research taxonomy of Chapter 2.

4 Implementation

This chapter details the software and hardware components of the virtual networking environment that will support objectives 2.5, 2.6 and 2.7 as well as the limitation of the implementation. A full description is given of the hardware and software architecture.

Finally, the TE architecture is discussed in 4.3, which is then extended to include the control loop in 4.5. A connection diagram, Figure 4.6, further describes the topology of the research platform. This control loop, Figure 4.8 is the final TE system for the HSDN, that will enable the automatic adjustment and configuration of the LSP bandwidths.

4.1 Limitations of the Implementation

For this research, an attempt is made to approximate a real-world networking environment as closely as possible. This is done because of the prohibitive cost implication of setting up a laboratory with at least four routers to realise the test environment. Also, these routers must run the latest router operating system with support the new PCEP and PCC features.

To that end, a virtual environment, as opposed to a simulation environment, was chosen. The virtual environment is seen as a closer match to the real-world network than the simulation environment.

Based on this choice, this section serves to points out the cost constraints that informed the decision, the constraints on the research and the limitations of the virtual environment.

4.1.1 Cost Constraint

- The cost of a single router is very high. Due to the prohibitive cost of laboratory equipment, a virtual router environment is used.
- At the time of implementation, Cisco Systems Inc. released virtual images of their router IOS at no cost into the public domain, in a drive to support SDN development.
- These virtual routers have capabilities to interface with an SDN controller. Furthermore, each router had PCC capability and implemented the PCEP. These technologies described in 1.3.3, are important features for the environment to eventually realise the automatic bandwidth adjustment functionality.
- Several controller architectures, described in 1.4, are freely available for users and did not add any cost to the research.
- The choice of the controller was based on the interoperability ease between the Cisco Systems Inc. virtual routers and the controller.

4.1.2 Limitations of the Virtual Network

- The virtual routers operate the same router operating system as the commercial physical routing equipment however, the virtual routers have reduced data plane capability since the ethernet cards are realised in software.
- Since the ethernet cards are realised in software, there are no physically wired optic cables connecting the routers.
- All source-destination connections are realised in software and the line speed, 1Gbps in this case, is not realised. It will therefore not be possible to transmit source-destination data, i.e. video or voice, at 1Gbps. LSP bandwidths are reserved as a portion of the link speed.
- The sum of the bandwidths of all established LSPs crossing the same link will be limited by an upper bandwidth of 1Gbp for that link.

4.1.3 Implication of the Limitations on the Research

- The implementation requires traditional routing protocols such as MPLS, OSPF-TE, BGP-TE and PCEP to always keep track of the network state.
- To this end, for the features required for this research, there is no limitation on the routing capabilities of the Cisco Systems Inc. virtual routers and such capability matches the real-world scenario very well.
- Source-destination data transfer is limited to the transfer of routing information between nodes. The limitation here is that throughput cannot be measured on the virtual routers and is therefore inferred from the sum of bandwidths of the established source-destination LSPs.
- An established source-destination LSP indicates that a source-destination path could be found and that enough bandwidth exists on all the links of the source-destination path
- The demand is an expected value of data to be carried by the network for a time window and does not infer that real source-destination data was transmitted during testing.

4.2 Software and Hardware Requirements

This section describes the software and hardware requirements to support thesis objectives 2.5, 2.6 and 2.6.2.1. The components of the virtual HSDN environment, shown in Figure 1.8, will be detailed.

To evaluate both hypotheses 2.4.1 and 2.4.2 requires that a legacy MPLS network be used in an SDN context. This section will describe the hardware to implement the legacy MPLS network and finally, to implement the hybrid SDN environment.

4.2.1 Software and Hardware System Overview

The overall view of the four-layer virtual network architecture is shown in Figure 4.1.

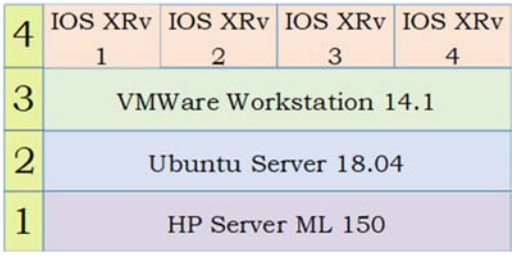

Figure 4.1: Virtualised network architecture.

The architecture in Figure 4.1 will provide a four router MPLS network to realise the legacy MPLS network described in hypothesis I, section 2.4.1. This will also be the base architecture to implement the hybrid SDN discussed in the research hypotheses section 2.4.

The software and hardware requirements discussed next will address these four layers.

4.2.2 Software Requirement

The software components required for the architecture in Figure 4.1 are:

i. ***Layer 1 - Server operating system:*** Must be capable of supporting the virtualisation environment.

ii. ***Layer 2 - Controller software:*** The controller provides the SDN components and is the interface between the application where the automation

iii. ***Layer 3 - Virtualisation software:*** All virtual routers will have to operate in a virtual environment as independent units.

iv. ***Layer 4 - Virtual router software:*** The implementation of the IOS-XRv virtual routers enables a fully programmable router environment.

1. Layer 1 - Virtual Router Software

IOS must have the capability to interface with an SDN controller and support protocols such as OSPF-TE, BGP-LS, PCC, PCEP as well MPLS.

Based on the above requirements, the following technologies are used to implement the research environment:

- ***Cisco IOS-XRv:*** Cisco provides a hybrid approach to SDN with three different operating systems, namely, (*Internetwork Operating System*) IOS, IOS-XR and (*Network Operating System Software*) NX-OS that are all different technologies

[107]. A freely available virtual version of the IOS-XR named the IOS-XRv seamlessly integrates with the VMware virtualisation platform. The virtual router contains all the IOS-XR router features, but with a reduced data plane.

- ***IOS*** is a family of software that is used on Cisco Systems routers and switches used for routing, switching, internetworking and telecommunications functions [108].

- ***IOS-XRv*** is a 32-bit virtual machine-based platform running the IOS-XR software containing a single router processor with control plane functionality and network interfaces with their functionality. The IOS-XRv router is not an emulator [110].

2. Layer 2 - Virtualisation Software

- VMware is the leader in server virtualisation. A hypervisor is computer software, firmware or hardware that creates and runs virtual machines, where a virtual machine is an emulation of a computer system. The computer on which the hypervisor runs is the host machine. VMware workstation is a hypervisor that runs on an x64 version of Windows or Linux operating system.

- VMWare Workstation Pro version 14.1.3 is used as the virtualisation environment to support the multiple OS installations of the IOS-XRv router software. Four installations of the Cisco IOS XRv release 611 operating system are embedded in the virtualiser to perform the traditional router functions.

3. Layer 3 - Controller Software

- The Opendaylight Project is an open-source project with a modular and pluggable controller at its core, implemented strictly in software within a Java Virtual Machine.

- The project is supported by Cisco, IBM, Juniper, Microsoft, Redhat, VMWare Ciena, Intell, Dell, HP and many more. The controller is designed to provide northbound APIs to be used by applications.

- The controller allows for control of the network infrastructure [42, 47] and has seen over 100 deployments with companies such as Orange, China Mobile, AT&T, T-Mobile, Comcast, Deutsche Telekom and Globe Telecom to name but a few [96].

4. Layer 4 - Server Operating System

The long-term support release of 64-bit Ubuntu Server version 16.04.5 was used as the OS platform for most of the development but was later upgraded to long-term support release 64-bit Ubuntu Server version 18.04.1.

4.2.3 Hardware Requirement

The deployment shown in Figure 4.1 will run on a Hewlett Packard Enterprise ProLiant ML150 Gen9 server, with CPU frequency 2.6 GHz, 10 cores and 40GB of RAM.

Hardware is responsible for the virtualised environment to operate, which has high memory and processing requirements. Each virtual router reserves a minimum of 3GB and a maximum of 8 GB of RAM.

4.3 Hybrid SDN Traffic Engineering Architecture

For flow-based routing, the traditional MPLS network is not adequate. An additional mechanism allowing flexible configuration, adjusting and deleting of LSPs carrying flows is needed.

The flexibility of flows and network programmability is provided by the *OpendDaylight* (ODL) controller. The implemented tiered architecture in Figure 4.2 provides a three-layer traffic engineering system:

- ***Layer 1:*** Virtual routers perform the traditional routing tasks of updating the routing table via the OSPF-TE protocol and enabling the MPLS environment. The BGP-TE protocol shares the routing table information with the controller in layer 2.

- ***Layer 2:*** The SDN controller is the intermediary transferring CRUD commands via its southbound PCEP interface to layer 1 router devices. The controller northbound interface interacts with the layer 3 application via a REST interface.

- ***Layer 3:*** Consists of a PCE module equipped with the IBM LP solver. The optimisation processes and LSP determination occurs in this layer.

3	Application Layer – PCE Module
2	Controller Layer – ODL
1	Router Layer – IOS-XRv

Figure 4.2: PCE based three-layer traffic engineering system.

4.3.1 Controller Layer Deployment

Project ODL is a collaborative large open-source SDN controller hosted by the Linux Foundation [113]. The ODL project controller, distribution Karaf version 0.5.4 Boron SR4 is selected from among various other open-source controllers because of the strong worldwide development support [32].

In [25] where a range of controller platforms is analysed, ODL is found to be the only controller supporting southbound APIs for PCEP and BGP-LS, which are two crucial features needed to enable the test environment of this research.

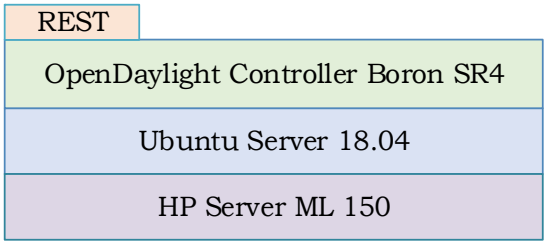

Figure 4.3: Controller deployment.

The controller implementation is shown in Figure 4.3. The ODL controller southbound protocols are essential in extending the capabilities of the legacy MPLS test network towards an H-SDN network.

The ODL controller architecture is shown in Figure 4.4 [114], consisting of the *Model-Driven Service Abstraction Layer* (MD-SAL). Data structure trees are built in the abstraction layer with the two main data structures being the configuration tree to store the network state and the operational tree for the current runtime status.

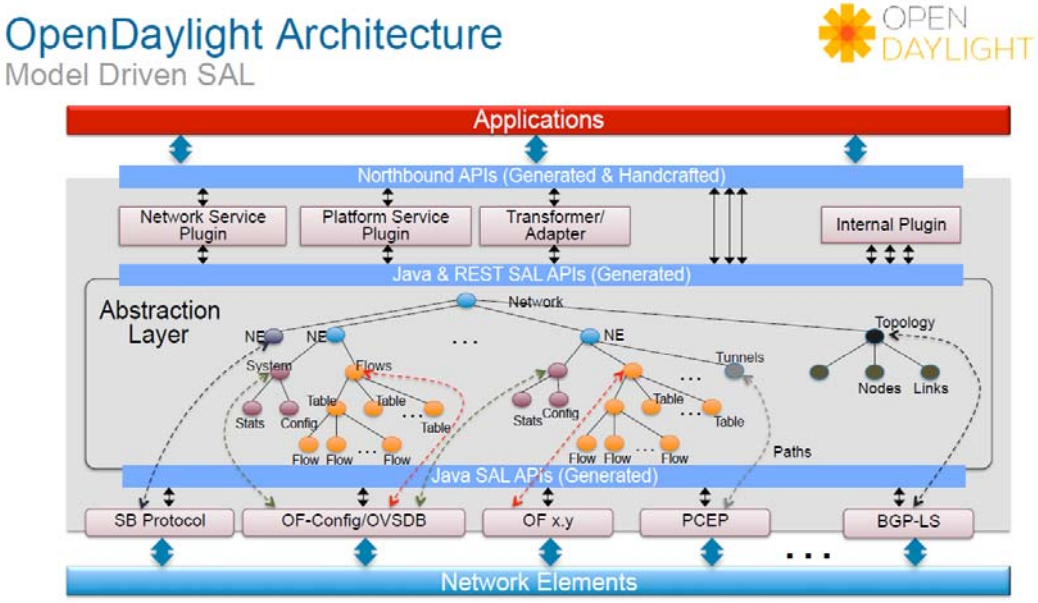

Figure 4.4: Model-driven service abstraction layer.

The ODL controller addressed the objective of hypotheses I and II, section 2.4 by providing:

- Southbound protocol plug-ins such as BGP-LS and PCEP. This allows the controller to interface with the legacy MPLS network creating a hybrid SDN environment.

- Standards-based northbound interfaces required for the automatic bandwidth adjustment network application. The application is discussed later in section 4.5. Path computation and network optimisation are key features of this network application.

Built on a Java platform, ODL controller can be deployed on any operating system that supports Java [34]. Other important features include [115] a bidirectional REST API and a web GUI.

The applications use the bidirectional RESTful programming interfaces to expose the underlying network over large scale networks to perform a wide range of advanced network operations [37]. This software enabling environment allows for the development of TE features on a software platform.

4.3.2 HSDN Network Topology

Figure 4.5 illustrates the realisation of the physical network topology. *Virtual Machine Networks* (VMnet), a feature of VMware workstation, is used to configure the routers in the topology shown in Figure 4.5.

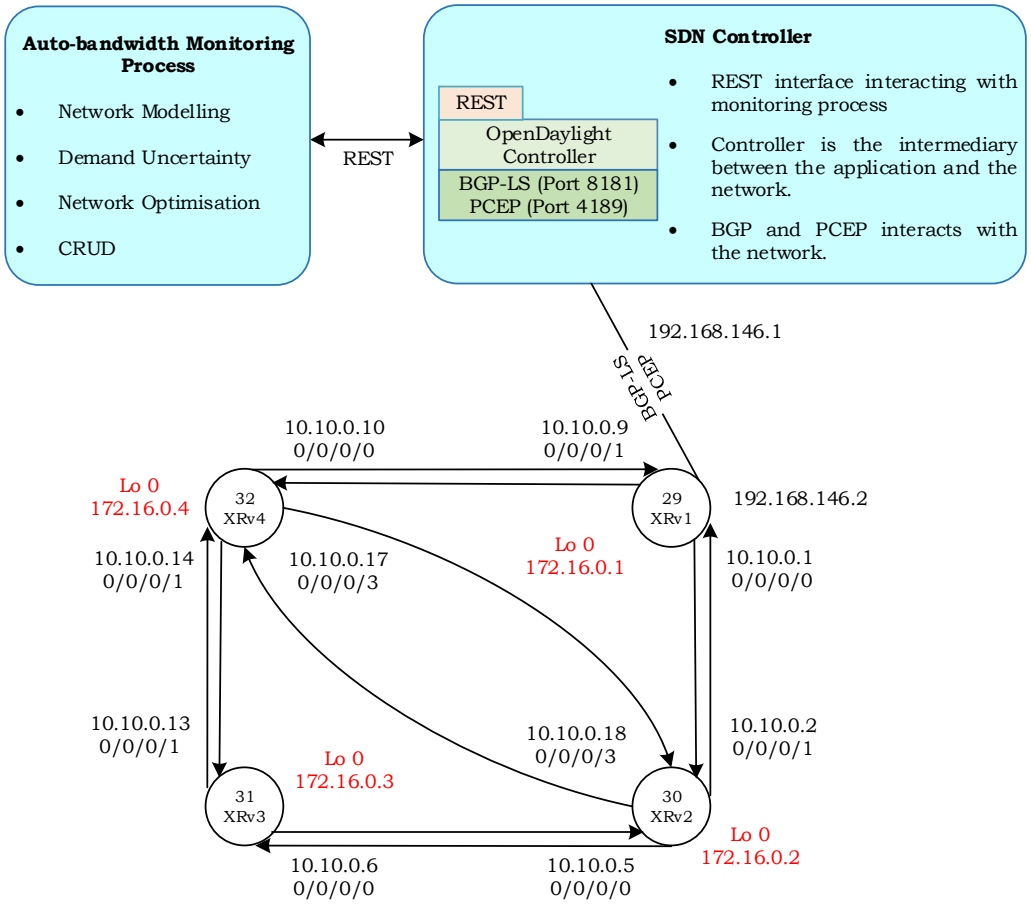

Figure 4.5: HSDN physical topology.

In SDN terms, the design is viewed as being composed of two layers, namely, the data layer testbed and the control layer. Within each MPLS router, the hardware layer primarily forwards packet and IGP updates. The controller layer consists of a controller, hosting the BGP-LS and PCEP southbound interfaces.

The MPLS nodes enable distributed routing. One node is used to distribute the routing table via BGP-LS to the controller layer.

The controller maintains a global abstracted view of the network state in the TED and the LSPDB. The controller computes paths centrally and provisions these paths via the southbound PCE interface. The active stateful PCE then builds a single instance of the local TED which contains the LSPDB information.

4.4 Implementing the Traffic Engineering Architecture

Details on the configuration of the routers and controller can be found in 9.1-9.4 of the appendix.

4.4.1 Implement the Hybrid SDN Environment

Addressing this autoconfiguration of the logical topology requirement requires the implementation of the SDN controller. A detailed description of the environment is given in 1.3.2.

The combination of the legacy MPLS network and the SDN controller then provides the HSDN environment to realise the hypothesis plausibility testing objectives described in 2.5, 2.6 and 2.7.

Furthermore, the HSDN will have constraints as described in 2.8, where, in particular, the testbed constraints are described in 2.8.1.

4.4.2 Implementing the MPLS Network

In addressing the research question in 2.3.1, a legacy MPLS network is required to adapt to changing demand. Additionally, from 0, the question requires that a logical topology must be shown to have the capability to reorganise itself, in response to the changing demand.

Addressing the above questions requires not only an MPLS environment, where a logical topology can be configured, but also requires a means to automatically setup and make changes to the topology.

4.4.3 Platform Setup

The network infrastructure, shown in Figure 4.6, is assumed to be operated by a single operator, resulting in a single domain network. Global knowledge of the network is maintained through the centralised control logic. As a result, BGP-TE can exchange information about traffic engineering databases without any restrictions.

The centralised controller maintains logical connectivity to all network elements via the PCEP and is capable of remotely configuring all nodes via the PCC.

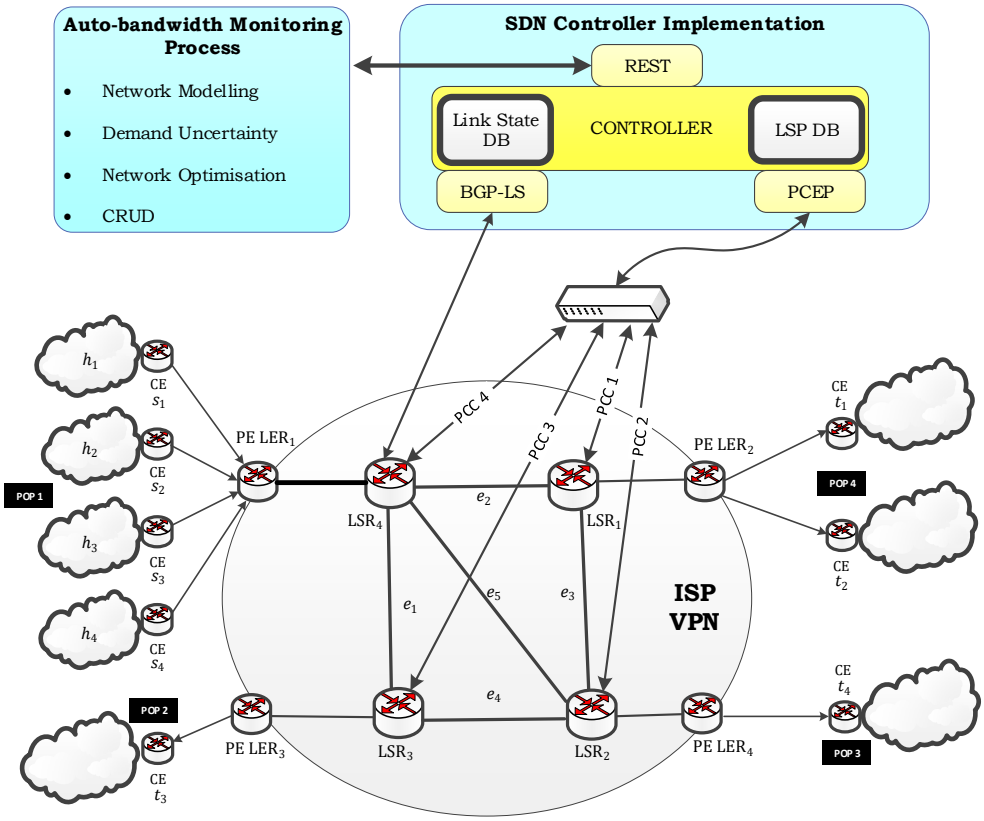

Figure 4.6: Network connectivity infrastructure.

- The SDN application interacts with the SDN controller through the REST interface.
- LSP computation relies on the topology information stored in the *Link State-Traffic Engineering Database* (LS-TED).
- The controller LSP-TED is kept current by the BGP interface, which has been configured in both the router and controller to share the IGP database of the router.
- The information remains local within an autonomous system resulting in path computation in a single administrative domain.

4.4.4 Network Topology Functionality

- The PCC topology is further illustrated in Figure 4.6 with the individual links from each router to the southbound PCEP interface. On each of the four routers, a PCC interface with the controller via the controller PCEP interface.

- In this solution, the PCE initiates the computation of a path. The PCE results are sent to the PCC, which in turn signals the LSP. The PCC does not request anything from the PCE.

- The configured LSP is a PCE-initiated LSP. The network is dynamically programmed to manage tunnels between network nodes. In this hybrid networking approach, the controller accommodates protocols such as BGP and PCEP.

- Of significance to this work are the BGP and PCEP southbound interfaces. These interfaces are used to communicate with legacy networks.

- For the case where multiple protocols, such as OSPF-TE and IS-IS-TE, are used to collect information for the TED, the IOS XRv router sets the preferences in the TED according to the protocols administrative distance. The higher the distance the less preferred the protocol and TED is for routing. IOS XR prefers the TED collected by the OSPF-TE protocol.

- The credibility values are derived from the default protocol preferences which are 10 – OSFP, 15 – IS-IS Level 1 and 18 – IS-IS Level 2. Hence, OSPF-TE becomes the most credible protocol.

4.4.5 VPN MPLS Testbed

In Figure 4.7, the core network containing LSRs(1-4) is shown. LERs(1-4) provide the entry points for traffic onto the VPN service provider network. The full mesh network is conceived to be a VPN, servicing demands via the CE routers.

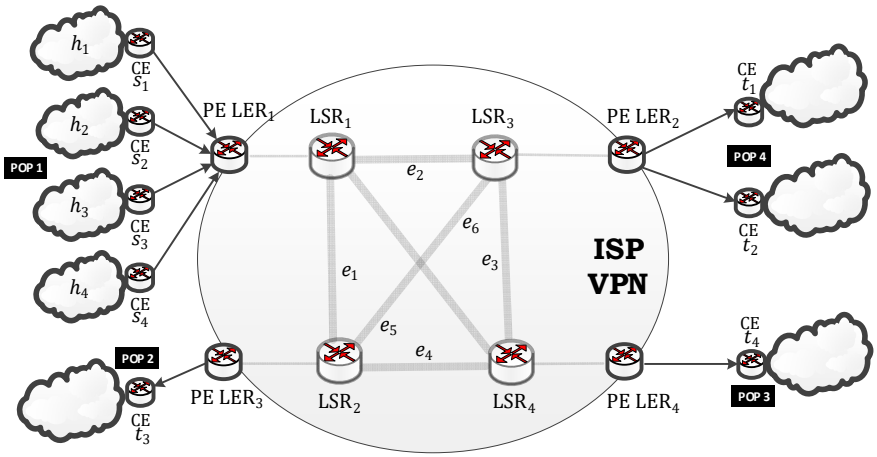

Figure 4.7: Network connectivity view.

The *Point of Presence* (POP) is where the *Customer Edge* (CE), and *Provider Edge* (PE), are interfaced. Distributed LSPs are constructed by partitioning the total end-to-end demand onto several sub-LSP's and distributing the paths through the network.

4.5 Traffic Engineering Control Loop Implementation

Figure 4.8 shows the block diagram of the auto-bandwidth monitoring process implemented in this work. It represents the complete auto-bandwidth control system, incorporating all the features to enable the realisation of hypothesis 1 and hypothesis 2.

The process is implemented in software and forms a feedback loop into the physical IP/MPLS network. The adaptive traffic engineering control loop formed has the typical control system elements namely:

- Controller:
    - PCE performs the optimisation process.
    - The optimisation process uses the TED as input.
- Process:
    - This is the virtual router network topology.
    - These routers have been configured to form the IP/MPLS network.
    - Label switched paths are routed throughout the IP/MPLS network.
- Feedback element:
    - BGP-TE shares the TED information with the controller.
    - OSPF-TE constantly updates the local router TED with any network changes.
- Error detector:
    - Prediction of traffic patterns is used as input and compared to current paths servicing the traffic demand.
    - If the LSPs bandwidths are over-provisioned, the bandwidth is reduced.
    - Similarly, if the provisioned LSP bandwidth is less than the demand, the LSP bandwidth must be increased.

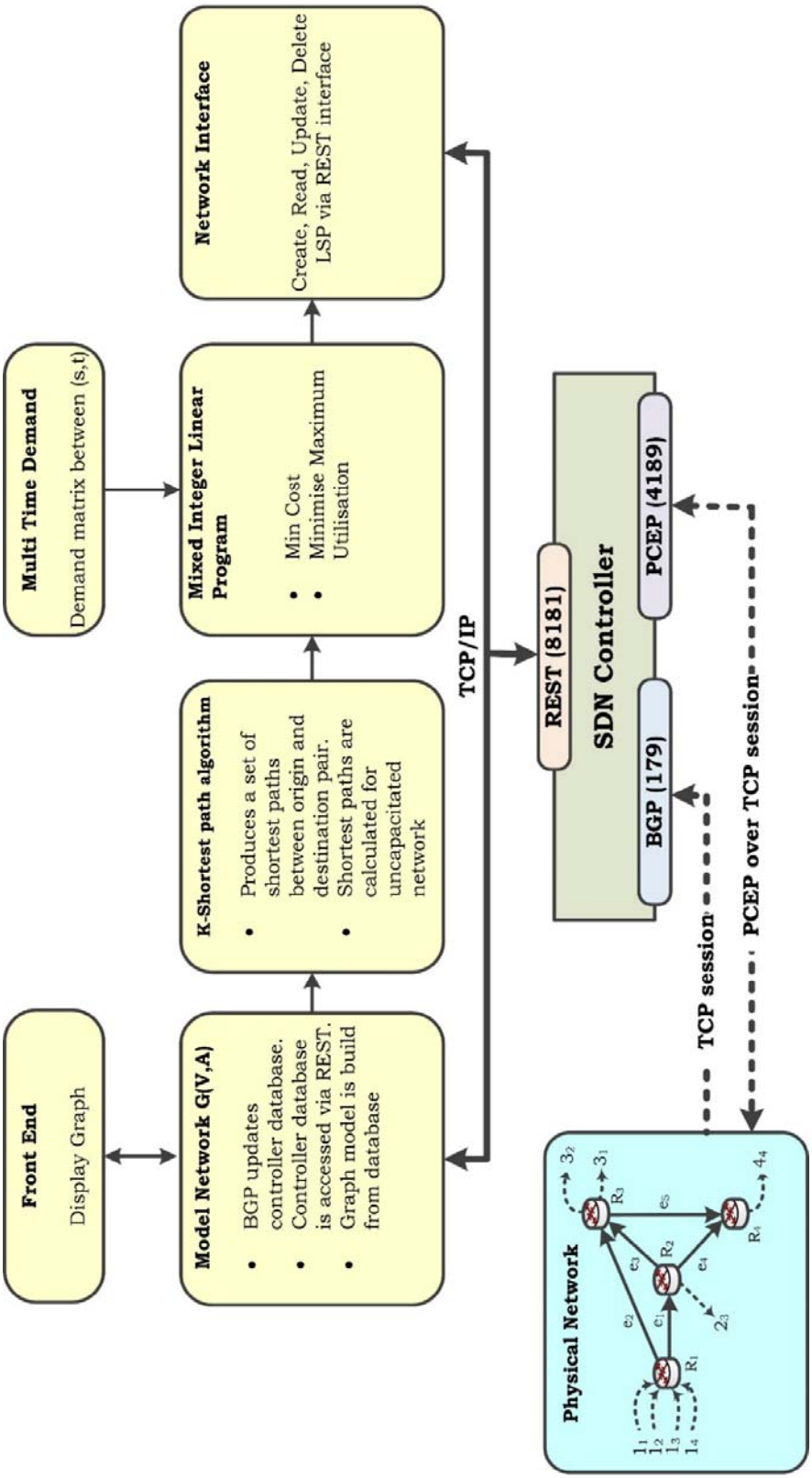

Figure 4.8: Auto-bandwidth mapping control loop.

A description of each process in the block diagram follows.

4.5.1 Physical Network

The physical topology is represented by the virtualised network that runs the network protocols OSPF-TE and BGP-TE to update the local and remote link-state databases on the current network state. Figure 4.9 shows the implemented routers using the VMWare virtualiser. Each router is fully capable of running OSPF-TE, BGP-TE and MPLS.

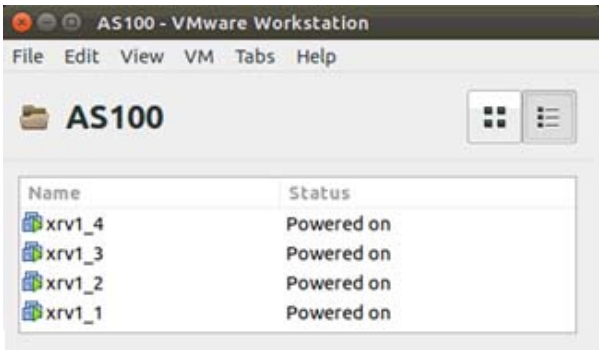

Figure 4.9: VMWare Network Topology

Only one router runs the BGP protocol and communicates with the controller via a TCP session. Any changes to the physical topology are tracked by the OSPF-TE protocol, which updates the local routing tables. These routes are then advertised to the SDN controller.

```
RP/0/0/CPU0:xrv1-1#show ip ospf database
Sat May  4 14:35:16.689 UTC

            OSPF Router with ID (172.16.0.1) (Process ID 100)

                Router Link States (Area 0)

Link ID         ADV Router      Age         Seq#       Checksum Link count
172.16.0.1      172.16.0.1      1560        0x8000002b 0x00aca1 7
172.16.0.2      172.16.0.2      1316        0x8000002b 0x00104a 7
172.16.0.3      172.16.0.3      1552        0x8000002b 0x001b1d 7
172.16.0.4      172.16.0.4      1838        0x8000002b 0x007db8 7

                Type-10 Opaque Link Area Link States (Area 0)

Link ID         ADV Router      Age         Seq#       Checksum Opaque ID
1.0.0.0         172.16.0.1      1560        0x80000028 0x00b0de        0
1.0.0.0         172.16.0.2      1316        0x80000028 0x00b4d8        0
1.0.0.0         172.16.0.3      1552        0x80000028 0x00b8d2        0
1.0.0.0         172.16.0.4      1838        0x80000028 0x00bccc        0
1.0.0.2         172.16.0.4      1838        0x80000028 0x0051d1        2
1.0.0.3         172.16.0.1      1560        0x80000028 0x0055de        3
1.0.0.3         172.16.0.2      1316        0x80000028 0x0076b3        3
1.0.0.3         172.16.0.3      1552        0x80000028 0x0052d7        3
1.0.0.3         172.16.0.4      1838        0x80000028 0x008493        3
1.0.0.4         172.16.0.1      1560        0x80000028 0x009987        4
1.0.0.4         172.16.0.2      1316        0x80000028 0x00270c        4
1.0.0.4         172.16.0.3      1552        0x80000028 0x009e78        4
1.0.0.5         172.16.0.4      1838        0x80000028 0x00739b        5
1.0.0.6         172.16.0.1      1560        0x80000028 0x00a265        6
1.0.0.6         172.16.0.2      1316        0x80000028 0x009974        6
1.0.0.6         172.16.0.3      1552        0x80000028 0x0062a5        6
RP/0/0/CPU0:xrv1-1#
```

Figure 4.10: Router OSPF LSDB

Figure 4.10 shows the routers OSPF LSDB, which contains the LSAs that describe the topology of the network.

4.5.2 SDN Controller

The SDN controller, Figure 4.11, constructs a node tree of the network within the MD-SAL. BGP shares the link-state information with the controller, which then updates the node tree within the MD-SAL.

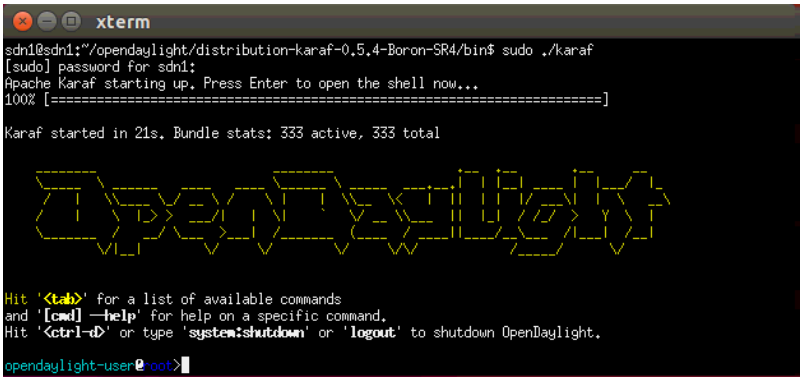

Figure 4.11: ODL controller.

The bandwidth control loop interacts with the network via the patch communications clients on each of the routers. Communications are maintained via the path communications element protocol, which ensures that LSP configuration instructions initiated by the path communications element reach the path communications clients.

4.5.3 Network Model

The bandwidth control TE application requests the LS database from the controller using the REST interface. The Javax and *JavaScript Object Notation* (JSON) Java packages are useful tools in performing these actions. The controller responds with the requested database in JSON format. The JSON file is processed, producing a graph table as shown in the screenshot of Figure 4.12.

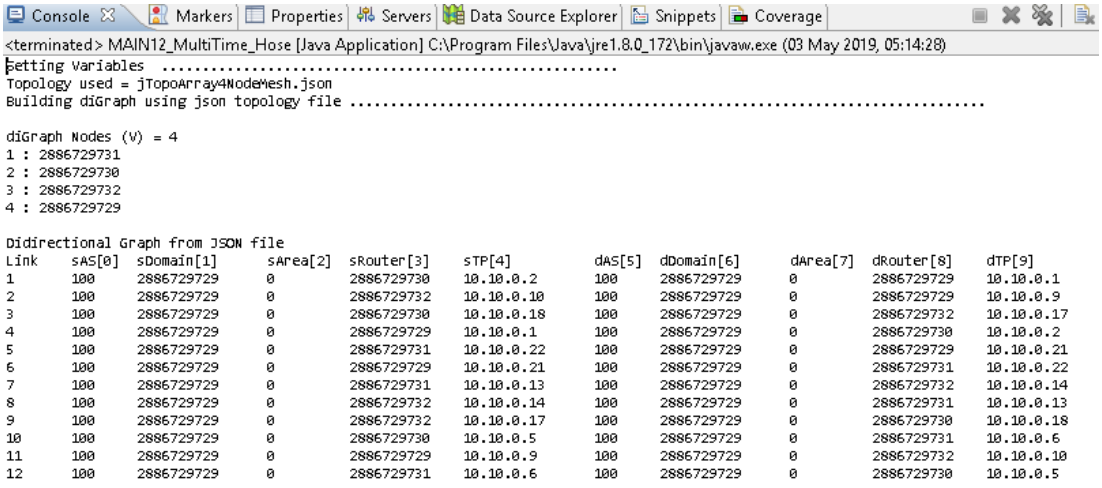

Figure 4.12: Output of processed JSON file.

Twelve bidirectional links are shown each with its source router and destination router IP address. In this representation, each link in the virtual network has a separate transmit and receive link representation in the JSON file output. The descriptors of the links in the table are only populated if the physical links in the virtual network are up. If any link fails or is shut down, the link descriptor information is no longer maintained in the database.

For any change in the state of the link, OSPF-TE will update the router local database, and BGP will share that state with the controller MD-SAL and in doing so keep both the router local database and the MD-SAL network abstraction in sync. These table entries are then formulated into a graph model of the form $G = (V, E)$, with the graph model updated every time there is a change of state in the virtual network topology.

4.5.4 k-Shortest Paths Algorithm

The requirement of the link-path formulation discussed in Chapter 3.2 is that a set of available paths be precomputed. Depending on the value of k, several k shortest paths between all source and destination nodes are computed on the graph $G = (V, E)$.

The Dijkstra loopless shortest path algorithm is used and repeated k times. The algorithm removes the links from the previously calculated path before recalculating the shortest path. The JGraph open-source Java library is used for this purpose.

In Figure 4.13, the ingress traffic at the LSR_1 node is distributed on sub-LSPs, which can either be link-disjoint or not.

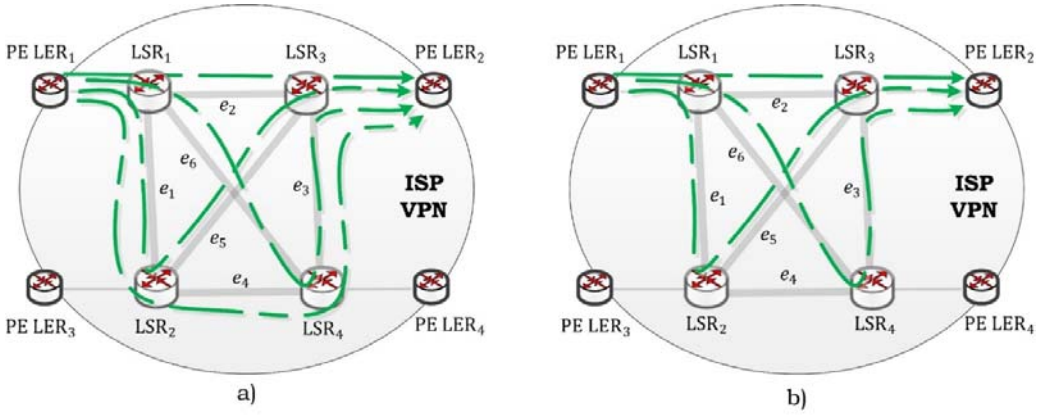

Figure 4.13: Label switched paths.

Figure 4.13 a) shows a path from the source to destination without any requirement that the paths should be link-disjoint.

Figure 4.13 b) shows the shortest path link disjoint requirement. Disjoint paths do not necessarily have the same number of hops.

4.5.5 Mixed Integer Linear Programming

The optimisation stage calculates the minimum routing cost or maximises the minimum link utilisation with graph $G = (V, E)$ as input and produces a set of paths used to configure the LSPs through the virtual network.

The IloClpex library from CPLEX is used to solve the *Mixed Integer Linear Programming* (MILP) problem to optimise the LSP assignment based on the set of demands. See Chapter 3.3-3.4 for a full description of the objective functions using the multi-commodity formulation. Multicommodity is the case where more than one demand pair has positive demand volume. The set of paths resulting from the KSP algorithm is used as the path options to attempt to satisfy the demand requirements. The resulting multi-commodity flow problem is then optimised based on minimum cost or minimising the maximum link utilisation.

4.5.6 Multi Time Demand

Multi-time demand is used to describe the demand for bandwidth on the network over time. Chapter 3.5 – 3.6 gives a full description of the multi-time window concept as well as the traffic demand matrix.

4.5.7 Network Interface

In the final stage, the calculated LSPs can be created in the virtual MPLS network. If the LSP path already exists, it can also be updated or deleted from the virtual MPLS network. The southbound PCEP enables such remote configuration capabilities.

Figure 4.14 shows a few of the LSP's, also called tunnels, configured through the virtual MPLS network. The status of each path is indicated as either up or down with the various path bandwidths indicated.

```
RP/0/0/CPU0:xrv1-1#sh mpls traffic-eng tunnels | inc "Signalled-Name|Admin|Ban$
Fri Jul 19 15:21:48.513 UTC
  Signalled-Name: 1000
    Admin:    up Oper:   up   Path:  valid   Signalling: connected
    Bandwidth Requested: 5000 kbps  CT0
    Bandwidth:     5000 kbps (CT0) Priority:  7  7 Affinity: 0x0/0xffff
  Signalled-Name: 1001
    Admin:    up Oper:   up   Path:  valid   Signalling: connected
    Bandwidth Requested: 5000 kbps  CT0
    Bandwidth:     5000 kbps (CT0) Priority:  7  7 Affinity: 0x0/0xffff
  Signalled-Name: 1002
    Admin:    up Oper:   up   Path:  valid   Signalling: connected
    Bandwidth Requested: 5000 kbps  CT0
    Bandwidth:     5000 kbps (CT0) Priority:  7  7 Affinity: 0x0/0xffff
  Signalled-Name: 1003
    Admin:    up Oper:   up   Path:  valid   Signalling: connected
    Bandwidth Requested: 5000 kbps  CT0
    Bandwidth:     5000 kbps (CT0) Priority:  7  7 Affinity: 0x0/0xffff
  Signalled-Name: 1004
    Admin:    up Oper: down   Path: not valid   Signalling: Down
    Bandwidth Requested: 100000 kbps  CT0
    Bandwidth:   100000 kbps (CT0) Priority:  7  7 Affinity: 0x0/0xffff
```

Figure 4.14: PCE configured LSP.

4.5.8 Front End

The front end, Figure 4.15, gives a basic visual display of the graph $G = (V, E)$ representing the network topology.

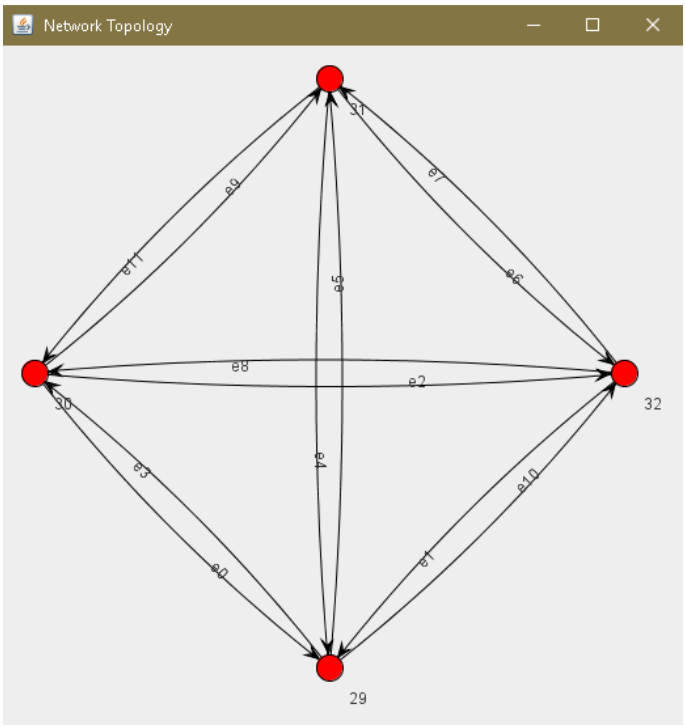

Figure 4.15: Visualised network topology.

Four nodes are shown representing the core routers. A full mesh topology is shown with bidirectional arcs connecting the nodes.

4.6 Cross-Functional Flow

The cross-functional flow chart shown in Figure 4.16, illustrates the functions within each layer and how these layers interact with each other. The three layers described are:

- REST application layer:
    - The REST application performs the function of PCE where the optimisation processes compute the routes that will serve the demands.
    - The application can be resident on any device if there is IP connectivity between the device and the PCE application.
- Controller layer:
    - Extends the programmability capability of the MPLS network.
    - Provides a PCEP Southbound interface enabling PCE to PCC communications.

- Network topology layer:
    - Virtualised routers run protocols such is OSPF-TE, BGP-TE and MPLS.
    - A full mesh and partial mesh topology were configured.
    - All routers have a PCC configured.

4.6.1 REST Application Layer API

Interface: RESTfull HTTP Methods: POST, GET, PUT, DELETE.

Input: Digraph $G = (V, E)$and demand pairs.

Output: Flow allocation resulting from the optimisation process for LSP construction.

1. Initialises the source and destination demand pairs and set the demand volume for each source to destination pair.
2. Instantiates the directional graph (digraph) type.
3. Uses the buildGraphFromRest method to retrieve the JSON format topology array from the controller.
    a. Constructs link and node array from JSON topology array.
    b. Constructs digraph $G = (V, E)$ with AS, domain, area, router and termination point information for both source and destination node.
    c. Returns digraph $G = (V, E)$ as the current network topology graph.
4. Using JGraph, find KSP in $G = (V, E)$ to the specified demands.
5. Call Optimiser:
    a. Minimise Maximum link utilisation with the KSP to demands as input.
        i. Return flow allocation for MinMax.
    b. Minimum Cost ink utilisation with the KSP to demands as input.
        i. Return flow allocation for MinCost.
6. Call PCEP module to create LSP via the RESTfull interface.
    a. MinMax link capacity array and KSP description as input.
    b. MinCost link capacity array and KSP description as input.

4.6.2 Controller Layer

Interface South: PCEP and BGP

Interface North: REST

Input: BGP updates to the LS-DB and LSP-DB

Output: Responds to RESTfull HTTP requests using methods types POST, GET, PUT and DELETE.

1. The BGP module updates the controller database on the status of the network.
2. The PCE creates the LSP using the resident PCC at each node.

4.6.3 Network Topology Layer

Interface South: PCEP and BGP

Input: Physical topology

Output: Routing tables via BGP

1. Physical topology running the IOS-XRv images.
2. Physical topology has been constructed as full mesh topology.
3. BGP-LS, OSPF-TE protocols update the LSP-TED and LS-TED.
4. MPLS protocol enables an LSP environment and allows for flexibility in LSP construction.
5. PCEP and PCC protocols allow CRUD operations.

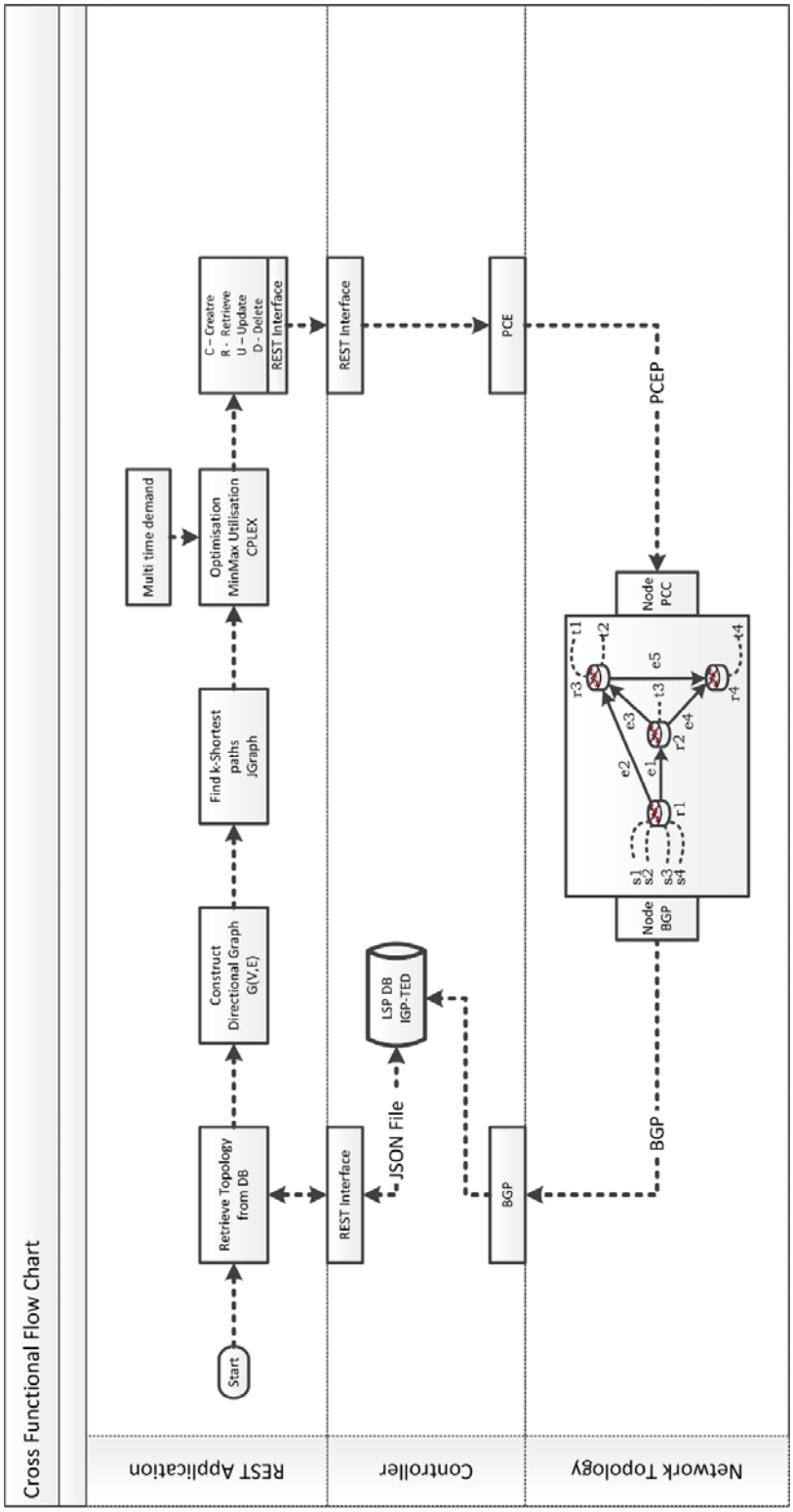

Figure 4.16: Cross-functional flowchart.

4.7 Conclusion

A functional description of the auto-bandwidth monitoring process is given. These processes illustrated in Figure 4.*8* and Figure 4.*16* completely describes the interrelationship between the systems in the control loop.

The auto-bandwidth control loop describes processes with capabilities to monitor the current network state, react to the current demand for network capacity and then to automatically adjust network resources continuously.

The virtualised routers perform the important task of implementing the standard TE protocols and is key for creating the MPLS environment. This network environment is enhanced through the implementation of the SDN controller.

This creates a hybrid SDN environment, which enhances the programmability capability of the network. As a result, the SDN application can both read the state of the physical network as well as perform CRUD tasks on the network through the controller PCEP interface.

The SDN application as part of the auto-bandwidth adjust control loop then optimises the network continuously while being aware of the current network state and the demand for that time window.

The next chapter, Chapter 5, will describe the experimental work. The goal of the experiments is to test the SDN application and its interaction with the virtual topology in achieving the objectives of this thesis.

5 Experimental Work

The experimental work will attempt to validate that the research hypotheses and objectives, key research questions and implementations were addressed and to what extend the addressed work validates the hypotheses and

5.1 Objectives Addressed in this Work

This section lists the research objectives described in chapter 2 and states whether the objectives were addressed.

5.1.1 Research Objectives

Objectives Related to Hypothesis I	**Addressed in this work**
Development Work for Hypothesis I • Develop two testbeds. One with a partial mesh topology and one with a full mesh topology. • The IP/MPLS network will contain single-vendor devices. Southbound protocols PCEP and BGP-LS will be implemented.	Yes
Test Case 1 – Optimisation Objective Functions • Test case 1 was developed to test the plausibility of hypothesis 1, see section 2.4.1. and address research question 2.3.1. • The test compares the overall network throughput when the two optimisation processes are performed on a full mesh topology.	Yes
Test Case 2 – Multi-time Window Hose Model Optimisation • Test case 2 was developed to test the plausibility of hypothesis 1, see section 2.4.1. and address research question 2.3.2. • The test observes the total bandwidth reservation over the entire network for every time window. The bandwidth of all the configured LSPs for that time window is summed and expressed as an instantaneous value. • The result will attempt to show how bandwidth is shifted based on the demand at the time.	Yes
Objectives Related to Hypothesis II	**Addressed in this work**
Development Work for Hypothesis II • Implement a control loop that will dynamically respond to increases in demand. The control loop is essential for automatic dynamic control.	Yes

Test Case 3 – Total Network Reservation • Test case 3 was developed to test the plausibility of hypothesis II, see section 2.4.2. • Bandwidth optimisation within the network eases the bottlenecks due to increase in demand and release bandwidth back to the network when there is no longer a demand	Yes
Additional Objectives Addressing Research Question 2.3.3	**Addressed in this work**
Test Case 4 – Increasing Path Diversity • Test case 4 was developed to address research question 2.3.3. • The objective of the test is to evaluate at which point the variable *k*, the number of candidate paths, no longer has any impact on the optimisation process. • For the HSDN and using the PCE, the test aims to use one of the optimisation methods, and observe how demand is distributed over the links. The hose model is used to ensure that there are always enough network resources.	Yes
Test Case 5 – Varying LSP Bandwidth, Topology and Optimisation • Test case 5 was developed to further address research question 2.3.3. • The purpose of the test is to observe the PCE performing the minimisation of maximum link utilisation process and the minimum routing cost process. • This test looks at the number of LSP that can be routed through the network as the LSP BW is increased. • The full mesh and partial mesh topologies are used to assess the impact that topology has on throughput.	Yes

5.1.2 Research Questions

Key Questions	**Addressed in this work**
• Can the current legacy MPLS network devices of ISPs, adapt the logical topology dynamically in response to time-varying demand for bandwidth by automatically adjusting LSP bandwidths and routes (2.3.1)?	Yes
• Can the logical topology of current legacy MPLS data networks be optimised continuously, in response to time-varying demand by the integration of HSDN and the PCE into existing data networks (2.3.2)?	Yes
• For the HSDN network; how is network throughput affected by varying the LSP bandwidth, network topology and optimisation method (2.3.3)?	Yes

5.2 Evaluation of SDN Application and Control Process.

The output of the HSDN auto-bandwidth adjustment optimisation application is observed for both the full mesh and partial mesh topologies. The CPLEX MIP solver is used to solve equations 3.22 and 3.27. The network responds to demand variation per time window by adjusting the LSP bandwidth and route automatically.

5.2.1 Development Work for Hypothesis I

Figure 5.3 shows the *full mesh* (FM) topology of the isolated source to destination terminal pairs with node 1 acting as the source node. The arcs joining the nodes are bidirectional. All link capacities for the network under test has a bandwidth of 1Gbps. The cost per link $e \in E$ is one. The test network considered has one service class and four busy hours for the multi-time window scenario

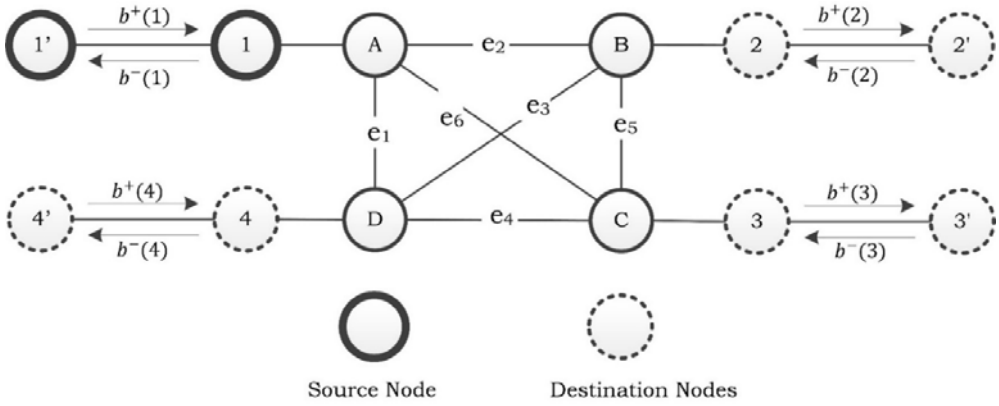

Figure 5.1: Single source to destination pairs. (Full Mesh)

Figure 5.2 shows the source to destination pairs for the *partial mesh* (PM) topology with node 1 still acting as the source node.

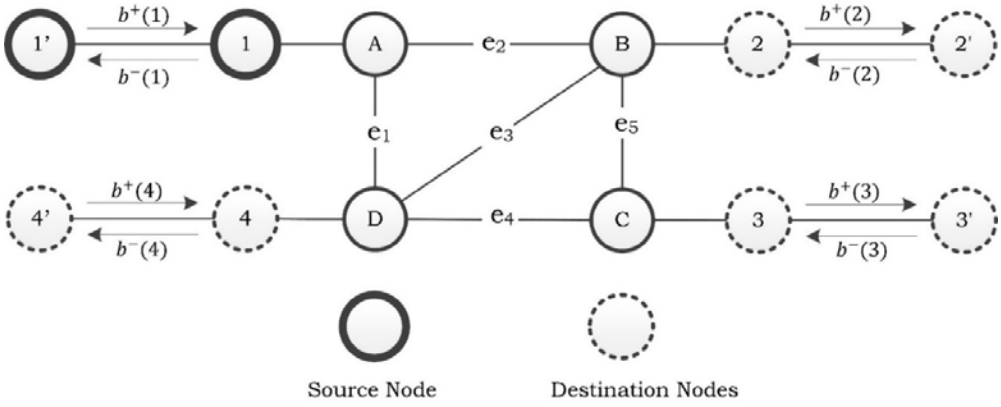

Figure 5.2: Single source to destination pairs. (Partial Mesh)

To evaluate the throughput, the demands are configured as bidirectional pairs traversing end to end LSPs with the return path instantiated at the same time as the forward path.

5.2.2 Test Case 1 – Optimisation Objective Functions

Minimum Routing Cost

The following procedures evaluate the throughput performance of the SDN application. The implementation of both the Min-Cost and Min-Max objective functions are compared as a function of throughput on the full mesh and partial mesh topologies.

```
Throughput: Demand increase min-cost (V, E, c, d);
        Ce = 1Gbps; d={demand pairs};
        p={k-shortest paths};
        h=0bps;   //Initial demand
        DV_increment = 100Mbps;
        while min-cost < infinity do
                h += DV_increment;
                x_mc = min_max(h, d, p);
                while PCEP.create != done;
                        PCEP.create(x_mc, p);
                end;
        end;
end;
```

Figure 5.3: Throughput evaluation as a function of demand (Min-Cost approach).

Figure 5.3 starts by initialising the link capacities and demand pairs as well as finding k-shortest paths between all demand pairs. Feasible flows are calculated using the Min-Cost routing function based on the demand volume h, demand pairs d and the set of k best shortest paths linking the (s,t) pairs in d.

Minimising the Maximum Link Utilisation

Similarly, Figure 5.4 finds the feasible flows using the Min-Max objective function with inputs h, d and p. In both cases, the loop repeatedly increases the demand volume h (bps) by a constant factor *DV_increment* after which the PCEP module is called to create the LSP's for the feasible flows.

```
Throughput: Demand increase min-max (V, E, c, d);
        Ce = 1Gbps; d={demand pairs};
        p={k-shortest paths}; h=0bps;
        while min-max < infinity do;
                h += DV_increment;
                x_mm = min_max(h, d, p);
                while PCEP.create != done ;
                        PCEP.create(x_mm, p);
                end;
        end;
end;
```

Figure 5.4: Throughput evaluation as a function of demand (Min-Max approach).

5.2.3 Test Case 2 – Multi Time-Window Hose Model Optimisation

Table 5.1 illustrates the multi time-window demands. For the four time-windows, demand pairs $d_{(s,t)}$ represents demand from source node s to destination node t.

To accommodate the changing demand, a Hose is implemented, where the total output bandwidth, BW_{out} is defined as the upper bandwidth over a time window. The BW_{out} and BW_{in} can now be scheduled on a per time window basis for the known demand requirement.

Table 5.1: Multi-time demand allocation.

Multi-time window (Hrs)	(Mbps)	(Mbps)	(Mbps)	(Mbps)	(Mbps)	(Mbps)	(Mbps)	(Mbps)
	$\boldsymbol{BW_0^{out}}$		$\boldsymbol{BW_1^{out}}$			$\boldsymbol{BW_2^{out}}$		$\boldsymbol{BW_3^{out}}$
	$d_{(0,1)}$	$d_{(0,2)}$	$d_{(1,0)}$	$d_{(1,2)}$	$d_{(1,3)}$	$d_{(2,1)}$	$d_{(2,0)}$	$d_{(3,1)}$
0	50	50	50	50	50	50	50	50
1	100	150	100	100	100	50	50	50
2	100	100	50	50	0	100	100	100
3	150	100	50	50	50	100	150	50

The pairs, $d_{(s,t)}$ and $d_{(t,s)}$ form the transmit and receive demands between two sites. BW_n^{out} is the sum of all the demands from node n to other destination sites. The demand volume for a demand pair can be split over multiple paths during a time window.

As demand changes, the network responds automatically to the changes in demand by increasing or decreasing the bandwidth of the affected LSPs while respecting the upper bandwidth of the symmetric hose formed between the node servicing the demands and the node that is part of the core network.

Thus, the maximum bandwidth of the hose is restricted by the capacity of the link between these two nodes, which is 1Gbps

The multi-time window represents a period over which demand uncertainty are accommodated for by observing the ingress and egress upper bandwidths $b^+(s)$ and $b^-(s)$ such that $\sum_{t:t\neq s} d_{s,t} \leq b^+(s)$, $\forall s \in Q$, and $\sum_{t:t\neq s} d_{t,s} \leq b^-(s)$, $\forall s \in Q$.where Q is the set of edge nodes.

The traffic is specified as the total outgoing or incoming traffic from or to the node, where $b^+(s)$ is the maximum rate of traffic that node s can send into the network.

The traffic model that is bounded by $b^+(s)$ and $b^-(s)$ is the hose model. The traffic demand between each source-destination pair does not need to be specified.

Figure 5.5 shows the implementation of the hose on the egress traffic for the four nodes in the mesh topology.

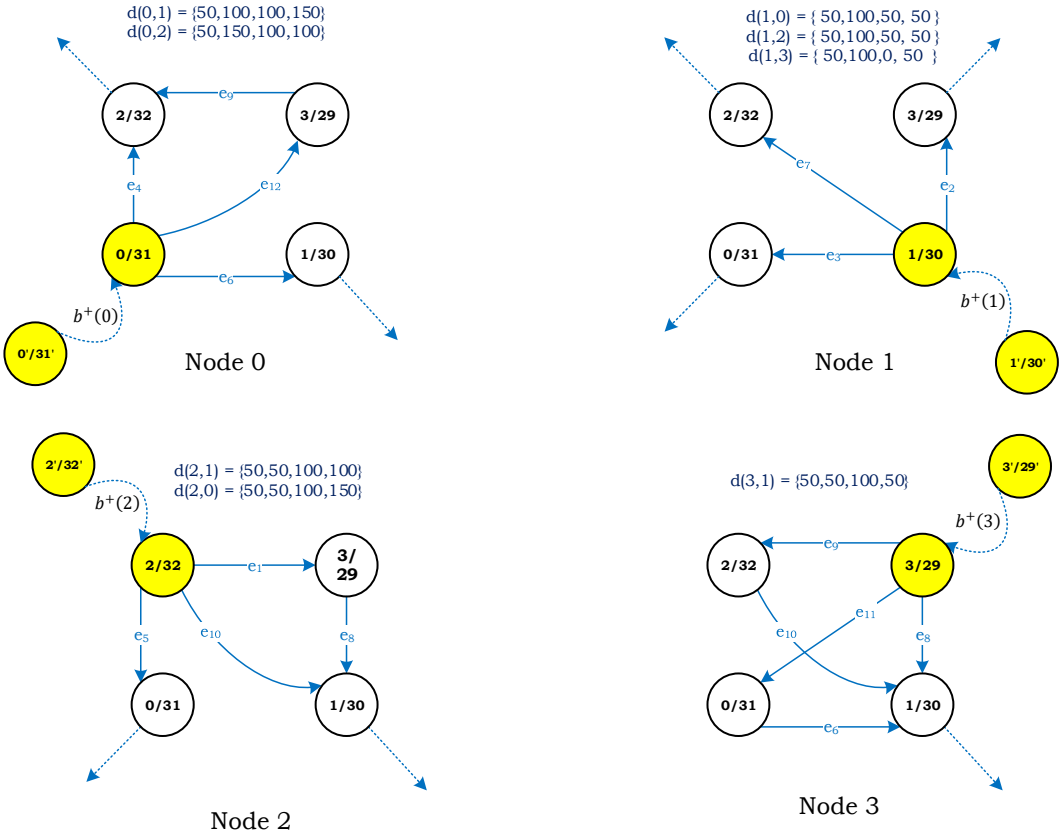

Figure 5.5: Hose implementation on egress traffic.

The hose model is implemented in the network application. The hose bandwidth is then reserved in the network by LSP establishment. As the demand varies over the different time windows, the reserved bandwidth is adjusted automatically.

Unused bandwidth is released back to the network when the reservation bandwidth is decreased over the time intervals.

5.2.4 Test Case 3 - Total Network Reservation

The next section will investigate how demands have been allocated to paths. Path utilisation refers to how much demand has been allocated to each path.

Table 5.2 illustrates the allocation of demand $d_{(s,t)}$ to the paths connecting node *s* to node *t*. Four time-windows are specified per (*s, t*) demand with their demands as indicated. During each time window, these demands changes and the optimiser must recompute optimal paths for these demands to be routed onto.

The number of paths between node *s* and *t* is controlled by the variable *k*. In this case $k = 5$.

Table 5.2: Multi-commodity Flow Allocation.

(s, t)	Source To Destination Path		Min-Max				Min-Cost			
			TW_0	TW_1	TW_2	TW_3	TW_0	TW_1	TW_2	TW_3
(1,0)	(30 : 31)	x[0][0]	50	100	50	50	50	100	50	50
(1,2)	(30 : 32)	x[1][0]	50	100	50	50	50	100	50	50
(1,3)	(30 : 29)	x[2][0]	50	100	50	50	50	100	50	50
(2,1)	(32 : 30)	x[3][0]	50	50	100	100	50	50	100	100
(2,0)	(32 : 31)	x[4][0]	50	50	100	100	50	50	100	150
(2,0)	(32 : 29), (29 : 31)	x[4][2]	0	0	0	50	0	0	0	0
(3,1)	(29 : 30)	x[5][0]	50	50	117	50	50	50	150	50
(3,1)	(29 : 31), (31 : 30)	x[5][1]	0	0	17	0	0	0	0	0
(3,1)	(29 : 32), (32 : 30)	x[5][2]	0	0	17	0	0	0	0	0
(0,1)	(31 : 30)	x[6][0]	50	100	100	100	50	100	100	150
(0,1)	(31 : 29), (29 : 30)	x[6][2]	0	0	0	50	0	0	0	0
(0,2)	(31 : 32)	x[7][0]	50	100	100	100	50	150	100	100
(0,2)	(31 : 29), (29 : 32)	x[7][2]	0	50	0	0	0	0	0	0

The optimiser is presented with five paths for each source to destination pair.

5.2.5 Test Case 4 – Increasing Path Diversity

This section describes the evaluation of the maximum throughput for the two objective functions in the context of HSDN with the following scenarios:

- Link-path formulation requires a predetermined set of candidate paths as input. The number of paths, governed by the value *k*, is varied. This has the effect of finding shortest paths from source to destination equal to the value of *k*. For $k = 1$, only one shortest path from source to destination is found. For $k = 2$, two paths are found and so forth.

- An attempt is made to find an optimal value for *k* that will maximise the throughput.
- Demand is bidirectional and is accommodated by configuring LSPs between the source node and the destination node in both directions.
- The LSP bandwidth is varied, where the LSP bandwidth has pre-established values {32, 128, 256, 512} Mbps. The impact of the increased LSP bandwidth is then assessed.
- The Min-Max objective functions are evaluated in terms of its impact on the maximum throughput.

The implementation of Min-Max routing, equation 3.27 is evaluated by varying the number of alternate *k*-shortest paths, where *k* is the number of shortest paths between a source and destination pair while observing the throughput.

5.2.6 Test Case 5 - Varying LSP Bandwidth, Topology and Optimisation Method

In Figure 5.6, the initialisation process sets the demand pairs *d* to a single (*s, t*) pair with demand volume *h* set to zero. The demand pairs are then increased to the maximum demand pairs possible for the network.

At each increase, the Min-Max feasible flow is calculated and configured with the PCEP module.

```
Throughput: Demand pair increase min-max (V, E, c, d);
            Initialisation;
            Ce = 1Gbps;
            d={one demand pair};
            p={ k=5 shortest paths};
            h=0;
            while d < maximum demand pairs for G do;
                  d += demand pairs D;
                  while min-max < infinity do;
                  h += DemandVolume_increment;
                  x_mm = min_max(h, d, p);
                        while PCEP.create != done ;
                              PCEP.create(x_mm, p);
                        end;
                  end;
            end;
      end;
```

Figure 5.6: Throughput as a function of demand variation.

The demand pairs originating from a single source is increased at the destination side only resulting in a single source with multiple destinations.

For every iteration, Figure 5.*7*, the LSP bandwidth between each demand pair is increased at discreet intervals while observing the network throughput.

```
Throughput: LSP bandwidth increase min-max (V, E, c, d);
        Initialisation;
        Ce = 1Gbps;
        d= 8 {demand pairs};
        p={ k=5 shortest paths};
        h=0;{demand}
        LSP_BW={32, 128, 256, 512};
        while LSP_BW do;
                while min-max < infinity do;
                h = LSP_BW;
                x_mm = min_max(h, d, p);
                        while PCEP.create != done ;
                                PCEP.create(x_mm, p);
                        end;
                end;
        end;
end;
```

Figure 5.7: Varying LSP bandwidth.

5.3 Conclusion

Five test cases were described in 5.2.1 to 5.2.6. The experimental work described aims to test the hypotheses and address the objectives of this work as described in 5.1. An effort was made to test the functional blocks of the key features of the auto-bandwidth adjust SDN application system.

The next chapter provides the results for the test procedures described and will elaborate on the interpretation of these results.

6 Results

The scripts of all experimental test cases were written in Java using the Eclipse integrated development environment. The Cisco IOS XRv virtualised router was installed on VMware workstation and hosted on an HP server with Ubuntu 18.04 as the operating system. Section 6.1 gives a detailed description of the hardware, software and tools used, followed by the results obtained from the experimental work performed.

6.1 Software and Hardware Details

6.1.1 Software Description

- **Programming environment**: The procedures of all experimental test cases are written in Java with Eclipse JEE Neon as the integrated development environment.
- **Virtual routers software**: The Cisco IOS XRv6.0.1 virtualised router is used.
- **Virtualiser software**: VMware workstation 15 Pro version 14.1.1 with support for Linux operating system is used.
- **ODL controller version**: Distribution Karaf-0.5.4-Boron-SR4 is used.
- **Operating system**: Ubuntu 18.04.02 Server with long term support is installed with Bionic Beaver as the GUI.

6.1.2 Hardware Description

- **Server model**: HPE ProLiant ML 150 Gen 9 server tower.
- **Processor**: Genuine Intel Xeon CPU E5-2609 v3 at 1.90GHz, family 6
- **Memory**: 8 x 8GB DIMM DDR4, 2133MHz (64GB RAM total).

6.1.3 Other Software Tools Used

- **Named pipe TCP proxy**: A utility that provides access to a named pipe for windows.
- **Secure CRT**: For remote router access from a windows machine.
- **Socat command-line utility**: For router access from an Ubuntu terminal. Used to transport data between two points. In this case, from VMware named pipe to Ubuntu terminal.
- **Postman**: For Windows version 7.1.1 is a GUI based API Development Environment for testing API commands.
- **Optimiser**: IBM ILOG CPLEX Optimisation Studio Community Edition (64 bit) 12.7.0.

6.2 Evaluation of SDN Application and Control Process.

The following section will evaluate the results for the test cases described in chapter 5. An attempt was made to test the plausibility of hypotheses I and II and address the research questions.

6.2.1 Test Case 1 – Optimisation Objective Functions

In performing this test case, an attempt is made to address hypothesis I, section 2.3.1. Whether a legacy MPLS network, updated with new SDN features, to form an HSDN network, can dynamically adapt to changes in demand. To this end, test case 1 was developed.

LSPs are created based on demand, and as that demand changes over various periods, the control process responds by making optimised adjustments to the LSP configuration of the network. Two objective functions were used to optimise the LSP placement over the edges of the network. The objective functions were minimising the maximum link utilisation and minimising the routing cost. The results of this test now follow.

Minimising the Maximum Link Utilisation

Minimising the maximum link utilisation for load balancing seeks to distribute the demand over as many links as possible, 3.4.2, without overloading one link.

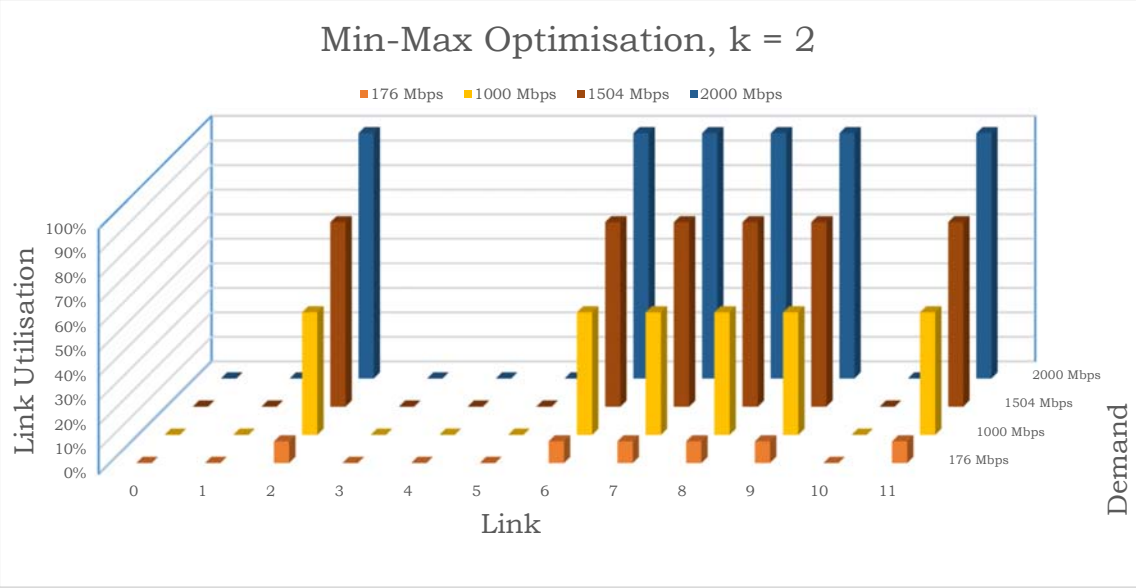

Figure 6.1: Bandwidth distribution – Full Mesh, $k = 2$.

Minimising the maximum link utilisation, Figure 6.*1*, shows that if only two alternate shortest paths, $k = 2$, between source and destination pair are identified, that load balancing is not effective since the optimiser has only two paths to consider. As a result, links 2,6,7,8,9 and 11 are utilised whereas links 0,1,3,4,5 and 10 are not used at all. This remains the case, even as the demand goes from 176Mbps to 2000Mbps.

This result is expected because of the lack of diverse paths used as input to the optimisation process, resulting in all the LSPs being configured on the same links in the network. This results in up to 100% congestion on some of the links while other links are not utilised at all.

The task is to minimise the link utilisation over the links on which the demands are routed for the given set of demands D, such that for any demand $d \in D(i)$, the demand for each commodity $(s,t) \in H$, is routed over no more than k-shortest paths in P_{st}. The set of paths that uses link e is denoted by P_e. The total demand over link e is therefore $L_d^e = \sum_{p \in P_e} B(p)$, where $B(p)$ is a function that returns the demand transmitted on path p given the demand matrix $\mathbf{t} = (t_{ij})_{i \in S, j \in S \setminus \{i\}}$.

As the number of alternative paths between source and destination is increased to k = 5, the overall link utilisation is reduced as shown in Figure 6.*2*, because of the increase in path diversity. For the same demand, say 1000Mbps, the link utilisation for k = 2 in Figure 6.*1* goes from 50% to 33% for k = 5 in Figure 6.*2*.

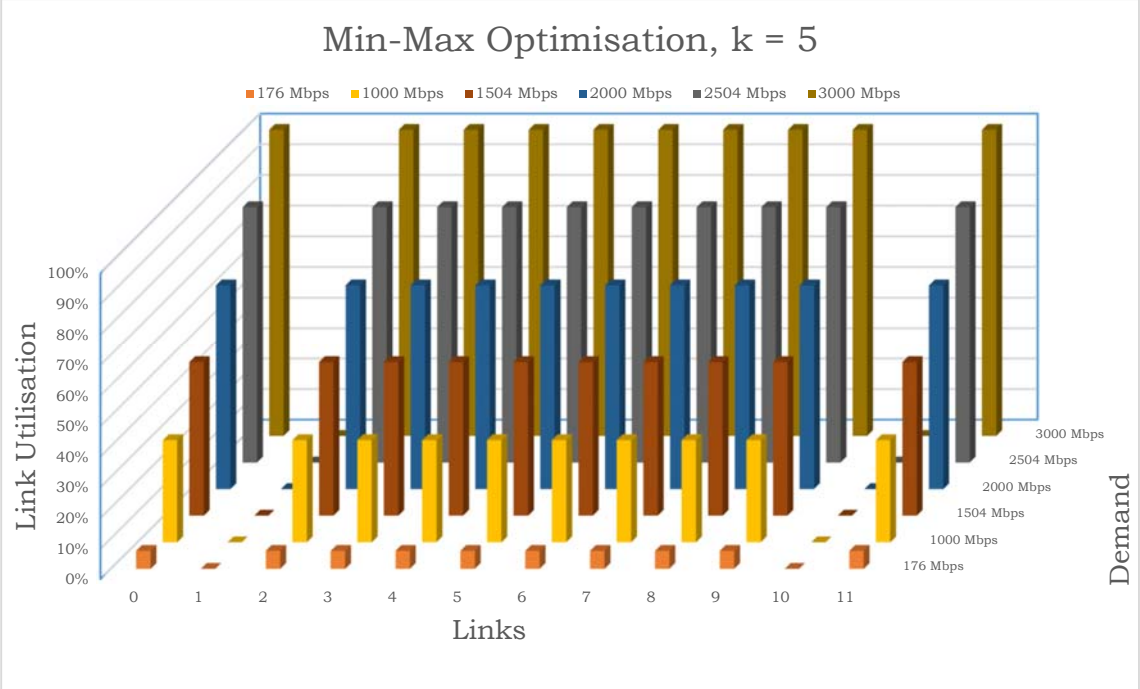

Figure 6.2: Bandwidth distribution. (Full Mesh, k = 5).

The link utilisation per demand remains evenly balanced between the links even as the LSP bandwidth increases from 176Mbps to 1000Mbps and finally to a maximum demand of 3000Mbps. Here the links used are links 0, 2, 3, 4, 5, 6, 7, 8, 9 and 11 with only two links, links 1 and 10, not being used. As noted before, [101], and also discussed later in 6.2.4, there is no increase in throughput for values of k > 5, hence throughput for values of k > 5 were not considered.

As observed in Figure 6.*1* for k = 2, and Figure 6.*2* for k = 5, the overall maximum link utilisation is reduced especially for values of k > 2. Demands are routed over multiple candidate-precomputed paths. Since the objective function is to minimise

the maximum link utilisation, the increase in candidate paths produces an optimal solution that reduces overall link utilisation.

Minimum Routing Cost

For the minimum routing cost objective function, the per-link cost of all links in the network is set to 1. Since the hop-length-based *k*-shortest path algorithm is used, setting the link cost to 1 is enough to determine candidate paths. The *k*-shortest paths can be generated using just the hop count to reflect cost on a link. The routing-cost will, therefore, tend to find an admissible routing that routes commodities over shorter distance paths to minimise the routing cost.

The minimum routing cost multipath routing implementation shown in Figure 6.*3* allows the unicast demands to be split over multiple shortest paths through the network.

Since the optimisation criteria seek only to optimise the distribution of serviceable demands based on the shortest source-destination path through the network, even though there are two paths, since $k = 2$, there is an over utilisation of links 9 and 10, when the demand is 176Mbps. These two links make up the shortest source-destination path in the network, while the other links making up the second path have zero utilisation. Even when the demand is 1000Mbps, resulting in a link utilisation of 100%, indicated with the yellow bars, only links 9 and 11 are used. It is only when the demand is 1504 Mbps, and the shortest path, links 9 and 11 are at 100%, that the link making up the second source-destination path is used, indicated by links 2, 6, 7 and 8.

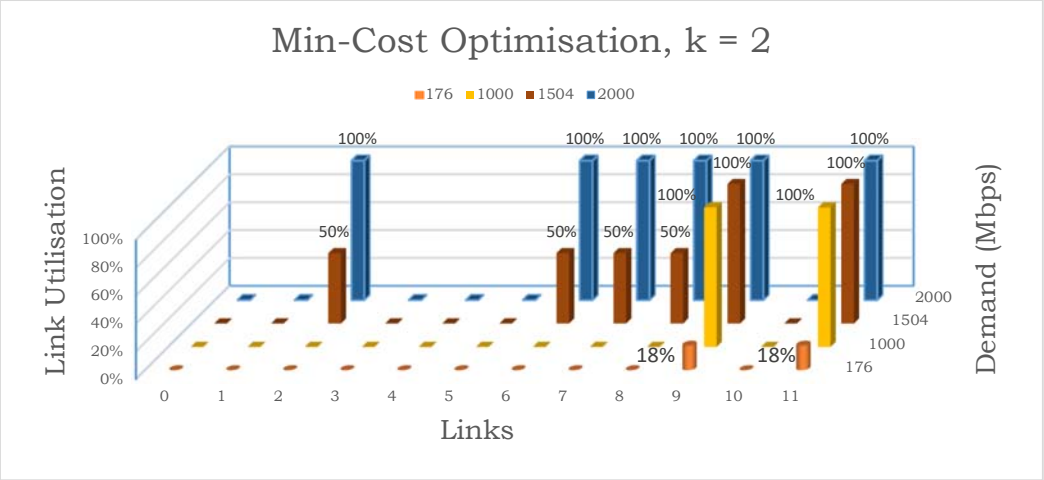

Figure 6.3: Bandwidth distribution – Full Mesh, $k = 2$.

For $k = 5$, the optimisation process fully capacitates the 1st shortest path before utilising the 2nd shortest path. The process continues until the 5th shortest path is capacitated. This leads to links with bottlenecks while other links have not been capacitated at all. As the number of precomputed candidate paths are increased to $k = 5$ as shown in Figure 6.*4*, the optimal multi-path routing solution achieved using

the minimum routing cost objective function provides no improved solution in terms of overall maximum link utilisation.

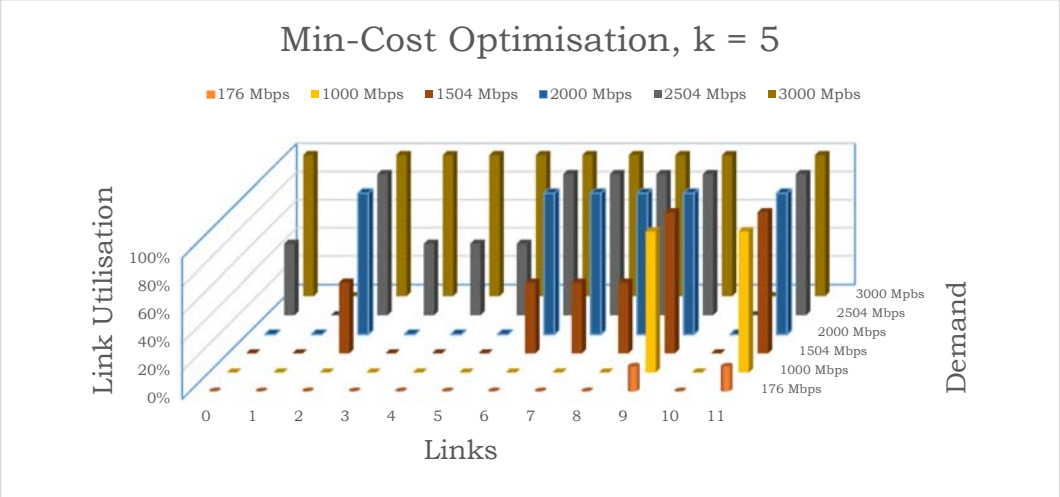

Figure 6.4: Bandwidth distribution - Full Mesh, $k = 5$.

From the observations made in Figure 6.*3* and Figure 6.*4*, bandwidth management of the overall network using Min-Cost optimisation is less affected by the value of k for reducing bottlenecks in the network. However, the observations made in Figure 6.*1* and Figure 6.*2* using Min-Max optimisation shows that increasing the value of k has a much bigger effect on reducing bottlenecks in the network.

Thus, if the objective is primarily to service time-sensitive demands where arrival times of packets over different path lengths have tight arrival time margins, then using the Min-Cost criteria even with a network where path diversity is reduced, i.e. $k = 2$, would be enough. This comes at the cost of the over utilisation of some shortest paths' links between the source and the destination. However, for less time-sensitive demands, the min-max implementation offers reduced link utilisation throughout the network as more alternate paths are used to service the demands.

In Summary:

Test case 1 sought to address hypothesis I, (2.4.1), as well as research question (2.3.1). There is a requirement to show that a legacy IP/MPLS network, converted to an HSDN network, can respond dynamically to time-varying demand. In chapter 4, the implementation details the implementation of the required protocol such as PCEP and BGP-LS with a detailed description of the HSDN traffic engineering architecture in (4.3). The full control loop, (4.5), shows the final implementation of the HSDN in the context of the control loop.

This serves to demonstrate that, the legacy IP/MPLS network was converted to a hybrid SDN network, and that the protocols that facilitated this process, was the PCEP and BGP-LS protocols. The SDN controller is a key component that was also implemented. By implementing the two different optimisation functions, and then sequentially increasing the demand, the results demonstrated that the network can be reorganised efficiently based on the implemented objective function.

An argument can be made that only single-objective function optimisation was implemented and that multi-objective function optimisation could produce better results in terms of throughput. However, the test aimed to illustrate that, the legacy IP/MPLS network, converted to an HSDN network, can adapt dynamically to varying demand.

To that end, the two different optimisation functions illustrated the reorganisation of the LSP placement on the edges of the network. As the source-destination demand is increased, the change required reoptimisation of the network and LSP placement. LSP reconfiguration is a function of the PCEP and PCE protocols. This too is evident in Figure 6.1, Figure 6.2, Figure 6.3 and Figure 6.4, where, because of the demand increases, the link utilisation differs based on the optimisation function and available source-destination path diversity k.

6.2.2 Test Case 2 – Multi-Time Window Hose Model Optimisation

The multi-time window represents the change in traffic demand over four time periods. This scenario is typical of fluctuating demand over 24 hours, with morning, midday, evening and midnight traffic as an example of four possible periods, where the fluctuation in traffic is defined a priori.

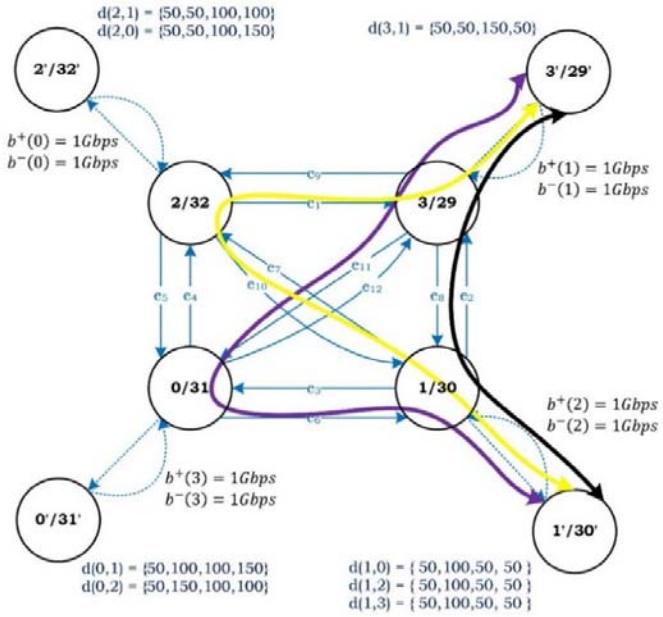

Figure 6.5: Demands originating at node 3.

Figure 6.5 shows the full mesh topology with demand pairs originating at node 3. The demand volume allocated to each path during the first time-window, TW_0 is 50Mbps followed by 50Mbps, 150Mbps and 50Mbps for TW_1, TW_2 and TW_3 respectively. During each time-window, the LSP bandwidths are adjusted. To accommodate the changing demand, a hose is implemented, where the total output bandwidth, BW_{out} is defined as the upper bandwidth over a time window.

The BW_{out} and BW_{in} can now be scheduled on a per time window basis for the known demand requirement. The hose model is used as discussed in 1.6.1 and 3.7.1, where the sum of all egress demands for the site $b^{+}(v)$ and ingress demands for the site $b^{-}(v)$ must not exceed the capacity of the link of 1Gbps for all the $e \in E$.

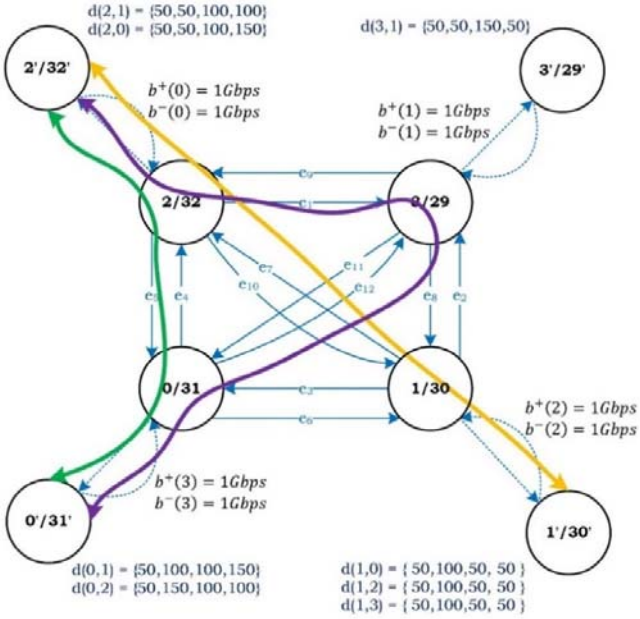

Figure 6.6: Demand originating at node 2.

Figure 6.6 illustrates the paths used to service the demands originating at node 2. The demand volume for d (2, 0) is split over two paths shown as the green and purple paths.

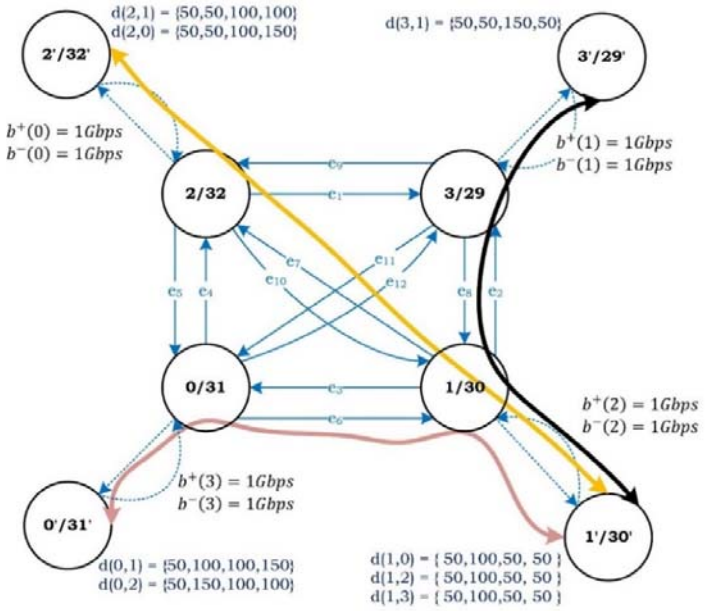

Figure 6.7: Demands originating at node 1.

Figure 6.7 illustrates the paths for the demands originating at node 1. The Min-Max optimiser aims to find routes that balance the demand over the links of the network, but this does not result in the demand being balanced over all the paths of the network.

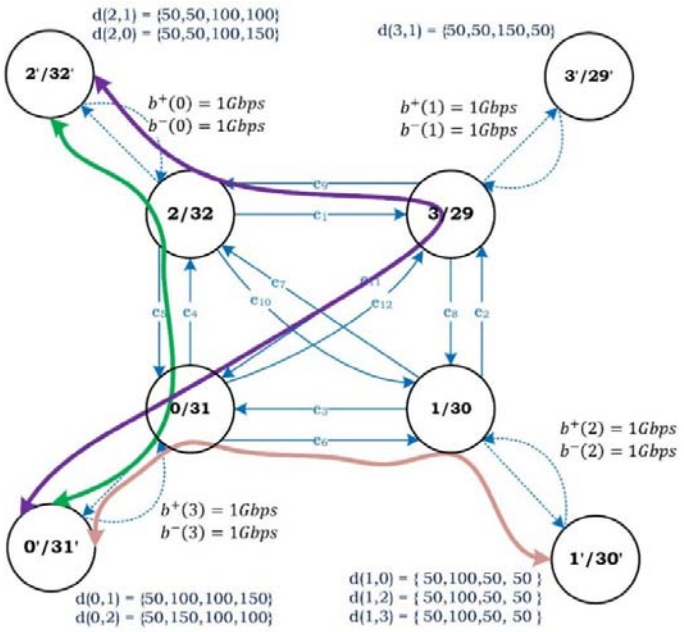

Figure 6.8: Demands originating at node 0.

Similarly, Figure 6.8 uses two paths to service the demand volume for the $d(0, 2)$ demand pair.

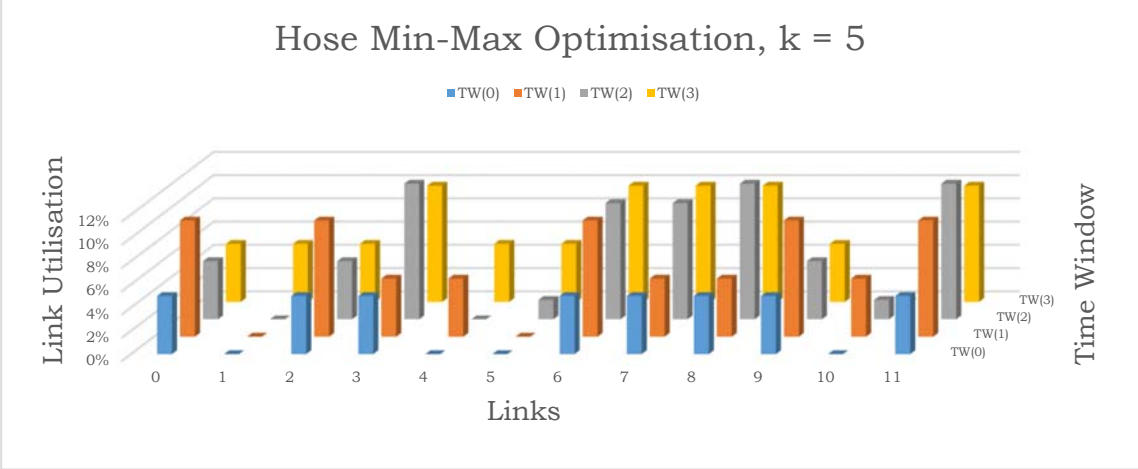

Figure 6.9: Multi-time demand link utilisation.

As these demands vary from TW(0) to TW(3), the route optimiser tends to balance the demand over the available links of the network that makes up the k = 5 source-destination paths as shown in Figure 6.9. Each time-window represents a new set of demand requirements that is automatically balanced over the links of the network by the optimiser.

In Summary:

Test case 2 sought to further address hypothesis 1, (2.4.1), as well as research question 0. The concept of time-window is introduced, where it refers to the change of demand over various periods. To this end, four time-periods were observed. In addressing research question 0, it must also be demonstrated that there is the continuous optimisation of the network as the demand varies over time.

That the network can adapt to demand has already been demonstrated with the results shown in (6.2.1). By using the objective function that minimises the maximum link utilisation, and for a single time window, Figure 6.5, Figure 6.6, Figure 6.7 and Figure 6.8 shows how paths have been allocated to demands originating at various nodes. This process is performed by the PCE.

Figure 6.9, where the concept of the time window is introduced, illustrates a continuous process of optimisation and network reconfiguration as the demand varies over the four time-windows. This is evident from the changes that link loading that can be observed between the time windows.

This result seems to suggest that there is the continuous optimisation of the network, even as the demand varies over the time windows. and that the HSDN can reorganise its LSP placement in response to varying demand.

6.2.3 Test Case 3 – Total Network Reservation

This section shows how the automatic bandwidth adjust control loop responds to these changes in demand by dynamically adjusting LSP bandwidths. The hose model, described in sections 1.6.1 and 3.7.1, is used for this implementation.

The following results evaluate the automatic adjustment of LSP bandwidth as demand levels increases and decrease. During a time-window period, all demands remain the same for that period. The demand only changes over the next time window. The network then adjusts for the change in demand as well as release the unused bandwidth back to the network

Table 6.1: Network reserved bandwidth.

	Hose BW out				Network reserved BW $[Mbps]$
	BW_0^{out} $[Mbps]$	BW_1^{out} $[Mbps]$	BW_2^{out} $[Mbps]$	BW_3^{out} $[Mbps]$	
TW_0	100	150	100	50	400
TW_1	250	300	100	50	700
TW_2	200	100	200	100	600
TW_3	250	150	250	50	700

From Table 6.1, BW_s^{out} $[Mbps]$ is the total bandwidth of all the LSPs originating at node $s \in V$ for the (s,t) pairs. TW_n is the time-window under consideration during period n.

The *Total Bandwidth Reservation* on the network during time-window n, denoted by $TBR(TW_n)$ is a measure of how much of the total available network capacity has been reserved for VPN purposes and is expressed as:

$$TBR(TW_n) = \sum_{e \in E} BR(e^{TW_n}), \text{where } BR(e^{TW_n}) \leq c_e \; \forall \; e \in E, n = 0, 1, 2, 3 \qquad (6.1)$$

The automatic bandwidth adjustment control loop (Figure 4.*8*, pg. 68) performs the function of dynamically adjusting the LSP bandwidths, as the demand changes over time. As demand between the source and destinations pairs varies over these time windows, the LSP bandwidth adjusts dynamically in response to the demands.

Specific time windows of demand have been specified as shown in Table 6.1. The total *Bandwidth Reserved (BR)* on hose link $e \in E$ for the VPN is denoted by $BR(e)$. .

Because a multi-time approach is used, the $BR(e)$ value changes in response to the time-window of demand as shown in Table 6.1. The $BR(e)$ value associated with time-window n will then be denoted by $BR(e^{TW_n})$, which is the total bandwidth reserved on edge e, TW is the time-window and n represents the 4 time-windows, $n = \{0, 1, 2, 3\}$.

Using Min-Max routing, the reservation for each link produced a multi-time window total bandwidth reservation, $TBR(TW_n)$ shown in Figure 6.*10*.

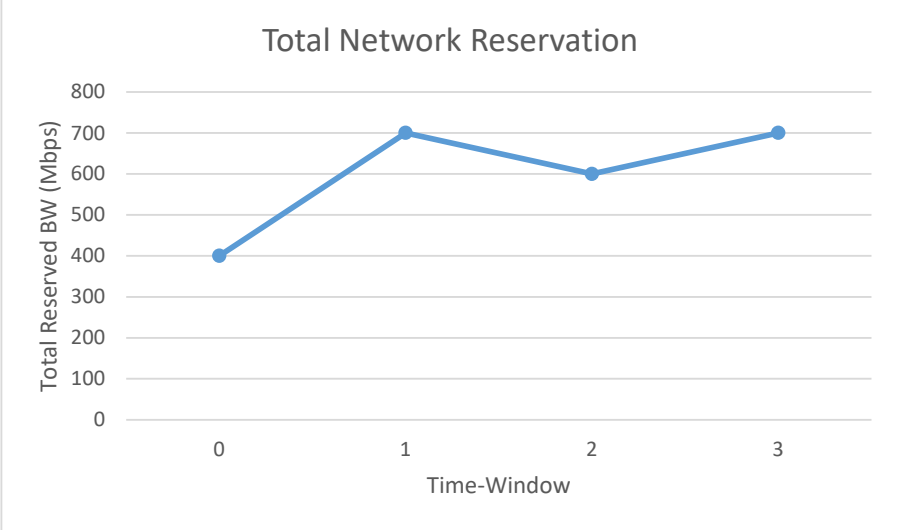

Figure 6.10: Total network reservation.

As the TBR value changes with every time-window, reserved bandwidth is released back to the network by the automatic bandwidth adjust control loop, for the demand pairs where the demand defined over that period has decreased.

This results from the affected LSP BWs being decreased dynamically in response to the demand. For that same period, sites experiencing a higher demand can then reserve more bandwidth by dynamically increasing the LSP BWs for the LSPs affected by the increase in demand.

The hose implementation allows for bandwidth to be released to the core of the network from nodes experiencing lower demand requirements. This is done by adjusting the reserved symmetric BW_{out} and BW_{in}hose limitation. The demand is then redistributed to nodes where the demand requirement is higher.

In Summary:

Test case 3 sought to address hypothesis II, (2.4.2). The main purpose of this test case was to illustrate that unused bandwidth, which results from overprovisioned LSPs, are returned to the network. To that end, the total reserved bandwidth, across the HSDN network is observed for the four time-windows.

Where demand has dropped, the LSP bandwidth decreases, releasing bandwidth back to the network. Similarly, as demand increases, the LSP bandwidth needs to increase automatically in response to the increase in demand. This is evident from Figure 6.10, where the total reserved bandwidth varies over the periods shown. This is interpreted as resulting from the LSP bandwidths being adjusted in response to the demand for that time window.

LSPs are configured in the HSDN based on current demand. As demand varies over time windows, the LSPs servicing these demands adjusts dynamically to accommodate these changes.

6.2.4 Test Case 4 – Increasing Path Diversity for Single Source

By increasing k, the number of shortest paths between the single source to multiple destination pairs is increased. By increasing the number of distinct paths between the source-destination pair, Figure 6.11, there is an increase in throughput for values of $k \leq 5$.

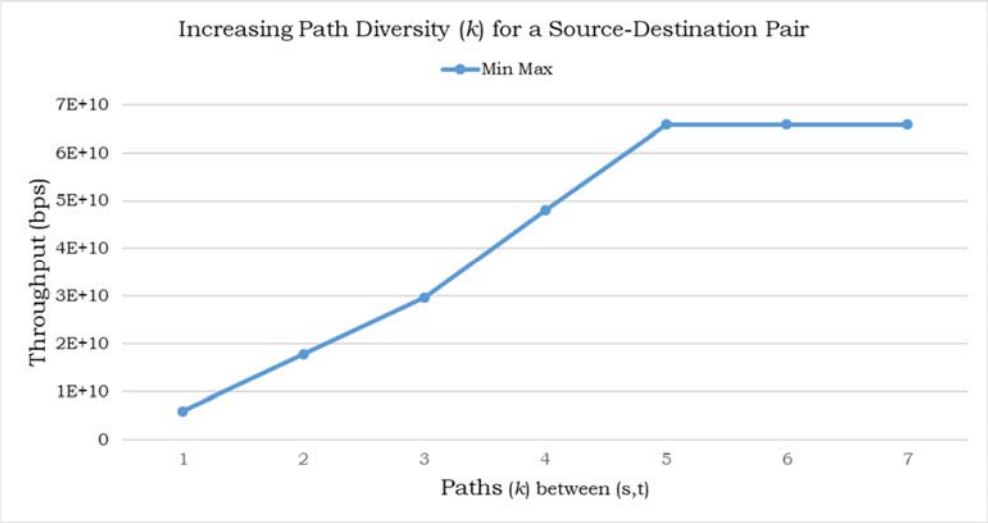

Figure 6.11: Increasing path diversity.

There is no improvement in throughput for values of $k > 5$. In other words, even as the number of alternate paths between the source-destination pair is increased

beyond five paths for the full mesh topology network, the throughput remains the same. For most moderate to large networks, generating 5-10 candidate paths per demand pair is sufficient and on a 50 node network take only a few seconds to compute on an off-the-shelf computer [101].

The (*k-shortest Path*) KSP algorithm not only finds the first KSP, which is the shortest path, but also each other shortest path up to k paths. For this small 4-node full mesh topology used, there simply is not enough links in the network for the KSP algorithm to find more source-destination shortest paths, than $k \leq 5$.

In Summary:

Test case 4 sought to address the research question (2.3.3). The test established a relationship between throughput and source-destination path diversity. Though for this four-node full mesh topology, $k = 5$ was found to be the upper limit beyond which throughput no longer increased, this result is limited to this specific topology.

6.2.5 Test Case 5 – Varying LSP Bandwidth, Topology and Optimisation

In Figure 6.12, the throughput of the network is observed for, varying topologies, varying LSP bandwidth and the optimisation method used.

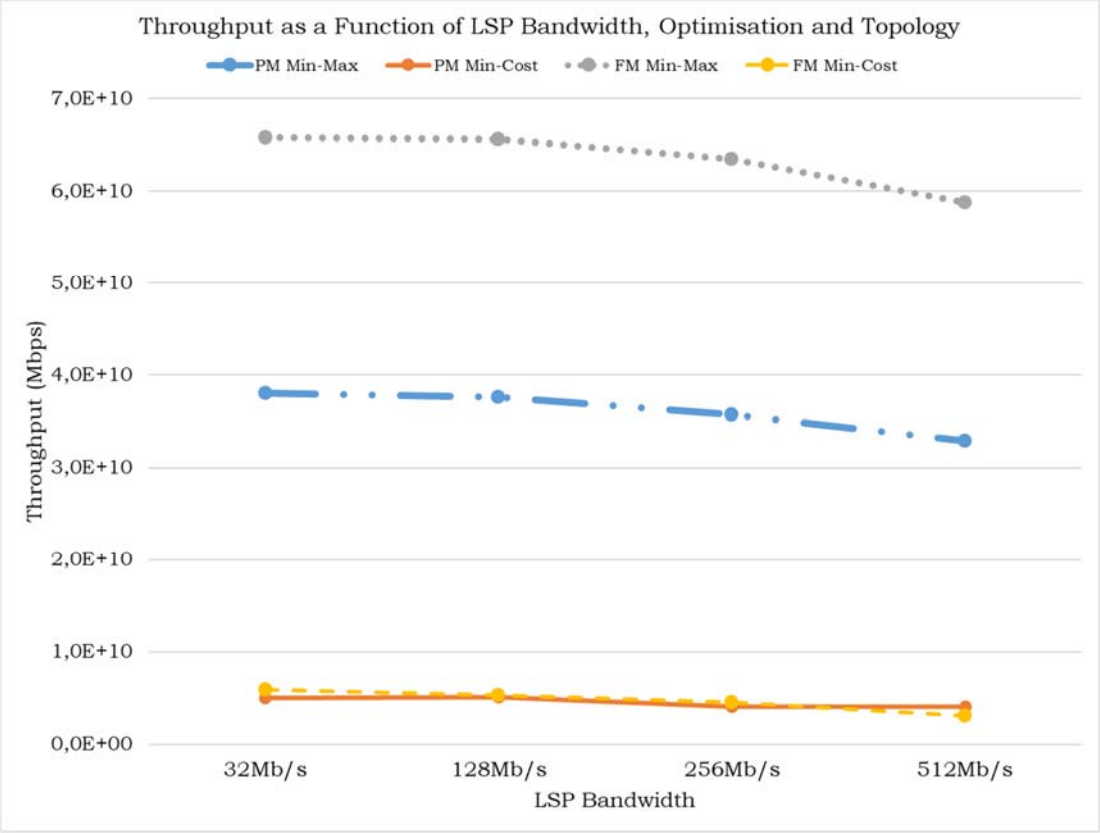

Figure 6.12: Throughput Observation.

Varying the Network Topology:

From Figure 6.12, the throughput for the mesh topology is higher than that for the partial mesh topology as shown by the *FM Min-Max* legend line. This is however only true when minimising the maximum link utilisation optimisation method is used.

When the same topology is used, there is however a decrease in throughput when Min-Cost routing is used as opposed to Min-Max routing.

Varying the LSP Bandwidth:

There is a decrease in maximum throughput as the bandwidth of the LSPs is increased. LSPs with larger bandwidths, 256Mbps and 512Mbps, are less likely to route through the core network resulting in a larger portion of the core network being underutilised resulting in an overall drop in throughput. LSP's with smaller bandwidths, 32Mbps and 128Mbps are more likely to route since smaller flows can route onto smaller portions of excess core network bandwidth

Optimisation Methods:

Topology plays some part in increasing the network throughput, with the full mesh topology having the highest throughput of the two topologies considered as indicated with the *FM Min-Max* legend line in the figure. However, combining the topology with the optimisation process, in this case minimising the maximum link utilisation, produced the highest throughput. Even when the same optimisation process is applied to the part mesh topology, the throughput was the second highest as shown by the *PM Min-Max* legend line in Figure 6.12.

The impact of the optimisation method is so drastic, that even when the Min-Max optimisation method is applied to the partial mesh topology it outperformed the full mesh topology throughput figure when Min-Cost optimisation is used.

In Summary:

Test case 6 sought to address research question (2.3.3). The results demonstrate the interplay between the topology architecture and the optimisation method used. Both impact throughput, with the best combination being a full mesh topology using the objective function that minimises the maximum link utilisation. Additionally, as the LSP bandwidth in increase, there is a decrease in throughput.

Overall, the results seem to suggest that by using smaller bandwidths, in the range of 32-128Mbps on a full mesh topology where the objective function that minimises the maximum link utilisation is used, will produce the highest throughput. However, even when the partial mesh topology is used, the throughput can still be maximised by configuring LSPs with smaller bandwidth and using the objective function that minimises the maximum link utilisation.

The next section is an overview of the objectives of this thesis and how the work presented speaks to those objectives.

6.3 Overview of Test Cases

The following section captures the thesis objectives of chapter 2 and the extent to which they were achieved.

6.3.1 Research Objectives

Objectives – Hypothesis I	**Addressed in this work**	**Extend to which objectives were achieved**
Development Work for Hypothesis I • Develop two testbeds. One with a partial mesh topology and one with a full mesh topology. • The IP/MPLS network will contain single-vendor devices. Southbound protocols PCEP and BGP-LS will be implemented.	Yes	**The focus of this work was on the development work for hypothesis 1 described in 2.5.1.** • Two virtual topologies were implemented which formed the legacy IP/MPLS network portion of the testbed. • The other components of the testbed were the SDN controller and the PCE software. • All required southbound protocols, which came as part of the virtual router operating system were implemented successfully. Overall all implemented protocols worked as expected. • The IP/MPLS network, do have a reduced data plane, which restricts source-destination data transmission. • Even with this restriction, the testbed proved very useful in that all routing protocols were unaffected by this restriction. • The contribution of the PCEP and BGP-LS protocols formed the cornerstone for all the test cases developed. The PCEP protocol enabled remote configuration and tearing down of LSPs, which is key to making the legacy IP/MPLS network programmable.

Test Case 1 – Optimisation Based on Objective Functions • Test case 1 was developed to test the plausibility of hypothesis 1, see section 2.4.1. and address research question 2.3.1 • The test compares the overall network throughput when the two optimisation processes are performed on a full mesh topology.	Yes	**The focus of this work is hypothesis 1 and research question 2.3.1** • The test needed to prove that the legacy IP/MPLS HSDN network can adapt dynamically to varying demand and that the LSPs can reorganise to time-varying demand. • The Min-Max and Min-Cost objective functions used to organise the logical topology. It showed that based on the objective function, the control loop could reorganise the LSPs every time the demand was changed. • At the same time, the results showed that the control loop adapted the LSPs dynamically. • The test evaluates the performance of the optimisation process performed by the PCE.
Test Case 2 – Multi-time Window Hose Model Optimisation • Test case 2 was developed to test the plausibility of hypothesis 1, see section 2.4.1. and address research question 0 • The test observes the total bandwidth reservation over the entire network for every time window. The bandwidth of all the configured LSPs for that time window is summed and expressed as an instantaneous value. • The result will attempt to show how bandwidth is shifted based on the demand at the time.	Yes	**The focus of this work is hypothesis 1 (2.4.1) and research question 0.** • The result shows that the bandwidth reserved changed over time intervals. • This change in reserved bandwidth indicates that the HSDN network did return unused bandwidth to the network. • The SDN features allowed the network to be programmable.

Objectives – Hypothesis II	Addressed in this work	Extend to which objectives were achieved
Development Work for Hypothesis II • Implement a control loop that will dynamically respond to increases in demand. The control loop is essential for automatic dynamic control.	Yes	**The focus of this work was on the development work for hypothesis II (2.4.2) described in 2.6.1** • The full control loop is captured in Figure 4.*8* with the cross-functional flowchart in Figure 4.*16*. • These two figures capture all the building blocks that constitute the development work of the Auto-bandwidth Control loop. • A shortcoming of the control loop is that there is no sensing of the current demand. • This would have required measurement of the current demand with the control loop then responding to that demand. This process is expensive and would have requirement measurement equipment. • Since the control loop does not measure current demand increases and decreases, this shortcoming is addressed by treating the demand changes as well-behaved traffic that can be predicted, with the time-windows reflecting the demand over that period. • For the control loop to be truly dynamic, this feature will need to be developed in future work, with sensors in the network measuring current demand. • This will require a new mathematical approach to optimising the LSP placement. • Even with this measurement shortcoming, the control loop illustrates that the automatic adjustment of LSP bandwidth is possible with demand driving the process.

Test Case 3 – Total Network Reservation • Test case 3 was developed to test the plausibility of hypothesis II, see section 2.4.2. • Bandwidth optimisation within the network eases the bottlenecks due to increase in demand and release bandwidth back to the network when there is no longer a demand. • The test observes the total bandwidth reservation over the entire network for every time window. The bandwidth of all the configured LSPs for that time window is summed and expressed as an instantaneous value. The result will attempt to show how bandwidth is shifted based on the demand at the time	Yes	**The focus of this work is hypothesis II (2.4.2).** • The test aimed to observe if the HSDN IP/MPLS network could release unused LSP bandwidth back to the network. • A key feature of this test was to demonstrate a flexible network, capable of being dynamic in its response to changing demand. • By observing the overall network utilisation in response to changes in demand, the test shows that network utilisation changed as the demand changed. • This change would suggest that there was a change in the configured LSP bandwidths, that this change happened dynamically and that the reconfiguration happened automatically. • Additionally, the demand fluctuations between time windows are abrupt, and not a natural gradual increase in demand. Such granularity can be overcome by reducing the time-window over which the demand change is observed. The discrete changes in demand, however, helps to magnify the routing changes that occur as a result of the optimisation process and how the auto-bandwidth control application responded to it. • In its current form, the process illustrates that automatic reconfiguration is possible on an IP/MPLS network, that has been converted to an HSDN network. • Because of the programmability of the network, the SDN application could adapt the reserved resources on the network in response to variable demand

Additional Objectives Addressing Research Question 2.3.3	**Addressed in this work**	**Extend to which objectives were achieved**
Test Case 4 – Increasing Path Diversity • Test case 4 was developed to address research question 2.3.3. • The objective of the test is to evaluate at which point the variable *k*, the number of candidate paths, no longer has any impact on the optimisation process. • For the HSDN and using the PCE, the test aims to use one of the optimisation methods, and observe how demand is distributed over the links. The hose model is used to ensure that there are always enough network resources.	Yes	**The focus of this work is research question 2.3.3.** • The results, Figure 6.11, illustrates the relationship between path diversity and throughput. • As the number of sour-destination paths is increased, there is a clear increase in overall network throughput. • An upper throughput limit is reached, beyond which point, any increase in the number of source-destination paths has no effect. For the network under test, that value is *k=5*.
Test Case 5 – Varying LSP bandwidth, topology and optimisation method. • Test case 5 was developed to further address research question 2.3.3. • The impact of LSP bandwidth on throughput is observed. • The two objective functions are observed as it relates to throughput. • The full mesh and partial mesh topologies are used to assess the impact that topology has on throughput.	Yes	**The focus of this work is research question 2.3.3.** • This test looks at the number of LSP that can be routed through the network as the LSP bandwidth is increased. • For the two network topologies used, the routing was flexible in adapting to the topological differences. • The MD-SAL maintained a global view of the network because of the forwarding functionality of BGP as well as the IGP in the form of OSPF-TE.

6.3.2 Research Questions

Key Questions	Addressed in this work	Extend to which objectives were achieved
• Can the current legacy MPLS network devices of ISPs, adapt the logical topology dynamically in response to time-varying demand for bandwidth by automatically adjusting LSP bandwidths and routes (2.3.1)??	Yes	• Addressed and discussed in test case 1, section 6.2.1.
• Can the logical topology of current legacy MPLS data networks be optimised continuously, in response to time-varying demand by the integration of HSDN and the PCE into existing data networks (0?	Yes	• Addressed and discussed in test case 2, section 6.2.2.
• For the HSDN network; how is network throughput affected by varying the LSP bandwidth, network topology and optimisation method (2.3.3)?	Yes	• Addressed and discussed in test cases 4 and 5, sections 6.2.4 and 6.2.5.

6.4 Conclusion

The experimental work described aims to test the key features of the auto-bandwidth adjust SDN application. An effort was made to test the functional blocks of the system in relation to the objectives of the research.

The next chapter will present the conclusions drawn from the results of the analysis of the test cases and make recommendations for further research.

7 Conclusions

This research presented a mixed-integer linear programming formulation using the robust minimisation of maximum link utilisation and minimum link cost objective functions, combined with the programmability of the hybrid SDN network to allow bandwidth adaptation to traffic fluctuations in response to time varied demands.

An overarching theme of this research was to address the hypothesis that legacy network equipment can, by the introduction of SDN features into the network, provide benefit to the internet service provider in terms of network automation, data throughput and network programmability.

7.1 Key Research Features

Key features of this research will now be highlighted as well as the research contributions and future research.

7.1.1 Auto-Bandwidth Adjustment

Auto-bandwidth adjustment is addressed by developing a control loop where the reserved bandwidth is adjusted as a function of demand over various time-windows, allowing a stronger match between current demand and reserved bandwidth resources. The aim is to decrease the unused reserved bandwidth when there is low demand.

The network state, as well as knowledge of the actual demands per time-window used, are provided as inputs to the path computation element optimisation process. Based on the path computation element outputs, the LSP bandwidths are adjusted automatically by using the path computation element communications protocol feature of the controller. The features mentioned, *Path Computation Element Communications Protocol* (PCEP), *Path Computation Element* (PCE) and the controller are SDN features enhancing the programmability of the legacy network.

The results in Figure 6.10 demonstrates how total network reservation changes per time window in response to the changes in demand. The results confirm that the control loop provides a means for the network to dynamically reserve bandwidth and that the process is automated.

7.1.2 Releasing Unused Bandwidth

For traffic patterns where the demand matrix is known apriori, the control loop can adjust the reserved bandwidth of the LSP according to the traffic patterns for the various sites. For sites where the demand is low, the LSP bandwidth is decreased resulting in the unused bandwidth being returned to the network.

The benefit that this brings to the service provider is that this bandwidth can then be used to service soured-destination demand at other sites in the network where the demand is higher.

7.1.3 Improvements to MPLS LSP Computation

The MPLS network was enhanced using the SDN in a hybrid configuration, which had the effect of providing a network programmable environment. Optimising the computed routes through the network for the LSPs is heavily reliant on knowledge of the current network state. When the optimisation process is not aware of the current network state, erroneous routes can be chosen as valid paths for the LSP to be configured on.

By using the centralised controller architecture, a centralised view of the network is maintained if the OSPF-TE and BGP-TE protocols are operational. While this introduces a single point of failure, there is a high availability architecture that can be explored in future research, where a standby controller can be tasked to assume control should a failure occur.

The introduction of the PCE allows optimisation features not previously possible with the legacy network equipment. This allows for optimisation based on the network topology and path diversity as illustrated in Figure 6.12.

7.1.4 Testbed Development

For the testbed to achieve the research objectives, key features were introduced. MPLS provides the ideal environment for configuring LSPs through the network, because of the technologies existing LSP configuration capability. This capability was extended through the introduction of SDN features. These features enabled LSP bandwidth adjustment as well as the source-destination rerouting of LSPs through the network.

To perform *Create, Read, Update, Delete* (CRUD) operations on the network, the PCEP functionality was enabled. The centralised path computation model provided a simpler architecture as opposed to distributed path computation with or without collaboration. These testbed elements allow the development of the network application.

7.1.5 Modelling of Network

A network model was developed via the RESTfull interface of the ODL controller. This interface allows the application to capture the state of the network, which could then be processed. The JAVA package JGraph was used for front-end development. The modelling, analysis and visualisation of data were done by the *Java Universal Network/Graph* (JUNG) framework. CPLEX is invoked for optimisation processing.

With the MPLS network being the system, the measurement function is performed by the OSPF-TE protocol operating on the routers, and the state of the network relayed to the controller via BGP-TE, thus closing the control feedback loop.

7.1.6 Research Contribution

- The results demonstrate that the lifespan of existing network equipment, specifically IP/MPLS network equipment, can be extended through the introduction of key SDN features into the network.
    - The protocols PCEP and PCC were introduced by the Cisco vendor to their existing network equipment, via an OS update. These two features are key to enabling the programmability of the network, and as such, is the minimum capability required of all other network equipment.
    - Two network topologies were used to demonstrate the features of the HSDN network namely full mesh and partial mesh topologies.
- The research further demonstrates that network automation is possible through the implementation of a control loop application, Figure 4.8.
    - The programmability that is introduced as a result of the SDN features, makes it possible for the same infrastructure to be flexible and autonomous in how it distributes demand across the network
- That various optimisations functions can be implemented depending on the traffic engineering objective.
    - In this work two objective functions were implemented to illustrate this capability, namely minimising the maximum link utilisation and minimum cost routing.
    - These objective functions are well known which simplified the interpretation of the results, but the PCE can be programmed to implement any other objective function if the needed information can be retrieved from the network state.

7.1.7 Generalisability of Results

- Emphasis was placed on using a near real-world network as opposed to using simulation. As a result, a virtual network topology was used, through the deployment of virtual routers configured into a network topology.
    - With this approach, the application provides a more realistic representation of how a real-world network would respond when upgraded with SDN features in terms of network programmability and automation features.
    - The control loop application demonstrated how these features enabled network automation, by adjusting the LSP bandwidth dynamically in response to demand.
    - Despite the use of a virtual network topology, the control loop application is easily transferable to a real-world network.

- The virtual routers have full routing capability but with a data plane that was reduced by the vendor. This means that the actual traffic that the routers can permit is greatly reduced.
    - Because of the reduced data plane, the configured LSPs could not carry any network traffic.
    - Thus, while the reservation of resources was optimised, the results lack consideration for the impact of path changes on traffic such as video and audio.
    - To this end, future work must consider the impact of LSP changes on traffic.
- Furthermore, the time window concept used to illustrate changes in demand over time presented these demand changes as abrupt instantaneous changes at certain time intervals and assumed the traffic was predictable and linear over periods.
    - While this representation of demand is not very realistic the abrupt demand changes at specific time intervals helped to make the impact of the optimisation process more distinguishable to the observer when considering the results discussed in chapter 6.

7.1.8 Future Research

Several researchable aspects have been identified that can be pursued in future research. The results showed that SDN enhancement can greatly benefit legacy network equipment, but some aspects of the research warrant further investigation.

- The reconfiguration capability of the automatic bandwidth adjustment control loop needs to be refined in future work to consider the impact of mass LSP reconfiguration on a capacitated network. This impact was not addressed in this study. For this control loop measure to be adopted by internet service providers, there is a need for clear guidelines as to which extend LSP reconfiguration can be tolerated especially the impact of reconfiguring a high number of LSP on a capacitated network.
- The centralised ODL controller models the network resources as objects in the MD-SAL. These models are updated via traditional protocols such as OSPF-TE and BGP-TE. The PCE knows the available resources via the TED and active LSPs via the LSP-DB. Currently, only a single autonomous system is considered. The research should be extended to multi autonomous systems exploring the federation of such systems. Global optimisation of multi autonomous systems with automatic bandwidth adjust should extend this work.
- The four-node network model takes advantage of the SDN controller's total network view, which is heavily reliant on the interior gateway protocols, in this case, OSPF-TE, to communicate the network state via the BGP speaker. Failures

are a common occurrence in IP networks due to a variety of reasons. Links can also exhibit intermittent failures called flapping. These failures, coupled with a larger topology, can impact network convergence, and hence result in a controller topology database that is out of sync with the actual network. Future research should consider mitigating techniques against such failures to make the hybrid SDN control loop more robust against such events.

- Finally, traffic modelling, which in this research has been modelled using well behaved traffic with demand changing at discrete time intervals, should be extended to either a measurement-driven model of traffic demand or to explore other forms of traffic matrix estimation. Traffic demand estimation is a research field on its own, but it can contribute greatly to how accurately network resources can be reserved for current demand.

8 Reference

[1] Cisco, “Cisco Visual Networking Index: Forecast and Methodology, 2016-2021,” *White Pap. Cisco public*, pp. 1–17, Jun. 2017.

[2] A. N and J. Jayageetha, “A Survey On Multi-Protocol Label Switching,” *Int. J. Technol. Enhanc. Emerg. Eng. Researh*, vol. 3, no. 02, pp. 25–28, 2015.

[3] E. Rosen, A. Viswanathan, and R. Callon, “Multiprotocol Label Switching Architecture,” *Internet Req. Comment*, vol. RFC 3031, pp. 1–62, Jan. 2001.

[4] D. Awduche, J. Malcolm, J. Agogbua, M. O’Dell, and J. McManus, “Requirements for Traffic Engineering Over MPLS,” *Internet Req. Comment*, vol. RFC 2702, pp. 1–29, 1999.

[5] Z. (Frank) Xu, *Designing and Implementing IP/MPLS-Based Ethernet Layer 2 VPN Services An Advanced Guide for VPLS and VLL*. John Wiley and Sons Ltd, 2009.

[6] E. Rosen and Y. Rekhter, “BGP/MPLS IP Virtual Private Networks (VPNs),” *Internet Req. Comment*, vol. RFC 4363, pp. 1–47, Feb. 2006.

[7] D. Awduche, J. Malcolm, J. Agogbua, J. Mcmanus, and M. O’Dell, “Requirements for Traffic Engineering Over MPLS,” *Req. Comments*, vol. RFC 2702, pp. 1–29, Sep. 1999.

[8] T. D. Nadeau, *MPLS Network Management*. Elsevier, 2003.

[9] A. Mendiola, J. Astorga, and E. Jacob, “A survey on the contributions of Software-Defined Networking to Traffic Engineering,” *Surv. Tutorials*, vol. 19, no. 2, pp. 918–953, 2017.

[10] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, “A roadmap for traffic engineering in software defined networks,” *Comput. Networks*, vol. 71, pp. 1–30, 2014.

[11] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, “RSVP-TE: Extensions to RSVP for LSP Tunnels,” *Internet Req. Comment*, vol. RFC 3209, pp. 1–61, Dec. 2001.

[12] N. Wang, K. H. Ho, G. Pavlou, and M. Howarth, “An overview of routing optimization for internet traffic engineering,” *IEEE Commun. Surv. Tutorials*, vol. 10, no. 1, pp. 36–56, 2008.

[13] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao, “Overview and Principles of Internet Traffic Engineering,” *Internet Req. Comment*, vol. RFC 3272, pp. 1–71, May 2002.

[14] J. de Oliveira, J. Vasseur, L. Chen, and C. Scoglio, “Label Switched Path (LSP) Preemption Policies for MPLS Traffic Engineering,” *Internet Req. Comment*, vol. RFC 4829, pp. 1–19, Apr. 2007.

[15] H. Farhady, H. Lee, and A. Nakao, “Software-Defined Networking: A survey,” *Comput. Networks*, vol. 81, pp. 79–95, 2015.

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

[17] S. Ortiz, “Software-defined networking: On the verge of a breakthrough?,” *Computer (Long. Beach. Calif).*, vol. 46, no. 7, pp. 10–12, Jul. 2013.

[18] S. Islam, N. Muslim, and J. W. Atwood, “A Survey on Multicasting in Software-Defined Networking,” *IEEE Commun. Surv. Tutorials*, no. c, pp. 1–33, 2017.

[19] Open Networking Foundation, “SDN Architecture Overview,” *Onf*, no. 1, pp. 1–5, 2013.

[20] J. H. Cox *et al.*, “Advancing Software-Defined Networks: A Survey,” *IEEE Access*, vol. 5, pp. 25487–25526, 2017.

[21] W. Wang, W. He, and J. Su, “Boosting the Benefits of Hybrid SDN,” *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 2165–2170, 2017.

[22] W. Wang, W. He, and J. Su, “Enhancing the effectiveness of traffic engineering in hybrid SDN,” *IEEE Int. Conf. Commun.*, 2017.

[23] A. N. Katov, A. Mihovska, and N. R. Prasad, “Hybrid SDN architecture for resource consolidation in MPLS networks,” *Wirel. Telecommun. Symp.*, vol. 2015-Janua, 2015.

[24] F. Paolucci, F. Cugini, A. Giorgetti, N. Sambo, and P. Castoldi, “A survey on the path computation element (pce) architecture,” *IEEE Commun. Surv. Tutorials*, vol. 15, no. 4, pp. 1819–1841, 2013.

[25] D. Kreutz *et al.*, “Software-Defined Networking: A Comprehensive Survey,” *Proc. IEEE*, vol. 103, no. 1, pp. 1–61, 2015.

[26] G. Rzym, K. Wajda, and K. Rzym, “Analysis of PCE-based path optimization in multi-domain SDN/MPLS/BGP-LS network,” *Int. Conf. Transparent Opt. Networks*, vol. 2016-Augus, pp. 12–16, 2016.

[27] J. Ash and J. L. Le Roux, “Path Computation Element (PCE) Communication Protocol generic requirements,” *Internet Req. Comment*, vol. RFC 4657, Sep. 2006.

[28] A. Farrel, J. P. Vasseur, and J. Ash, “A Path Computation Element (PCE)-Based Architecture,” *Internet Req. Comment*, vol. RFC 4655, pp. 1–41, Aug. 2006.

[29] D. A. Schupke, S. Member, and F. Rambach, “Path Computation Element – An Enabler for Automated Network Operation,” pp. 12–15, 2013.

[30] J. Vasseur and J. Le Roux, “Path Computation Element (PCE) Communication Protocol (PCEP),” *Internet Req. Comment*, vol. RFC 5440, pp. 1–87, Mar. 2009.

[31] J. Medved, S. Previdi, A. Farrel, and S. Ray, “North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP,” *Internet Req. Comment*, vol. RFC 7752, pp. 1–48, Mar. 2016.

[32] J. Medved, R. Varga, A. Tkacik, and K. Gray, “OpenDaylight: Towards a model-

driven SDN controller architecture," *Proceeding IEEE Int. Symp. a World Wireless, Mob. Multimed. Networks 2014, WoWMoM 2014*, 2014.

[33] Y. Jarraya *et al.*, "A Survey and a Layered Taxonomy of Software Defined Networking," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 4, pp. 1955–1980, 2014.

[34] B. A. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past , Present , and Future of Programmable Networks," *Ieee Commun. Surv. Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[35] P. Fonseca and E. Mota, "A Survey on Fault Management in Software-Defined Networks," *IEEE Commun. Surv. Tutorials*, no. c, pp. 1–39, 2017.

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

[37] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, and T. Gayraud, "Software-defined networking: Challenges and research opportunities for future internet," *Comput. Networks*, vol. 75, no. PartA, pp. 453–471, 2014.

[38] R. Enns, A. Bierman, M. Bjorklund, and J. Schoenwaelder, "Network Configuration Protocol (NETCONF)," *Internet Req. Comment*, vol. RFC 6241, pp. 1–113, Jun. 2011.

[39] R. G. Gallager, "A Minimum Delay Routing Algorithm Using Distributed Computation," *IEEE Trans. Commun.*, vol. 25, no. 1, pp. 73–85, 1977.

[40] J. M. Jaffe and F. H. Moss, "A Responsive Distributed Routing Algorithm for Computer Networks," *IEEE Trans. Commun.*, vol. 30, no. 7, pp. 1758–1762, 1982.

[41] S. Cruz, S. M. Laboratories, and P. Alto, "Global Internet : Application and Technology A DISTRIBUTED ALGORITHM FOR MULTIPATH COMPUTATION," *Technology*, pp. 1689–1693, 1999.

[42] J. Göbel, A. E. Krzesinski, and D. Stapelberg, "A distributed scheme for responsive network engineering," *IEEE Int. Conf. Commun.*, pp. 2070–2075, 2007.

[43] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, "Unicast QoS Routing Algorithms for SDN: A Comprehensive Survey and Performance Evaluation," *IEEE Commun. Surv. Tutorials*, vol. X, no. c, pp. 1–29, 2017.

[44] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," *OSDI, Oct*, pp. 1--6, 2010.

[45] S. Jain *et al.*, "B4: experience with a globally-deployed software defined wan," *ACM SIGCOMM - Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.

[46] M. Jarschel, T. Zinner, T. Hoßfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, attributes, and use cases: A compass for SDN," *IEEE Commun. Mag.*, vol. 52, no. 6, pp. 210–217, 2014.

[47] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," *ACM Sigcomm Comput. Commun.*, vol. 44, no. 2, pp. 87–98, 2014.

[48] G. S. Poo and H. Wang, "Multi-path routing versus tree routing for VPN bandwidth provisioning in the hose model," *Comput. Networks*, vol. 51, no. 6, pp. 1725–1743, 2007.

[49] A. Kumar, R. Rastogi, A. Silberschatz, and B. Yener, "Algorithms for provisioning virtual private networks in the hose model," *IEEE/ACM Trans. Netw.*, vol. 10, no. 4, pp. 565–578, 2002.

[50] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive, "A flexible model for resource management in virtual private networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 95–108, Oct. 2004.

[51] L. Zhang, J. Muppala, and S. Chanson, "Provisioning virtual private networks in the hose model with delay requirements," *Proc. Int. Conf. Parallel Process.*, vol. 2005, no. 1, pp. 211–218, 2005.

[52] T. Erlebach and M. Rüegg, "Optimal bandwidth reservation in hose-model VPNs with multi-path routing," *Proc. - IEEE INFOCOM*, vol. 4, no. C, pp. 2275–2282, 2004.

[53] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," *Internet Req. Comment*, vol. RFC 4271, pp. 1–104, Jan. 2006.

[54] D. Joyal, P. Galecki, S. Giacalone, R. Coltun, and F. Baker, "OSPF Version 2 Management Information Base," *Internet Req. Comment*, pp. 1–121, Dec. 2006.

[55] J. Moy, "OSPF Version 2," *Internet Req. Comment*, vol. RFC 2328, pp. 1–244, Apr. 1998.

[56] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu, "Traffic engineering in SDN/OSPF hybrid network," *Proc. - Int. Conf. Netw. Protoc. ICNP*, pp. 563–568, 2014.

[57] J. Zhang, K. Xi, M. Luo, and H. J. Chao, "Load balancing for multiple traffic matrices using SDN hybrid routing," *2014 IEEE 15th Int. Conf. High Perform. Switch. Routing, HPSR 2014*, pp. 44–49, 2014.

[58] R. Yasunaga, Y. Nakayama, T. Mochida, Y. Kimura, T. Yoshida, and K. I. Suzuki, "Optimal load balancing method for symmetrically routed hybrid SDN networks," *2015 21st Asia-Pacific Conf. Commun. APCC 2015*, pp. 234–238, 2016.

[59] R. Casellas, R. Vilalta, R. Martínez, and R. Muñoz, "Highly Available SDN Control of Flexi-Grid Networks With Network Function Virtualization-Enabled Replication," *J. Opt. Commun. Netw.*, vol. 9, no. 2, p. A207, Feb. 2017.

[60] R. Casellas, R. Vilalta, R. Martínez, and R. Muñoz, "Active Stateful PCE high-availability for the control of Flexi- grid Networks with Network Function Virtualization enabled replication," *Opt. Fiber Commun. Conf. 2016*, pp. 7–9, 2016.

[61] O. González de Dios *et al.*, “Multipartner Demonstration of BGP-LS-Enabled Multidomain EON Control and Instantiation With H-PCE [Invited],” *J. Opt. Commun. Netw.*, vol. 7, no. 12, p. B153, 2015.

[62] N. Sengezer, B. Puype, E. Karasan, and M. Pickavet, “A comparative study of single-layer and multi-layer traffic engineering approaches on transparent optical networks,” in *Proceedings of 2007 9th International Conference on Transparent Optical Networks, ICTON 2007*, 2007, vol. 4, pp. 105–108.

[63] Q. Awais, M. H. Malik, S. Hussain, and H. V. Tuan, “Traffic engineering using multi-protocol label switching (MPLS) for delay sensitive traffic,” *Proc. - 2015 IEEE Int. Conf. Comput. Intell. Commun. Technol. CICT 2015*, pp. 465–470, 2015.

[64] S. H. Sridhar and J. Arunnehru, “Traffic Engineering: An Application of MPLS L3 VPN Technology,” *Proc. 2nd Int. Conf. Trends Electron. Informatics, ICOEI 2018*, no. Icoei, pp. 1147–1151, 2018.

[65] H. Agrawal, J. Andrew, and M. Gregory, “Robust traffic engineering,” in *2008 2nd International Symposium on Advanced Networks and Telecommunication Systems, ANTS 2008*, 2008.

[66] T. Otoshi *et al.*, “Framework for traffic engineering under uncertain traffic information,” in *2016 International Conference on Information and Communication Technology Convergence, ICTC 2016*, 2016, pp. 264–266.

[67] S. Bitzer, J. Bruineberg, and S. J. Kiebel, “A Bayesian Attractor Model for Perceptual Decision Making,” *PLoS Comput. Biol.*, vol. 11, no. 8, pp. 1–35, 2015.

[68] M. Taruk, E. Budiman, M. Wati, and Haviluddin, “OSPF Wireless Mesh with MPLS Traffic Engineering,” pp. 119–122, 2020.

[69] F. Li and J. Chen, “MPLS traffic engineering load balance algorithm using deviation path,” *Proc. - 2012 Int. Conf. Comput. Sci. Serv. Syst. CSSS 2012*, pp. 601–604, 2012.

[70] Y. Q. Fang and L. Z. Wang, “An algorithm of static load balance based on topology for MPLS traffic engineering,” *2009 WASE Int. Conf. Inf. Eng. ICIE 2009*, vol. 2, pp. 26–28, 2009.

[71] Z. Shu *et al.*, “Traffic engineering in software-defined networking: Measurement and management,” *IEEE Access*, vol. 4, pp. 3246–3256, 2016.

[72] M. R. Abbasi, A. Guleria, and M. S. Devi, “Traffic engineering in software defined networks: A Survey,” *J. Telecommun. Inf. Technol.*, 2016.

[73] G. Rzym, K. Wajda, and K. Rzym, “Analysis of PCE-based path optimization in multi-domain SDN/MPLS/BGP-LS network,” in *International Conference on Transparent Optical Networks*, 2016.

[74] R. Martinez *et al.*, “Experimental Validation of Transport SDN Restoration of Signal-Degraded Connections in Flexi-Grid Networks - IEEE Conference Publication,” *Optical Fiber Communication Conference and Exposition*, 2018. [Online]. Available: https://ieeexplore-ieee-org.libproxy.cput.ac.za/document/8385775. [Accessed: 09-Apr-2020].

[75] C. Morin, G. Texier, and C. T. Phan, “On demand QoS with a SDN traffic engineering management (STEM) module,” in *2017 13th International Conference on Network and Service Management, CNSM 2017*, 2017, vol. 2018-January, pp. 1–6.

[76] K. P. Kadiyala and J. A. Cobb, “Inter-as traffic engineering with SDN,” in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2017*, 2017, vol. 2017-January, pp. 1–7.

[77] M. H. Chen, W. M. Wang, I. H. Chung, and C. F. Chou, “Incremental hybrid SDN deployment for enterprise networks,” *Proc. - 2017 IEEE 15th Int. Conf. Dependable, Auton. Secur. Comput. 2017 IEEE 15th Int. Conf. Pervasive Intell. Comput. 2017 IEEE 3rd Int. Conf. Big Data Intell. Compu*, vol. 2018-Janua, pp. 1143–1149, 2018.

[78] E. Husni and A. Bramantyo, “Design and Implementation of MPLS SDN Controller Application based on OpenDaylight,” in *2018 International Symposium on Networks, Computers and Communications, ISNCC 2018*, 2018.

[79] C. Morin, G. Texier, and C. T. Phan, “On demand QoS with a SDN traffic engineering management (STEM) module,” *2017 13th Int. Conf. Netw. Serv. Manag. CNSM 2017*, vol. 2018-Janua, pp. 1–6, 2018.

[80] Y. Sinha, S. Bhatia, V. S. Shekhawat, and G. S. S. Chalapathi, “MPLS based hybridization in SDN,” in *2017 4th International Conference on Software Defined Systems, SDS 2017*, 2017, pp. 156–161.

[81] M. Caria, T. Das, A. Jukan, and M. Hoffmann, “Divide and conquer: Partitioning OSPF networks with SDN,” in *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015*, 2015, pp. 467–474.

[82] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu, “Optimize routing in hybrid SDN network with changing traffic,” *2017 26th Int. Conf. Comput. Commun. Networks, ICCCN 2017*, 2017.

[83] I. Šeremet and S. Cauševic, “An analysis of reconvergence delay when using BGP-LS/PCEP as southbound protocols,” in *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2019 - Proceedings*, 2019, pp. 415–420.

[84] M. K. Porwal, A. Yadav, and S. V. Charhate, “Traffic analysis of MPLS and non MPLS network including MPLS signaling protocols and traffic distribution in OSPF and MPLS,” *Proc. - 1st Int. Conf. Emerg. Trends Eng. Technol. ICETET 2008*, pp. 187–192, 2008.

[85] Z. Aouini, A. Kortebi, and Y. Ghamri-Doudane, “Towards understanding residential internet traffic: From packets to services,” in *2016 7th International Conference on the Network of the Future, NOF 2016*, 2017.

[86] Q. Grandemange, O. Ferveur, M. Gilson, and E. Gnaedinger, “A live network AS-level traffic characterization,” in *2017 International Conference on Computing, Networking and Communications, ICNC 2017*, 2017, pp. 901–905.

[87] Q. Grandemange, Y. Bhujwalla, M. Gilson, O. Ferveur, and E. Gnaedinger, “An as-level approach to network traffic analysis and modelling,” in *IEEE*

International Conference on Communications, 2017.

[88] D. Yang and Z. Rong, "Evolution of the Internet at the autonomous system level," in *Chinese Control Conference, CCC*, 2015, vol. 2015-September, pp. 1313–1317.

[89] B. Yang, Y. Lu, K. Zhu, Y. Zhang, and J. Liu, "Evolution of the Internet and its measures," in *1st International Conference on Electronics Instrumentation and Information Systems, EIIS 2017*, 2018, vol. 2018-January, pp. 1–4.

[90] X. Huang, S. Cheng, K. Cao, P. Cong, T. Wei, and S. Hu, "A Survey of Deployment Solutions and Optimization Strategies for Hybrid SDN Networks," *IEEE Commun. Surv. Tutorials*, vol. 21, no. 2, pp. 1483–1507, Apr. 2019.

[91] L. Zheng, C. Joe-Wong, J. Chen, C. G. Brinton, C. W. Tan, and M. Chiang, "Economic viability of a virtual ISP," in *Proceedings - IEEE INFOCOM*, 2017.

[92] P. Bangera, S. Hasan, and S. Gorinsky, "An advertising revenue model for access ISPs," *Proc. - IEEE Symp. Comput. Commun.*, pp. 582–589, 2017.

[93] X. Sun, "Research on QoS of next generation network based on MPLS," *Proc. 2012 IEEE Int. Conf. Inf. Sci. Technol. ICIST 2012*, pp. 294–296, 2012.

[94] D. Zhang and D. Ionescu, "QoS Performance Analysis in Deployment of DiffServ-aware MPLS Traffic Engineering," in *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*, 2007, vol. 2, pp. 963–967.

[95] T. Takeda, E. Oki, I. Inoue, K. Shiomoto, K. Fujihara, and S. I. Kato, "Implementation and experiments of Path Computation Element based backbone network architecture," *IEICE Trans. Commun.*, vol. E91-B, no. 8, pp. 2704–2706, 2008.

[96] R. Amin, M. Reisslein, and N. Shah, "Hybrid SDN Networks: A Survey of Existing Approaches," *IEEE Commun. Surv. Tutorials*, no. c, p. 1, 2018.

[97] K. H. Rosen, *Discrete Mathematics and Its Applications: Handbook of Graph Theory*, 2nd ed. Chapman and Hall, 2003.

[98] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[99] A. M. C. Z. Koster and X. Muñoz, *Graphs and Algorithms in Communication Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[100] P. Michal and D. Medhi, *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Elsevier.

[101] D. Medhi and K. Ramasamy, *Network Routing: Algorithms Protocols and Architectures The Morgan Kaufmann Series in Networking*. Elsevier, 2007.

[102] IBM, *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual*, 12.6. 2015.

[103] W. Ben-Ameur and H. Kerivin, "Routing of Uncertain Traffic Demands," *Optim. Eng.*, vol. 6, no. 3, pp. 283–313, Sep. 2005.

[104] G. de Veciana, S. Park, A. Sang, and S. Weber, “Routing and Provisioning VPNs based on Hose traffic models and/or constraints,” *Proc. 40th Annu. Allert. Conf. Commun. Control Comput.*, pp. 77–86, 2002.

[105] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, and B. Yener, “Provisioning a virtual private network: a network design problem for multicommodity flow.,” *Proc. thirty-third Annu. ACM Symp. theory Comput. STOC 2001. Hersonissos, Crete, Greece, July 6--8, 2001.*, pp. 389–398, 2001.

[106] J. G. M. Z. Ben, H. Olivier, and O. B. Jean-Marie Garcia, Mohamed Zied Ben Hamouda, “Network Planning : Traffic Matrices Estimation and Demand Uncertainty,” *Stud. Inform. Universalis*, vol. 8, no. 2, pp. 236–262, 2010.

[107] M. H. Raza, S. C. Sivakumar, A. Nafarieh, and B. Robertson, “A comparison of software defined network (SDN) implementation strategies,” *Procedia Comput. Sci.*, vol. 32, pp. 1050–1055, 2014.

[108] “Networking Software (IOS & NX-OS) - Products IOS Cisco IOS Software Releases - Cisco.” [Online]. Available: https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-software-releases-listing.html. [Accessed: 25-Apr-2019].

[109] “Cisco IOS XR Software Release 6.0 Operational Enhancements Data Sheet - Cisco.” [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-xr-software/datasheet-c78-736154.html. [Accessed: 25-Apr-2019].

[110] “Cisco IOS XRv Router Installation and Configuration Guide - Cisco IOS XRv Router Overview [Cisco IOS XRv Software] - Cisco Systems.” [Online]. Available: https://www.cisco.com/en/US/docs/ios_xr_sw/ios_xrv/install_config/b_xrvr_432_chapter_01.html. [Accessed: 25-Apr-2019].

[111] “Cisco NX-OS - Cisco.” [Online]. Available: https://www.cisco.com/c/en/us/products/ios-nx-os-software/nx-os/index.html. [Accessed: 25-Apr-2019].

[112] “HPE ProLiant ML150 Gen9 Server - Overview.” [Online]. Available: https://support.hpe.com/hpsc/doc/public/display?docLocale=null&docId=emr_na-c04518697. [Accessed: 25-Apr-2019].

[113] OpenDaylight Project, “OpenDaylight Project,” *OpenDaylight Project*, 2013. [Online]. Available: https://wiki.opendaylight.org/view/Main_Page. [Accessed: 28-Apr-2019].

[114] C. Hill and E. Volt, “Embracing SDN in Next Generation Networks.” [Online]. Available: https://www.slideshare.net/CiscoPublicSector/embracing-sdn-in-next-generation-networks?from_action=save. [Accessed: 19-Feb-2019].

[115] C. Trois, M. D. Del Fabro, L. C. E. De Bona, and M. Martinello, “A Survey on SDN Programming Languages: Toward a Taxonomy,” *IEEE Commun. Surv. Tutorials*, vol. 18, no. 4, pp. 2687–2712, 2016.

[116] A. Bierman, M. Bjorklund, and K. Watsen, “RESTCONF Protocol,” *Internet Req. Comment*, vol. RFC 8040, pp. 1–137, 2017.

[117] “OpenDaylight SDN Controller Platform (OSCP):Rest Reference - OpenDaylight Project.” [Online]. Available: https://wiki.opendaylight.org/view/OpenDaylight_SDN_Controller_Platform_(OSCP):Rest_Reference. [Accessed: 19-Feb-2019].

[118] “OpenDaylight dlux:Getting started - OpenDaylight Project.” [Online]. Available: https://wiki.opendaylight.org/view/OpenDaylight_dlux:Getting_started. [Accessed: 20-Feb-2019].

9 Appendixes

9.1 MPLS TE Configuration

```
1   mpls traffic-eng
2    interface GigabitEthernet0/0/0/0
3    interface GigabitEthernet0/0/0/1
4    interface GigabitEthernet0/0/0/2
5    pce
6     peer source ipv4 192.168.146.2
7     peer ipv4 192.168.146.1
8     stateful-client
9      instantiation
10     timers redelegation-timeout 0
11     timers state-timeout 0
12     delegation
13   auto-tunnel pcc
14    tunnel-id min 1000 max 5000
     reoptimize timers delay installation 0
```

Figure 9.1: MPLS TE configuration.

Figure 9.1 shows the configuration process for MPLS-TE (lines 1-4), PCE configuration (line 5-13) and PCC configuration (line 13). PCE location, as well as the MPLS-TE interfaces, are specified.

```
xterm
RP/0/0/CPU0:xrv1-1#sh mpls interfaces
Sat May  4 17:37:11.422 UTC
Interface                  LDP      Tunnel   Static   Enabled
-------------------------- -------- -------- -------- --------
GigabitEthernet0/0/0/0     Yes      Yes      No       Yes
GigabitEthernet0/0/0/1     Yes      Yes      No       Yes
GigabitEthernet0/0/0/3     Yes      Yes      No       Yes
RP/0/0/CPU0:xrv1-1#
```

Figure 9.2: Confirming MPLS TE configuration on the router.

For router 1, Figure 9.2 shows the status of the interfaces for which MPLS-TE label distribution protocol and support for tunnelling has been configured.

```
xterm
RP/0/0/CPU0:xrv1-1#sh mpls traffic-eng pce peer
Sun May  5 20:00:38.193 UTC
        Address   Precedence        State         Learned From
--------------- ------------ ------------ --------------------
  192.168.146.1          255           Up        Static config
RP/0/0/CPU0:xrv1-1#
```

Figure 9.3: Confirming PCE peer status.

Once the PCE is configured on the router, the router PCE process and controller PCE process establish a client-server relationship, where the router process becomes the client and the controller PCE process the server. Figure 9.3 shows the establishment of a peer relationship between the router and the controller. Communications between the client-server pair are over the PCEP.

9.2 BGP Configuration

With BGP-LS, a router maintains one or more databases storing link-state information from a given area.

```
1  router bgp 100
2   bgp router-id 172.16.0.1
3   address-family link-state link-state
4   neighbor 19.168.146.1
5    remote-as 100
6    session-open-mode passive-only
7    address-family link-state link-state
```

Figure 9.4: BGP configuration.

In Figure 9.4, a BGP-LS session is established with the northbound controller with OSPF as the IGP running in the network with the TE extensions configured. The TE router ID (line 2) needs to be a stable IP address of the advertising router. It is used to uniquely identify the router in the TED. It is advisable that the same router ID is used irrespective of the routing protocol. RFC 3630 refers to the router ID as the Router Address.

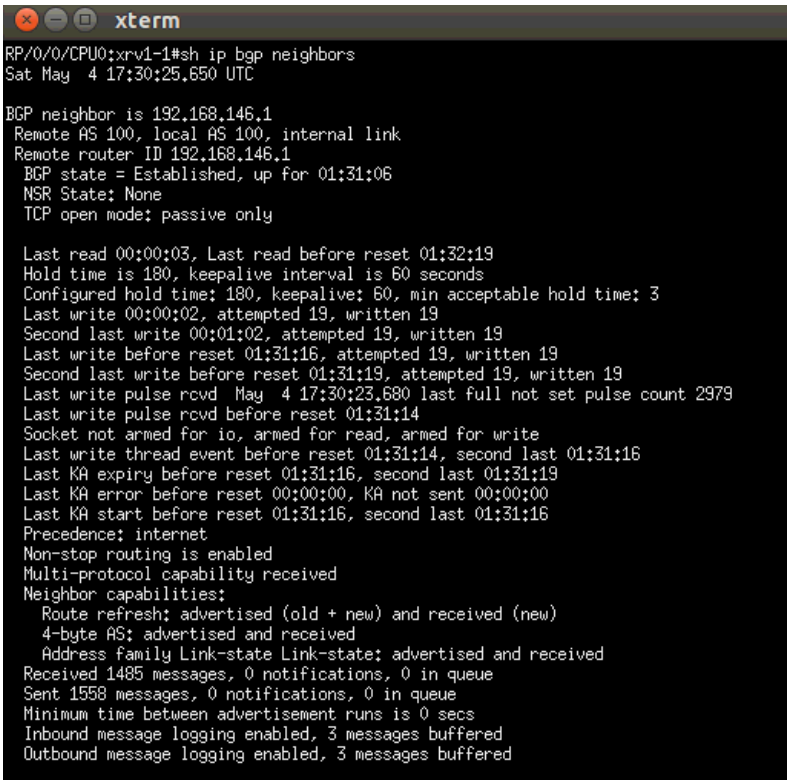

Figure 9.5: Confirming BGP configuration on the router.

In Figure 9.5, the router shows the BGP neighbour IP-Address as 192.168.146.1, which is the ODL controllers IP address. The BGP status shows that the session is established and active.

9.3 OSPF-TE Configuration

The IOS XRv router requires explicit configuration in order to enable the TE extensions for OSPF. In Figure 9.6, an OSPF process is created. For this implementation, only a single area was configured.

```
1  router ospf 100
2   distribute bgp-ls
3   address-family ipv4 unicast
4   area 0
5    mpls ldp auto-config
6    mpls traffic-eng
7    interface Loopback0
8     cost 1
9     passive enable
10   interface GigabitEthernet0/0/0/0
11    cost 10
12    network point-to-point
13   interface GigabitEthernet0/0/0/1
14    cost 10
15    network point-to-point
16   interface GigabitEthernet0/0/0/4
17    cost 10
18    network point-to-point
19  mpls traffic-eng router-id Loopback0!
```

Figure 9.6: OSPF configuration.

Apart from enabling the TE extensions in OSPF (line 6), there needs to be an explicit listing of the interfaces (lines 10-17) for which the TE information should be flooded.

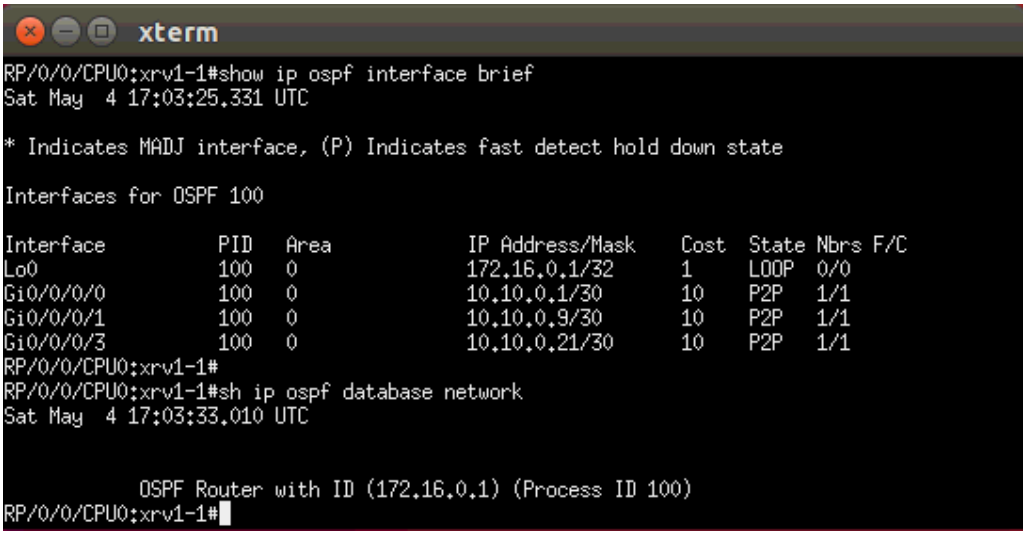

Figure 9.7: Confirming OSPF-TE router configuration.

Figure 9.7 shows the interfaces listed under OSPF area 0 for router 1 along with the link cost and type of link.

9.4 Controller Configuration

The ODL Boron SR4 controller with distribution Karaf 0.5.4 build was used as the target environment. The ODL Karaf container and Java class files are portable and run on any Java 7- or Java 8 compliant JVM. The Boron controller was used because of the ease with which the BGP PCEP features could be installed and configured. In ODL, the underlying network devices and applications are represented as objects or models, whose interactions are processed within the SAL.

For the network devices to be represented in the SAL, a minimal feature module installation, shown in Figure 9.8, is performed on the ODL controller.

```
1  feature install: odl-restconf
2  feature install: odl-bgpls-bgpcep-all
3  feature install: odl-pcep-bgpcep-all
4  feature install: odl-dlux-all
```

Figure 9.8: ODL configuration.

The explanations are as follows:

Line 1: The REST interface is installed, enabling REST API access to the MD-SAL. RESTconf is a standardised mechanism [116] to allow web applications to access configuration data, state data, remote procedure call operations and event notifications within a network device. RESTconf allows access to data stores located in the controller [117].

Line 2: The BGP link-state plugin is installed; this provides an implementation of BGP (RFC4271) as a southbound protocol and renders all basic BGP speaker capabilities such as inter or intra AS peering and routes advertising. Northbound the plugins API provides read-only access to all RIBs, link-state topology view and read-write programmable RIBs.

Line 3: PCEP plugin is installed; this extends the capabilities of the controller to include PCE and PCEP (RFC5440) allowing the LSP management functionality; the plugin consists of the protocol library, PCEP session handling, stateful PCE LSP-DB and active stateful PCE LSP operations.

Line 4: Graphical user interface for ODL is stalled. ODL DLUX is a Javascript-based stateless user interface. It communicates with the service backend to provide a consistent interface [118].

The ODL Karaf 0.5.4 edition comes pre-configured with a baseline BGP configuration, which can be edited as follows:

- The configuration file 31-bgp.xml, which defines the basic parser and RIB.
- For the test network, the important BGP peer and BGP RIB configurations are customised in file 41-bgp-example.xml.

Both files are in the path (etc/opendaylight/karaf) of the ODL deployment.

Figure 9.9 shows a listing of all the BGP and PCEP features that were installed on the controller.

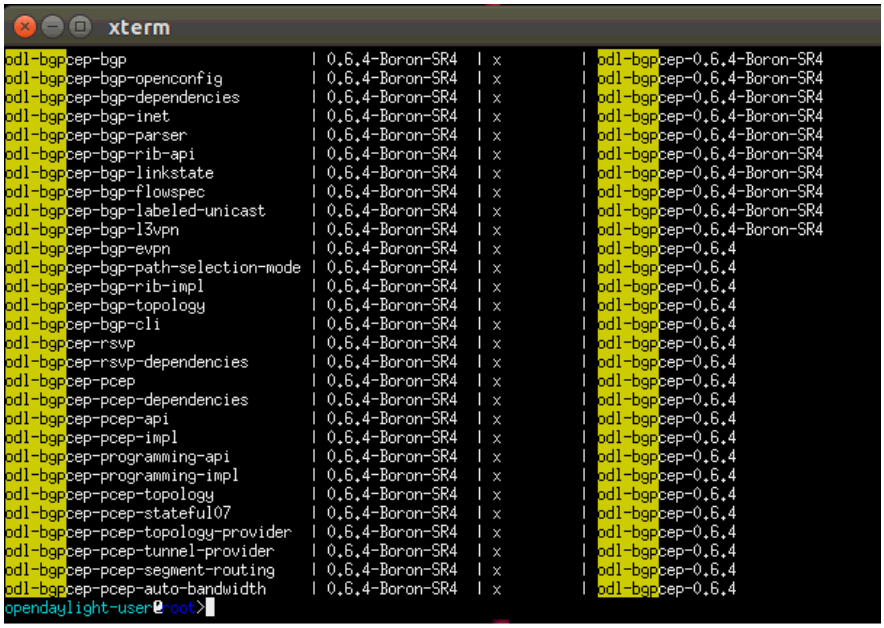

Figure 9.9: OpenDaylight SDN controller features.

9.5 MAIN12_MultiTime_Hose.java

Main program that drives all the processes.

```
1.  package sdnGraph;
2.  import java.io.BufferedWriter;
3.  import java.io.File;
4.  import java.io.FileWriter;
5.  import java.util.ArrayList;
6.  import java.util.List;
7.  import org.jgrapht.DirectedGraph;
8.  import org.jgrapht.GraphPath;
9.  import org.jgrapht.alg.shortestpath.KShortestPaths;
10. import org.jgrapht.graph.DefaultWeightedEdge;
11. import org.jgrapht.graph.SimpleDirectedGraph;
12. import org.json.simple.JSONArray;
13. import post.LSP;
14.
15. public class MAIN12_MultiTime_Hose {
16. /* Origin destination demand pairs
17.      * [0] = "2886729732";    [32] ----------[29]   [0] ----------[1]
18.      * [1] = "2886729729";    |\_________ |    |\_________   |
19.      * [2] = "2886729730";    |           \|    |            \|
20.      * [3] = "2886729731";    [31]----------[30]   [3]----------[2] */
21.
22.     /**   Setting source(s) destination (t) pairs   */
23.     static int[] source      = {1,1,1,  2,2, 3, 0,0};    // 12 source nodes
24.     static int[] destination = {0,2,3,  1,0, 1, 1,2};   // destination nodes
```

```
25.
26.     /** Scaling factor */
27.     static double d_SF  = 1 / 1e7;      // Demand Scaling factor
28.     static double ce_SF = 1 / 1e7;      // Scaling factor
29.     static double M     = Math.pow(10, 6);
30.
31.     /** Online or Offline */
32.     static int REST     = 1;            // 1-online 0-Offline
33.
34.     /** Setting demand volume between source and destination pairs for all demand pairs.  */
35.     static double H[][] = new double[][] {
36. /*1-0*/     /*1-2*/     /*1-3*/     /*2-1*/     /*2-0*/     /*3-1*/     /*0-1*/     /*0-2*/
37. {50*M*d_SF, 50*M*d_SF, 50*M*d_SF, 50*M*d_SF, 50*M*d_SF, 50*M*d_SF, 50*M*d_SF, 50*M*d_SF },  /*TW0*/
38. {100*M*d_SF, 100*M*d_SF, 100*M*d_SF, 50*M*d_SF, 50*M*d_SF, 50*M*d_SF, 100*M*d_SF, 150*M*d_SF },/*TW1*/
39. {50*M*d_SF, 50*M*d_SF, 50*M*d_SF, 100*M*d_SF, 100*M*d_SF, 150*M*d_SF, 100*M*d_SF, 100*M*d_SF },/*TW2*/
40. {50*M*d_SF, 50*M*d_SF, 50*M*d_SF, 100*M*d_SF, 150*M*d_SF, 50*M*d_SF, 150*M*d_SF, 100*M*d_SF } /*TW3*/};
41.
42. /**  Setting the bandwidth out  */
43. static double BW_0_o[] = new double[] { 100 * M * d_SF,250 * M * d_SF,  200 * M * d_SF,     250 * M * d_SF };
44. static double BW_0_i = 0 * M * d_SF;
45. static double BW_1_o[] = new double[] { 150 * M * d_SF,300 * M * d_SF,  150 * M * d_SF,     150 * M * d_SF };
46. static double BW_1_i = 0 * M * d_SF;
47. static double BW_2_o[] = new double[] { 100 * M * d_SF,100 * M * d_SF,  200 * M * d_SF, 250 * M * d_SF };
48. static double BW_2_i = 0 * M * d_SF;
49. static double BW_3_o[] = new double[] { 50 * M * d_SF,  50 * M * d_SF,  150 * M * d_SF,     50 * M * d_SF };
50. static double BW_3_i = 0 * M * d_SF;
51. public static void main(String[] args) {
52. System.out.println("Setting Variables  ..........................................................");
53. DiGraph diGraph = new DiGraph();
54.
55. /** Online **/
56. if (REST == 1) { diGraph = online(diGraph); }
57.
58. /** Offline **/
59. if (REST == 0) {diGraph = offline(diGraph); }
60.
61. double[] ce = new double[diGraph.getGraphLength()];
62. /** Set edge Capacities of all directional edges to 1Gbps **/
63. for (int n = 0; n < diGraph.getGraphLength(); n++) {
64.     ce[n] = Math.pow(10, 9) * ce_SF;
65. }
66. /** Use JGraph: Find k shortest paths to all nodes.  */
67. System.out.println("Finding k-shortest paths using JGraph ....................................");
68. int kMaxPaths = 5;          // max num of shortest paths that will be searched for using k-Shortest Paths
69. List<ArrayList<GraphPath<String,DefaultWeightedEdge>>>kShortestPathsToDemands=findShortestPathsUsingJGraph(di
    Graph, source, destination, kMaxPaths);    // Find k shortest paths from source - destination
70. System.out.println("\nGetting list of edges in diGraph ....................................");
71. ArrayList<String> listOfEdges = listOfEdgesInDigraph(diGraph);
72.
73. /**  Setting loop variables */
74. double edgeCapacityArrayMinMax[]    = { 0 };
75. double edgeCapacityArrayMinCost[]   = { 0 };
76. List<ArrayList<String>> edgeCapacityArrayListMinMax = new ArrayList<>();
77. List<ArrayList<String>> edgeCapacityArrayListMinCost = new ArrayList<>();
78. List<ArrayList<String>> demandVolume = new ArrayList<>();
79. StringpathCPLEXoutput1="C:\\Users\\brandta.ADS\\Dropbox\\Dev\\gitStuff\\2015\\REST12_MultiTime_Hose\\";
80. String filenameCPLEXoutput1 = "CPLEXoutEdgeCapacity_MinMax.txt"; // Work PC
```

```
81.  StringpathCPLEXoutput2="C:\\Users\\brandta.ADS\\Dropbox\\Dev\\gitStuff\\2015\\REST12_MultiTime_Hose\\";
82.  String filenameCPLEXoutput2 = "CPLEXoutEdgeCapacity_MinCost.txt"; // Work PC
83.
84.  for (int TW = 0; TW < H.length; TW++) {
85.     double node0_o = H[TW][6]+ H[TW][7];
86.     double node1_o = H[TW][0]+ H[TW][1]+ H[TW][2];
87.     double node2_o = H[TW][3]+ H[TW][4];
88.     double node3_o = H[TW][5];
89.     demandVolume.add(new ArrayList<String>());  // Add a new column for every loop
90.     edgeCapacityArrayListMinMax.add(new ArrayList<String>());   // Add a new column for every loop
91.     edgeCapacityArrayListMinCost.add(new ArrayList<String>());  // Add a new column for every loop
92.
93.     /**  Set demand for time window  */
94.     for (int l = 0; l < H[TW].length; l++) {
95.        demandVolume.get(TW).add(Double.toString(H[TW][l]));
96.     }
97.
98.     /** Reassign H[][] to h[] */
99.     double[] h = new double[] { H[TW][0], H[TW][1], H[TW][2], H[TW][3], H[TW][4], H[TW][5], H[TW][6], H[TW][7] };
100.
101.    /**  * Apply hose limitations  */
102.    if (node0_o > BW_0_o[TW]) {break;}
103.    if (node1_o > BW_1_o[TW]) {break;}
104.    if (node2_o > BW_2_o[TW]) {break;}
105.    if (node3_o > BW_3_o[TW]) {break;}
106.
107.    /** Optimise */
108.    System.out.println("Optimising Network using MinMax .....");
109. edgeCapacityArrayMinMax=optimiseNetworkUsingCPLEX_MinMax(ce_SF,listOfEdges,diGraph,kShortestPathsToDemands
     ,ce,h);   // Optimise network graph
110.
111.    System.out.println("Optimising Network using MinCost .....");
112. edgeCapacityArrayMinCost=optimiseNetworkUsingCPLEX_MinCost(ce_SF,listOfEdges,diGraph,kShortestPathsToDemand
     s,ce,h);    // Optimise network graph
113.
114.    // - Write edge capacity array to MinMax list
115.    for (int edge = 0; edge < edgeCapacityArrayMinMax.length; edge++) {
116.       edgeCapacityArrayListMinMax.get(TW).add(Double.toString(edgeCapacityArrayMinMax[edge]));
117.    }
118.    // - Write edge capacity array to MinCost list
119.    for (int edge = 0; edge < edgeCapacityArrayMinCost.length; edge++) {
120.       edgeCapacityArrayListMinCost.get(TW).add(Double.toString(edgeCapacityArrayMinCost[edge]));
121.    }
122.
123.    /** Write to file */
124.    if (TW == (H.length-1)) {
125.       System.out.println("\nTW "+(h.length-1) );
126.       /** SAVE -- CPLEX output file */
127.       System.out.println("\nSaving Min Max edgeCapacity list to file .....");
128.       saveFilev2(d_SF,ce_SF,pathCPLEXoutput1,filenameCPLEXoutput1,edgeCapacityArrayListMinMax,listOfEdges, h,
     TW, demandVolume);
129.       System.out.println("\nSaving Min Cost edgeCapacity list to file .....");
130.       saveFilev2(d_SF,ce_SF,pathCPLEXoutput2,filenameCPLEXoutput2,edgeCapacityArrayListMinCost,listOfEdges, h,
     TW, demandVolume);
131.    }
132. }
133. } // END OF MAIN12_MultiTime
```

```
134.
135. /**  Returns: biGraph biGraph is a directional graph.  */
136. public static DiGraph buildDiGraphFromRest(String topoFilenameSave, String baseURL) {
137.     String[] stringSplitSource  = new String[10];
138.     String[] stringSplitsTP     = new String[4];
139.     String[] stringSplitDest    = new String[10];
140.     String[] stringSplitdTP     = new String[4];
141.     GetJsonTopoArray getJsonTopoArray = new GetJsonTopoArray();
142.     DiGraph diGraph         = new DiGraph();
143.
144.     /**  Topology from URL: 0 : pcepTopology 1 : exampleLinkStateTopology baseURL  */
145.     int Topology = 1;
146.     URL url = new URL();
147.     JSONArray JSONTopoArray = getJsonTopoArray.fromREST(url.target(Topology, baseURL), topoFilenameSave);
148.
149.     /**  Get jsonLinkArray from the JSONTopoArray  */
150.     GetLinkArray getLinkArray = new GetLinkArray();
151.     JSONArray jsonLinkArray = getLinkArray.linkData(JSONTopoArray);
152.
153.     /**  Get jsonNodeArray from the JSONTopoArray  */
154.     GetNodeArray getNodeArray= new GetNodeArray();
155.     JSONArray jsonNodeArray = getNodeArray.nodeData(JSONTopoArray);
156.
157.     /**  The nodeArrayProcessing method populates the RouterArray. RouterArray is an array with the router
    names only. */
158. NodeArrayProcessing nodeArrayProcessing = new NodeArrayProcessing();
159. String[] routerArray = nodeArrayProcessing.getGraph(jsonNodeArray);
160.
161. /**  Removes delimiters from routerArray.  Then populates the nodes in diGraph with the delimited router
    information. */
162. String[] stringSplitNode = new String[4];
163. StringProcessing stringProcessing = new StringProcessing();
164. for (int k = 0; k < routerArray.length; k++) {
165.     stringSplitNode = stringProcessing.removeDelimterNode(routerArray[k], 3);   // destTP
166.     diGraph.setN(k, stringSplitNode[9]);    // This populates a nodes array in the diGraph class.
167. }
168.
169. /**  Print Nodes vector in digraph  */
170. System.out.println("\n" + "diGraph Nodes (V) = " + jsonNodeArray.size());
171. for (int k = 0; k < routerArray.length; k++) {
172.     System.out.println((k + 1) + " : " + diGraph.getN(k));
173. }
174.
175. /**  Link Array processing Construct digraph */
176. String Sas, Sdomain, Sarea, Srouter, Stp, Das, Ddomain, Darea, Drouter, Dtp;
177. LinkArrayProcessing linkArrayProcessing = new LinkArrayProcessing();
178. String[][] SrcDestData = linkArrayProcessing.getGraph(jsonLinkArray);
179. for (int k = 0; k < jsonLinkArray.size(); k++) {
180.     /** Split string at & =. */
181.     stringSplitSource = stringProcessing.removeDelimterSourceRouter(SrcDestData[k][0], k);// sourceRouter
182.     stringSplitsTP = stringProcessing.removeDelimterSourceTP(SrcDestData[k][1], k); // sourceTP
183.     stringSplitDest = stringProcessing.removeDelimterDestRouter(SrcDestData[k][2], k); // destRouter
184.     stringSplitdTP = stringProcessing.removeDelimterDestTP(SrcDestData[k][3], k); // destTP
185.     /** Building the graph */
186.     Sas = stringSplitSource[3];
187.     Sdomain = stringSplitSource[5];
188.     Sarea = stringSplitSource[7];
```

```
189.     Srouter = stringSplitSource[9];
190.     Stp = stringSplitsTP[3];
191.     Das = stringSplitDest[3];
192.     Ddomain = stringSplitDest[5];
193.     Darea = stringSplitDest[7];
194.     Drouter = stringSplitDest[9];
195.     Dtp = stringSplitdTP[3];
196.     diGraph.setGraph(k, Sas, Sdomain, Sarea, Srouter, Stp, Das, Ddomain, Darea, Drouter, Dtp);
197. }
198. /**  Printing the directional Graph in table form.  */
199. System.out.println("Didirectional Graph from JSON file");
200. System.out.println("Link\t" + "sAS[0]\t" + "sDomain[1]\t" + "sArea[2]  " + "sRouter[3]\t" + "sTP[4]\t\t"+"dAS[5]\t" + "dDo
     main[6]\t" + "dArea[7]  " + "dRouter[8]\t" + "dTP[9]");
201. for (int k = 0; k < jsonLinkArray.size(); k++) {
202. System.out.println(k+1+"\t"+diGraph.getGraph(0,k)+"\t"+diGraph.getGraph(1,k)+"\t"+diGraph.getGraph(2,k)+"\t"+diGra
     ph.getGraph(3,k)+"\t"+diGraph.getGraph(4,k)+"\t"+diGraph.getGraph(5,k)+"\t"+diGraph.getGraph(6,k)+"\t"+diGraph.get
     Graph(7,k)+"\t"+diGraph.getGraph(8,k)+"\t"+diGraph.getGraph(9,k));
203. }
204. return diGraph;
205. }
206. /** Build from file */
207. public static DiGraph buildDiGraphFromFile(String filePath, String topoFilenameRetrieve) {
208.     String[] stringSplitSource = new String[10];
209.     String[] stringSplitsTP = new String[4];
210.     String[] stringSplitDest = new String[10];
211.     String[] stringSplitdTP = new String[4];
212.     GetJsonTopoArray getJsonTopoArray = new GetJsonTopoArray();
213.     DiGraph diGraph = new DiGraph();
214.
215.     /**  The topology can be received from file: Use fromFile() Used for testing when network is not available.   */
216.     String jsonRawTopoFile = filePath + topoFilenameRetrieve;
217.     JSONArray JSONTopoArray = getJsonTopoArray.fromFile(jsonRawTopoFile);
218.
219.     /**  Get jsonLinkArray from the JSONTopoArray  */
220.     GetLinkArray getLinkArray = new GetLinkArray();
221.     JSONArray jsonLinkArray = getLinkArray.linkData(JSONTopoArray);
222.
223.     /**  Get jsonNodeArray from the JSONTopoArray  */
224.     GetNodeArray getNodeArray = new GetNodeArray();
225.     JSONArray jsonNodeArray = getNodeArray.nodeData(JSONTopoArray);
226.
227.     /**  The nodeArrayProcessing method populates the RouterArray. RouterArray is an array with the router
     names only. */
228.     NodeArrayProcessing nodeArrayProcessing = new NodeArrayProcessing();
229.     String[] routerArray = nodeArrayProcessing.getGraph(jsonNodeArray);
230.
231.     /**  Removes delimiters from routerArray.  Populates the nodes in diGraph with the delimited router
     information.  */
232.     String[] stringSplitNode = new String[4];
233.     StringProcessing stringProcessing = new StringProcessing();
234.     for (int k = 0; k < routerArray.length; k++) {
235.         stringSplitNode = stringProcessing.removeDelimterNode(routerArray[k], 3);   // destTP
236.         diGraph.setN(k, stringSplitNode[9]);    // This populates a nodes array in the diGraph class.
237.     }
238.
239.     /**  Print Nodes vector in digraph */
240.     System.out.println("\n" + "diGraph Nodes (V) = " + jsonNodeArray.size());
```

```
241.    for (int k = 0; k < routerArray.length; k++) {
242.       System.out.println((k + 1) + " : " + diGraph.getN(k));
243.    }
244.    System.out.println();
245.
246.    String Sas, Sdomain, Sarea, Srouter, Stp, Das, Ddomain, Darea, Drouter, Dtp;
247.    LinkArrayProcessing linkArrayProcessing = new LinkArrayProcessing();
248.    String[][] SrcDestData = linkArrayProcessing.getGraph(jsonLinkArray);
249.    for (int k = 0; k < jsonLinkArray.size(); k++) {
250.       /* Split string at & =. */
251.       stringSplitSource=stringProcessing.removeDelimterSourceRouter(SrcDestData[k][0],k);// sourceRouter
252.       stringSplitsTP = stringProcessing.removeDelimterSourceTP(SrcDestData[k][1], k); // sourceTP
253.       stringSplitDest = stringProcessing.removeDelimterDestRouter(SrcDestData[k][2], k); // destRouter
254.       stringSplitdTP = stringProcessing.removeDelimterDestTP(SrcDestData[k][3], k); // destTP
255.       /** Building the graph */
256.       Sas = stringSplitSource[3];
257.       Sdomain = stringSplitSource[5];
258.       Sarea = stringSplitSource[7];
259.       Srouter = stringSplitSource[9];
260.       Stp = stringSplitsTP[3];
261.       Das = stringSplitDest[3];
262.       Ddomain = stringSplitDest[5];
263.       Darea = stringSplitDest[7];
264.       Drouter = stringSplitDest[9];
265.       Dtp = stringSplitdTP[3];
266.       diGraph.setGraph(k, Sas, Sdomain, Sarea, Srouter, Stp, Das, Ddomain, Darea, Drouter, Dtp);
267.    }
268.
269.    /**  Printing the Didirectional Graph in table form.  */
270.    System.out.println("Didirectional Graph from JSON file");
271. System.out.println("Link\t"+"sAS[0]\t"+"sDomain[1]\t"+"sArea[2]"+"sRouter[3]\t"+"sTP[4]\t\t"
272. +"dAS[5]\t"+"dDomain[6]\t"+"dArea[7]"+"dRouter[8]\t"+"dTP[9]");
273. for (int k = 0; k < jsonLinkArray.size(); k++) {
274. System.out.println(k+1+"\t"+diGraph.getGraph(0,k)+"\t"+diGraph.getGraph(1,k)+"\t"+diGraph.getGraph(2,k)+"\t"+diGrap
     h.getGraph(3,k)+"\t"+diGraph.getGraph(4,k)+"\t"+diGraph.getGraph(5,k)+"\t"+diGraph.getGraph(6,k)+"\t"+diGraph.getGr
     aph(7,k)+"\t"+diGraph.getGraph(8,k)+"\t"+diGraph.getGraph(9,k));
275. }
276. return diGraph;
277. }
278. /**  Finds shortest path using the JGraph library. Returns: ShortestPathsToDemand */
279. public static List<ArrayList<GraphPath<String, DefaultWeightedEdge>>> findShortestPathsUsingJGraph(DiGraph
     diGraph,int[] source, int[] destination, int kMaxPaths) {
280.    /** JGraphT **/
281. DirectedGraph<String,DefaultWeightedEdge>directedGraph=newSimpleDirectedGraph<String,DefaultWeightedEdge>(Def
     aultWeightedEdge.class);
282.
283.    /**  Add vertexes to directed JGraph  */
284.    for (int vertex = 0; vertex < diGraph.getNodesSize(); vertex++) {
285.       directedGraph.addVertex(diGraph.getN(vertex));
286.    }
287.
288.    /**  Add directed edges. sAS[0] sDomain[1] sArea[2] sRouter[3] sTP[4] dAS[5] dDomain[6] dArea[7] dRouter[8]
     dTP[9]  */
289.    int sRouterColumn = 3; // 3 Refers to the column in the Bidirectional graph from the JSON file.
290.    int dRouterColumn = 8; // 8 Refers to the column in the Bidirectional graph from the JSON file.
291.    for (int edge = 0; edge < diGraph.getGraphLength(); edge++) {
292. directedGraph.addEdge(diGraph.getGraph(sRouterColumn, edge), diGraph.getGraph(dRouterColumn, edge));
```

```
293.    }
294.
295.    /**  Find k shortest paths to all nodes.  K-represents the max number of shortest paths the algorithm
        will  find. */
296. KShortestPaths<String,DefaultWeightedEdge>kShortestPathsToAll=newKShortestPaths<String,DefaultWeightedEdge>(dir
        ectedGraph,kMaxPaths);
297.
298.    /**  Sorts through values generated by k shortest and return path associated with stated source and
        destination. */
299. List<ArrayList<GraphPath<String,DefaultWeightedEdge>>>kShortestPathsToDemands=newArrayList<ArrayList<Gr
        aphPath<String,DefaultWeightedEdge>>>();
300.
301.    for (int numOfDemands = 0; numOfDemands < source.length; numOfDemands++) {
302. kShortestPathsToDemands.add((ArrayList<GraphPath<String,DefaultWeightedEdge>>)kShortestPathsToAll.getPaths(diGr
        aph.getN(source[numOfDemands]), diGraph.getN(destination[numOfDemands])));
303.    }
304.
305.    /**  Print paths for each demand.  */
306.    for (int demand = 0; demand < kShortestPathsToDemands.size(); demand++) {
307.       System.out.println("Paths for demand .."+demand);
308.       for (int n = 0; n < kShortestPathsToDemands.get(demand).size(); n++) {
309.          System.out.println(kShortestPathsToDemands.get(demand).get(n));
310.       }
311.    }
312.    return kShortestPathsToDemands;
313. }
314.
315. /** MinMax */
316. public static double[] optimiseNetworkUsingCPLEX_MinMax(double ce_SF, ArrayList<String> listOfEdges, DiGraph
        diGraph, List<ArrayList<GraphPath<String,
        DefaultWeightedEdge>>> kShortestPathsToDemands, double[] ce, double h[]){
317.    ArrayList<ArrayList<String>> pathsListToDemands = new ArrayList<>();
318.    for (int numOfDemands = 0; numOfDemands < h.length; numOfDemands++) {
319.       pathsListToDemands.add(new ArrayList<String>());
320.       for (int path = 0; path < kShortestPathsToDemands.get(numOfDemands).size(); path++) {
321. pathsListToDemands.get(numOfDemands).add(kShortestPathsToDemands.get(numOfDemands).get(path).toString());
322.       }
323.    }
324.    int numberOfnodes  = diGraph.getNodesSize();
325.    double[] edgeCapacityArrayMaxMin = OptNetv3.MinMax(ce_SF, ce, numberOfnodes, listOfEdges, pathsListToDemands,
        h);
326.    return edgeCapacityArrayMaxMin;
327. }
328.
329. /** MinCost */
330. public static double[] optimiseNetworkUsingCPLEX_MinCost(double ce_SF,
        ArrayList<String> listOfEdgesInDigragh,  DiGraph diGraph, List<ArrayList<GraphPath<String,
        DefaultWeightedEdge>>> kShortestPathsToDemands, double[] ce, double h[]) {
331.    ArrayList<ArrayList<String>> pathsListToDemands = new ArrayList<>();
332.    for (int numOfDemands = 0; numOfDemands < h.length; numOfDemands++) {
333.       pathsListToDemands.add(new ArrayList<String>());
334.       for (int path = 0; path < kShortestPathsToDemands.get(numOfDemands).size(); path++) {
335. pathsListToDemands.get(numOfDemands).add(kShortestPathsToDemands. get(numOfDemands).get(path).toString());
336.    }
337.    }
338.    int numberOfnodes = diGraph.getNodesSize();
```

```
339. double[]edgeCapacityArrayMinCost=OptNetv3.MinCost(ce_SF,ce,numberOfnodes,listOfEdgesInDigragh,pathsListToDema
     nds,h);
340.     return edgeCapacityArrayMinCost;
341. }
342. /** List of Edges */
343. public static ArrayList<String> listOfEdgesInDigraph(DiGraph diGraph) {
344.     ArrayList<String> listOfedges = new ArrayList<>();
345.     for (int i = 0; i < diGraph.getGraphLength(); i++) {
346.         String sourceR = diGraph.getGraph(3, i); // Source router col 3
347.         String destinationR = diGraph.getGraph(8, i); // Dest router col 8
348.         listOfedges.add("(" + sourceR + " : " + destinationR + ")"); // (source:destination)
349.     }
350.     return listOfedges;
351. }
352.
353. public static void configurePathUsingPCEP() {
354.     LSP lsp = new LSP();
355.     lsp.create();
356. }
357.
358. /** Offline */
359. public static DiGraph offline(DiGraph diGraph) {
360.     /* RETRIEVE -- diGraph from JSON */
361.     String filePath = "C:\\Users\\brandta.ADS\\Dropbox\\Development\\gitStuff\\2015\\REST12_MultiTime_Hose\\";
362.     String topoFilenameRetreive     = "jTopoArray4NodeMesh.json";
363.     System.out.println("Topology used = "+topoFilenameRetreive );
364.     /**   Build directional graph from stored json topology file when network in not available.   */
365.     System.out.println("Building diGraph using json topology file .....");
366.     diGraph = buildDiGraphFromFile(filePath, topoFilenameRetreive);
367.     return diGraph;
368. }
369.
370. /** Online */
371. public static DiGraph online(DiGraph diGraph) {
372.     /* SAVE -- diGraph from REST saved locally */
373.     String baseURL = "http://10.27.23.191:8181"; // SDN_Multi_AS
374.     String topoFilenameSave = "jTopoArray4NodeFMesh.json"; // Topology stored locally after reading from REST.
375.     System.out.println("Topology saved = " + topoFilenameSave);
376.     /**  Build directional graph from REST when network is available.  */
377.     System.out.println("Building diGraph using REST .....");
378.     diGraph = buildDiGraphFromRest(topoFilenameSave, baseURL);
379.     return diGraph;
380. }
381.
382. /** Save file */
383. public static void saveFilev2(double d_SF, double ce_SF, String pathCPLEXoutput, StringfilenameCPLEXoutput,List<Ar
     rayList<String>> edgeCapacityArrayList,
     ArrayList<String> listOfEdges, double[] h, intTW,List<ArrayList<String>> demandVolume) {
384.     try {
385.         String path = pathCPLEXoutput + filenameCPLEXoutput;
386.         File file = new File(path);
387.         if (!file.exists()) {
388.             file.createNewFile();
389.         }
390.         System.out.println("\nWRITE to file .... ");
391.         FileWriter fw = new FileWriter(file.getAbsoluteFile());
392.         BufferedWriter bw = new BufferedWriter(fw);
```

```
393.
394.        // Demand label
395.        for (int i = 0; i < h.length; i++) {
396.            bw.write("d"+i + " \t"); // A
397.        }
398.        bw.write("\t");
399.        bw.write("Demand\t");
400.        /* Get the edge names e.g. [32-31], and writes it to the top of the file as a label. */
401.        for (int i = 0; i < listOfEdges.size(); i++) {
402.            bw.write("e"+i+"\t");
403.        }
404.        bw.write("Net Utilsed Cap %\t");
405.        bw.newLine();
406.
407.        // Write edge capacities to file
408.        System.out.println("\nWriting capacity ....");
409.         for (int Loop = 0; Loop <= TW; Loop++) {
410.            for (int n = 0; n < demandVolume.get(0).size(); n++) {
411. bw.write(String.valueOf(Double.parseDouble(demandVolume.get(Loop).get(n))*(1/d_SF))+ "\t");
412.            }
413.            /**   Write the demand.  */
414.            bw.write("\t");
415.            bw.write("TW(" + Loop + ")\t");
416.            /**  Write the capacity per edge.  */
417.            for (int i = 0; i < edgeCapacityArrayList.get(Loop).size(); i++) {
418. bw.write(String.valueOf(Double.parseDouble(edgeCapacityArrayList.get(Loop).get(i))* (1/d_SF)/1e9)+"\t");
419.            }
420.            bw.newLine();
421.        }
422.        bw.newLine();
423.        bw.close(); // Close connection
424.        System.out.println("\nClosing bw.close !!!!");
425.        } catch (Exception e) { System.out.println(e); }
426. }
427. }
```

9.6 DiGraph.java

Setter and getter file for constructing the directional graph.

```
1.  package sdnGraph;
2.
3.  import java.util.ArrayList;
4.  import java.util.List;
5.
6.  public class DiGraph {
7.      private ArrayList<String> nodes       = new ArrayList<String>();
8.      private List<ArrayList<String>> Graph = new ArrayList<>();
9.
10.     private void CheckRowExist(int row) {
11.         if ((this.Graph.size() - 1) == 0 || (this.Graph.size() - 1) < row) {
12.             this.Graph.add(new ArrayList<String>());
13.         }
14.     }
```

```
15.
16.     public void setGraph(int k, String Sas, String Sdomain, String Sarea, String Srouter, String Stp, String Das,
17.             String Ddomain, String Darea, String Drouter, String Dtp) {
18.
19.         String[] temp = { Sas, Sdomain, Sarea, Srouter, Stp, Das, Ddomain, Darea, Drouter, Dtp };
20.
21.         for (int row = 0; row < temp.length; row++) {
22.             //System.out.println("row = "+row+ "  k= "+k);
23.             this.CheckRowExist(row);    // Check if the row exists. Add the row if not exist.
24.             //System.out.println("Graph size after checking = "+this.Graph.size());
25.             Graph.get(row).add(temp[row]);  // Set the value
26.         }
27.     }
28.
29.     public String getGraph(int m, int n) {
30.         return Graph.get(m).get(n);
31.     }
32.
33.     public void setN(int m, String string) {
34.         nodes.add(m, string);
35.     }
36.
37.     public int getGraphLength(){
38.         return this.Graph.get(0).size();
39.     }
40.
41.     public String getN(int m) {
42.         return nodes.get(m);
43.     }
44.
45.     public int getNodesSize(){
46.         return nodes.size();
47.     }
48. }
```

9.7 GetJsonTopoArray.java

Receives a JSON file from REST with the topology.

```
1.  package sdnGraph;
2.
3.  import java.io.FileReader;
4.  import java.io.IOException;
5.  import java.io.PrintWriter;
6.  import javax.ws.rs.client.Client;
7.  import javax.ws.rs.client.ClientBuilder;
8.  import javax.ws.rs.core.MediaType;
9.  import javax.ws.rs.core.Response;
10. import org.json.simple.JSONArray;
11. import org.json.simple.JSONObject;
12. import org.json.simple.parser.JSONParser;
13.
14. public class GetJsonTopoArray {
15.     public JSONArray fromFile(String jsonRawTopoFilePath) {
16.         try {
17.             FileReader topoReader = new FileReader(jsonRawTopoFilePath);
```

```
18.             JSONArray jTopoArray = (JSONArray) new JSONParser().parse(topoReader);
19.             return jTopoArray;
20.         } catch (Exception e) {
21.             e.printStackTrace();
22.              return null;
23.         }
24.     }
25.
26.     public JSONArray fromREST(String getTarget, String topologyFilenameSave) {
27.         String username = "admin";
28.         String password = "admin";
29.         String usernameAndPassword = username + ":" + password;
30.         String authorisationHeaderName = "Authorization";
31.         String authorisationHeaderValue ="Basic"+java.util.Base64.getEncoder().encodeToString
32.         (usernameAndPassword.getBytes());
33.         Client client = ClientBuilder.newClient();
34.         JSONParser parser = new JSONParser();
35.
36.         try {
37.             Response netTopoResponse = client.target(getTarget).request(MediaType.APPLICATION_JSON).
38.             header(authorisationHeaderName,authorisatin HeaderValue).get(Response.class);
39.             System.out.println(netTopoResponse.getStatus());
40.             if (netTopoResponse.getStatus() != 200)
41.                 throw new RuntimeException("Failed : HTTP error code : " + netTopoResponse.getStatus());
42.
43.             String output = netTopoResponse.readEntity(String.class);
44.             // System.out.println(output);
45.             Object object = parser.parse(output);
46.             JSONObject jObject = (JSONObject) object;
47.             JSONArray jTopoArray = (JSONArray) jObject.get("topology");
48.
49.             try {
50.                 System.out.println("Writing jTopoArray to file = "+ topologyFilenameSave);
51.                 PrintWriter writer = new PrintWriter(topologyFilenameSave, "UTF-8");
52.                 writer.println(jTopoArray);
53.                 writer.close();
54.             } catch (IOException e) {System.out.println("Error writing to file !!!");
55.             }
56.             return jTopoArray;
57.         } catch (Exception e) {
58.             e.printStackTrace();
59.             return null;
60.         }
61.     }
62. }
63.
```

9.8 GetLinkArray.java

Constructs a link array.

```
1.  package sdnGraph;
2.
3.  import org.json.simple.JSONArray;
4.  import org.json.simple.JSONObject;
```

```
5.
6.  public class GetLinkArray {
7.      JSONArray linkData(JSONArray JSONTopoArray) {
8.      JSONObject jTopoArray_obj0 = (JSONObject) JSONTopoArray.get(0);
9.      JSONArray jsonlinkArray = (JSONArray) jTopoArray_obj0.get("link");
10.     return jsonlinkArray;
11.     }
12. }
```

9.9 GetNodeArray.java

Constructs a node array.

```
1.  package sdnGraph;
2.
3.  import org.json.simple.JSONArray;
4.  import org.json.simple.JSONObject;
5.
6.  public class GetNodeArray {
7.      JSONArray nodeData(JSONArray TopoArray) {
8.      JSONObject jTopoArray_obj0 = (JSONObject) TopoArray.get(0);
9.      JSONArray nodeArray = (JSONArray) jTopoArray_obj0.get("node");
10.     return nodeArray;
11.   }
12.
13.   public static void main(String[] args) {
14.     // TODO Auto-generated method stub
15.     GetJsonTopoArray getJsonTopoArray = new GetJsonTopoArray();
16.
17.     /** Getting link state topology from URL  0 pcepTopology 1 exampleLinkStateTopology  */
18.     String baseURL = "http://10.27.22.47:8181"; // sdn1 server in Office
19.     URL url = new URL();
20.     String topologyFilenameSave = "jTopoArray4Node.json";  //Topology stored locally after reading from REST.
21.     JSONArray TopoArray = getJsonTopoArray.fromREST(url.target(1, baseURL), topologyFilenameSave);
22.
23.     /** Getting link state topology from file. */
24.     String jsonRawTopoFile = "C:\\Users\\brandta.ADS\\Dropbox\\MyDocs\\1Research\\1PhD\\jsonTopo.json";
25.     JSONArray TopoArray = getJSonTopoArray.fromFile(jsonRawTopoFile);*/
26.
27.     /** Get the link array  An array containing all the link data is returned  */
28.     GetNodeArray getNodeArray = new GetNodeArray();
29.     JSONArray jsonNodeArray = getNodeArray.nodeData(TopoArray);
30.
31.     /** To print jsonLinkArray */
32.     System.out.println("GetNodeArray.size = " + jsonNodeArray.size());
33.     for (int G = 0; G < jsonNodeArray.size(); G++) {
34.       System.out.println("JSONNodeArray[" + G + "] = " + jsonNodeArray.toArray()[G] + " "
35.           + jsonNodeArray.toArray()[G].toString().length());
36.       }
37.   }
38. }
```

9.10LinkArrayProcessing.java

Deconstructs the JSON object into an array with all the network and link information in a flat file format.

```
1.  package sdnGraph;
2.
3.  import org.json.simple.JSONArray;
4.  import org.json.simple.JSONObject;
5.  public class LinkArrayProcessing {
6.      /**
7.       * This method constructs a four column array.  The column data represents the following:
8.       * <source-router>[0][0] <source-tp>[0][1] <dest-router>[0][2] <dest-tp>[0][3]
9.       */
10.      String[][] getGraph(JSONArray jsonLinkArray) {
11.      int numOfColumns = 4;   //each column represents an object eg, source nodes, source-tp, dest nodes, dest-tp
12.      String[][]  Graph              = new String[jsonLinkArray.size()][numOfColumns];
13.      String[]    SourceRouterArray  = new String[jsonLinkArray.size()]; // column 0
14.      String[]    SourceTPArray      = new String[jsonLinkArray.size()]; // column 1
15.      String[]    DestRouterArray    = new String[jsonLinkArray.size()]; // column 2
16.      String[]    DestTPArray        = new String[jsonLinkArray.size()]; // column 3
17.
18.      int LinkArrayIndex  = 0;
19.      int sourceRouter    = 0;
20.      int sourceTP        = 1;
21.      int destRouter      = 2;
22.      int destTP          = 3;
23.
24.      while (LinkArrayIndex < jsonLinkArray.size()) {
25.         /* Get source object  */
26.         JSONObject LinkObject           = (JSONObject) jsonLinkArray.get(LinkArrayIndex);
27.         JSONObject SourceObject         = (JSONObject) LinkObject.get("source");
28.         JSONObject DestinationObject    = (JSONObject) LinkObject.get("destination");
29.
30.         /* Get source-router and source-tp object  */
31.         SourceRouterArray[LinkArrayIndex]   = (String) SourceObject.get("source-node");
32.         SourceTPArray[LinkArrayIndex]       = (String) SourceObject.get("source-tp");
33.
34.         /* Get destination-router and destination-tp object  */
35.         DestRouterArray[LinkArrayIndex]     = (String) DestinationObject.get("dest-node");
36.         DestTPArray[LinkArrayIndex]         = (String) DestinationObject.get("dest-tp");
37.         //DiGraph = [sourceNode] [SourceTP] [DestNode] [DestTP]
38.         Graph[LinkArrayIndex][sourceRouter] = SourceRouterArray[LinkArrayIndex].toString();
39.         Graph[LinkArrayIndex][sourceTP]     = SourceTPArray[LinkArrayIndex].toString();
40.         Graph[LinkArrayIndex][destRouter]   = DestRouterArray[LinkArrayIndex].toString();
41.         Graph[LinkArrayIndex][destTP]       = DestTPArray[LinkArrayIndex].toString();
42.         LinkArrayIndex++;
43.       }
44.       return Graph;
45.    }
46.
47.    int getNodes(JSONArray jsonNodeArray){
48.     int E = jsonNodeArray.size();
49.     return E;
50.    }
51. }
```

9.11NodeArrayProcessing.java

Populates the router array.

```
1.  package sdnGraph;
2.
3.  import org.json.simple.JSONArray;
4.  import org.json.simple.JSONObject;
5.  public class NodeArrayProcessing {
6.      /**
7.       * This method populates the RouterArray
8.       * The RouterArray vector is then returned
9.       */
10.     String[] getGraph(JSONArray jsonNodeArray) {
11.         String[] RouterArray = new String[jsonNodeArray.size()];
12.         int NodeArrayIndex  = 0;
13.         while (NodeArrayIndex < jsonNodeArray.size()) {
14.             /* Get source object  */
15.             JSONObject NodeObject           = (JSONObject) jsonNodeArray.get(NodeArrayIndex);
16.             RouterArray[NodeArrayIndex]     = (String) NodeObject.get("node-id");
17.              NodeArrayIndex++;
18.         }
19.         return RouterArray;
20.     }
```

9.12OptNetv3.java

Performs all the optimisation processes on the graph by allocating flows to the paths.

```
1.  package sdnGraph;
2.
3.  import java.util.ArrayList;
4.  import java.util.List;
5.
6.  import ilog.concert.IloException;
7.  import ilog.concert.IloLinearNumExpr;
8.  import ilog.concert.IloNumVar;
9.  import ilog.concert.IloObjective;
10. import ilog.cplex.IloCplex;
11. import ilog.cplex.IloCplex.UnknownObjectException;
12.
13. public class OptNetv3 {
14.
15. public static double[] MinMax(double SF, double[] ce, int nodes, ArrayList<String> listOfEdges,
    ArrayList<ArrayList<String>> pathsListToDemands, double h[]) {
16.
17.         try {
18.              IloCplex cplexMM = new IloCplex();
19.
20.             /**
21.             * Variable declaration
22.             */
23.             IloNumVar[][] x = new IloNumVar[h.length][5];
24.             for (int d = 0; d < pathsListToDemands.size(); d++) {
```

```
25.                 System.out.println("pathsListToDemands.size() ="+pathsListToDemands.size());
26.                  for (int p = 0; p < pathsListToDemands.get(d).size(); p++) {
27.                     System.out.println("pathsListToDemands.get(d).size() ="+pathsListToDemands.get(d).size()+" p="+p);
28.                     x[d][p] = cplexMM.numVar(0, Double.MAX_VALUE);
29.                     System.out.println("[d] "+d+" [p]"+p);
30.                  }
31.               }
32.
33.               IloNumVar[] edgeFlow = new IloNumVar[listOfEdges.size()];
34.               for (int e = 0; e < listOfEdges.size(); e++) {
35.                  edgeFlow[e] = cplexMM.numVar(0, Double.MAX_VALUE);
36.               }
37.
38.               /**
39.               * Objective function
40.               */
41.               // Expression
42.               IloNumVar r = cplexMM.numVar(0, Double.MAX_VALUE);
43.               IloLinearNumExpr F = cplexMM.linearNumExpr();
44.               F.addTerm(1.0, r);
45.               // Objective
46.               IloObjective obj = cplexMM.minimize(F);
47.               cplexMM.add(obj);
48.
49.               /**
50.               * Subject to:
51.               */
52.               // PATH FLOW The sum of the flows in the paths equals the demand.
53.               IloLinearNumExpr[] PathFlowMinMax = new IloLinearNumExpr[h.length];
54.               for (int d = 0; d < h.length; d++) {
55.                  PathFlowMinMax[d] = cplexMM.linearNumExpr();
56.                  for (int p = 0; p < pathsListToDemands.get(d).size(); p++) {
57.                     System.out.println("d="+d+" p="+p);
58.                     PathFlowMinMax[d].addTerm(1, x[d][p]);
59.                  }
60.                  cplexMM.addEq(PathFlowMinMax[d], h[d]);   //sum of  source-destination flows in each path equal  demand
61.               }
62.
63.               // The utilisation r, must be <= the edge capacity.
64.               for (int e = 0; e < listOfEdges.size(); e++) {
65.                  cplexMM.addLe(r, ce[e]);
66.               }
67.
68.               // The proportional flow per path must not exceed the utilisation.
69.               IloLinearNumExpr[] LinkFlowMinMax = new IloLinearNumExpr[listOfEdges.size()];
70.               for (int e = 0; e < listOfEdges.size(); e++) {
71.                  LinkFlowMinMax[e] = cplexMM.linearNumExpr();
72.                   String SrcDest = listOfEdges.get(e);
73.                  // System.out.println("\nSearching for edge "+SrcDest);
74.                  for (int d = 0; d < h.length; d++) {
75.                     for (int p = 0; p < pathsListToDemands.get(d).size(); p++) {
76.                        // System.out.println("p: "+p);
77.                         if (pathsListToDemands.get(d).get(p).contains(SrcDest)) {
78.                           LinkFlowMinMax[e].addTerm(1, x[d][p]);
79.                           //System.out.println("Found edge " + SrcDest + " in x[" + d + "][" + p + "]");
80.                           //System.out.println("addTerm [" + d + "][" + p + "] for edge " + e);
81.                        }
```

```
82.                     } // end for path
83.                 } // end for demand
84.
85.                 cplexMM.addEq(LinkFlowMinMax[e], edgeFlow[e]);
86.                 cplexMM.addLe(edgeFlow[e], ce[e]);
87.                 cplexMM.addLe(edgeFlow[e], cplexMM.prod(ce[e], r));
88.             }
89.
90.             //  The proportional flow must be less than the channel capacity.
91.             for(int d=0; d<pathsListToDemands.size(); d++) {
92.             for (int e = 0; e < listOfEdges.size(); e++) {
93.                 for (int p = 0; p < pathsListToDemands.get(0).size(); p++) {
94.                     if (pathsListToDemands.get(d).get(p).contains(listOfEdges.get(e))) {
95.                         cplexMM.addLe(x[d][p], ce[e]);
96.                         //System.out.println(d + "," + p + " edge " + e);
97.                     }
98.                 }
99.             }
100.            }
101.
102.            /**
103.            * 6. Control how much to display
104.            */
105.            cplexMM.setParam(IloCplex.IntParam.Simplex.Display, 0); // Record no information at all
106.            // cplexMM.setParam(IloCplex.IntParam.Simplex.Display, 1); // Display current  objective function
107.            // cplexMM.setParam(IloCplex.IntParam.Simplex.Display, 2); // Default value
108.
109.            /**
110.            * 7. Solve
111.            */
112.            System.out.println( "\nRetreiving MinMax results from solver..... OptNetv2.java");
113.             if (cplexMM.solve()) {
114.                 System.out.println("MinMax Solution status = " + cplexMM.getStatus());
115.                 System.out.println("obj = " + cplexMM.getObjValue());
116.                 for (int demand = 0; demand < h.length; demand++) {
117.                     for (int path = 0; path < pathsListToDemands.get(demand).size(); path++) {
118.                         System.out.println("x[" + demand + "][" + path + "]\t" + cplexMM.getValue(x[demand][path]));
119.                     }
120.                 }
121.
122.                 // Find the capacity associated with each path
123.                 double[] edgeCapacityArrayMinMax = FindTotalEdgeCapacityMinMax(cplexMM, pathsListToDemands, h,
    x,listOfEdges);
124.                 // OptNetv2.saveFile(edgeCapacityArray, listOfEdges, loop);
125.                 System.out.println("Printing edgeCapacityArrayMinMax");
126.                  for(int n = 0; n<edgeCapacityArrayMinMax.length; n++) {
127.                     System.out.println(edgeCapacityArrayMinMax[n]);
128.                 }
129.                 return edgeCapacityArrayMinMax;
130.             } else {
131.                 System.out.println("Min Max Model not solved");
132.                 double[] edgeCapacityArrayMinMax = { 0 };
133.                 // Break set variable to break the while loop. return edgeCapacityArry with all edges set to inf.
134.                 edgeCapacityArrayMinMax[0] = Double.POSITIVE_INFINITY;
135.                  cplexMM.end();
136.                 return edgeCapacityArrayMinMax;
137.             }
```

```
138.            // cplexMM.end();
139.        } catch (IloException exc) {
140.             exc.printStackTrace();
141.            System.out.println("ILO exeption MinMax");
142.        }
143.        return null;
144.    }
145.
146.    /**
147.     * h: Vector of commodities in Mb/s
148.     */
149.    public static double[] MinCost(double SF, double[] ce, int nodes, ArrayList<String> listOfEdges,
    ArrayList<ArrayList<String>> pathsListToDemands, double h[]) {
150.
151.        try {
152.            IloCplex cplexMC = new IloCplex();
153.
154.            /**
155.            * Flow Variables for Different paths. Path generated by k-shortest path
156.            * algorithm. pathsList contains paths based on demands between nodes.
157.            */
158.            IloNumVar[][] x2 = new IloNumVar[h.length][5];
159.            for(int d = 0; d < pathsListToDemands.size(); d++) {
160. //              for (int p = 0; p < pathsListToDemands.get(0).size(); p++) {
161.                for (int p = 0; p < pathsListToDemands.get(d).size(); p++) {
162.                    x2[d][p] = cplexMC.numVar(0, Double.MAX_VALUE);
163.                }
164.            }
165.
166.            /**
167.            * Variable declaration
168.            */
169.            IloLinearNumExpr F = cplexMC.linearNumExpr();
170.
171.            /**
172.            * Objective
173.            */
174.            // Determines the cost of each path based on hop count.
175.            List<ArrayList<Integer>> PathCost = OptNetv3.PathCost(pathsListToDemands, h, listOfEdges);
176.            for (int d = 0; d < h.length; d++) {
177.                for (int p = 0; p < pathsListToDemands.get(d).size(); p++) {
178.                    F.addTerm(PathCost.get(d).get(p), x2[d][p]);
179.                }
180.            }
181.            IloObjective obj = cplexMC.minimize(F);
182.            cplexMC.add(obj);
183.
184.            /**
185.            * Build the path Expressions and set equal to the demand
186.            */
187.            // PATH FLOW The sum of the flows in the paths equals the demand.
188.            IloLinearNumExpr[] PathFlowMinMax = new IloLinearNumExpr[h.length];
189.            for (int d = 0; d < h.length; d++) {
190.                PathFlowMinMax[d] = cplexMC.linearNumExpr();
191.                for (int p = 0; p < pathsListToDemands.get(d).size(); p++) {
192.                    PathFlowMinMax[d].addTerm(1, x2[d][p]);
193.                }
```

```
194.              cplexMC.addEq(PathFlowMinMax[d], h[d]); // the sum of the source-destination flows in each path must
195.                                                      // equal the demand
196.          }
197.          /**
198.           * Link Flow Constraints
199.           */
200.          // The sum of all the flows over an edge <= the capacity of that edge
201.          IloLinearNumExpr[] LinkFlowMinCost = new IloLinearNumExpr[listOfEdges.size()];
202.           int MaxBW_Factor = 1; // The maximum point to which the edge can be capacitated. 1 = 100%, 0.5 = 50%
203.          for (int e = 0; e < listOfEdges.size(); e++) {
204.              LinkFlowMinCost[e] = cplexMC.linearNumExpr();
205.              String SrcDest = listOfEdges.get(e); // Take every edge and search for it in pathsListToDemands
206.              for (int d = 0; d < h.length; d++) {
207.                  for (int p = 0; p < pathsListToDemands.get(d).size(); p++) {
208.                    if (pathsListToDemands.get(d).get(p).contains(SrcDest)) {
209.                        LinkFlowMinCost[e].addTerm(1, x2[d][p]);
210.                    }
211.                } // end for path
212.               } // end for demand
213.              cplexMC.addLe(LinkFlowMinCost[e], ce[e] * MaxBW_Factor);
214.          }
215.          // 6. Control how much to display
216.          cplexMC.setParam(IloCplex.IntParam.Simplex.Display, 0); // Record no information at all
217.          // cplexMinMax.setParam(IloCplex.IntParam.Simplex.Display, 1); // Display
218.          // current objective function
219.           // cplexMinMax.setParam(IloCplex.IntParam.Simplex.Display, 2); // Default value
220.
221.          // 7. Solve
222.          System.out.println( "CPLEX Solving MinCost equation  ..... OptNetv2.java");
223.          if (cplexMC.solve()) {
224.              System.out.println("MinCost Solution status = " + cplexMC.getStatus());
225.              System.out.println("obj = " + cplexMC.getObjValue());
226.              for (int demand = 0; demand < h.length; demand++) {
227.                  for (int path = 0; path < pathsListToDemands.get(demand).size(); path++) {
228.                      System.out.println("x[" + demand + "][" + path + "]\t" +
229.                       (cplexMC.getValue(x2[demand][path])));
230.                  }
231.              }
232.              double[] edgeCapacityArrayMinCost = FindTotalEdgeCapacityMinCost(cplexMC, pathsListToDemands, h, x2,
    listOfEdges);
233.              return edgeCapacityArrayMinCost;
234.
235.          } else {
236.              System.out.println("MinCost Model not solved");
237.              double[] edgeCapacityArrayMinCost = { 0 };
238.              // Break set variable to break the while loop.
239.              // return edgeCapacityArry with all edges set to inf.
240.               edgeCapacityArrayMinCost[0] = Double.POSITIVE_INFINITY;
241.              System.out.println("Printing edgeCapacityArrayMinCost");
242.              for(int n = 0; n<edgeCapacityArrayMinCost.length; n++) {
243.                  System.out.println(edgeCapacityArrayMinCost[n]);
244.              }
245.               cplexMC.end();
246.              return edgeCapacityArrayMinCost;
247.          }
248.          // ------------------------------END
249.      } catch (IloException exc) {
```

```
250.            exc.printStackTrace();
251.            System.out.println("MinCost ILO exeption");
252.        }
253.        return null;
254.    }
255.    public static List<ArrayList<Integer>> PathCost(ArrayList<ArrayList<String>> pathsListToDemands, double[] h,
256.            ArrayList<String> listOfEdges) {
257.        List<ArrayList<Integer>> PathCost = new ArrayList<>();
258.        for (int d = 0; d < pathsListToDemands.size(); d++) {
259.            PathCost.add(new ArrayList<Integer>()); // Add a new column for every loop
260.             for (int p = 0; p < pathsListToDemands.get(d).size(); p++) {
261.                int count = 0;
262.                for (int e = 0; e < listOfEdges.size(); e++) {
263.                    String edge = listOfEdges.get(e);
264.                    if (pathsListToDemands.get(d).get(p).contains(edge)) {
265.                        // increment count for every edge that is found in the path.
266.                        count++;
267.                    }
268.                }
269.                PathCost.get(d).add(p, count);
270.            }
271.        }
272.        return PathCost;
273.    }
274.    public static String SwopSrcDest(String SrcDst) {
275.        String source = SrcDst.substring(1, 11);
276.        String destination = SrcDst.substring(14, 24);
277.         String DstSrc = SrcDst.replace(destination, source);
278.        DstSrc = DstSrc.replaceFirst(source, destination);
279.        return DstSrc;
280.    }
281.    public static double[] FindTotalEdgeCapacityMinMax(IloCplex cplexMinMax,
    ArrayList<ArrayList<String>>pathsListToDemands, double h[], IloNumVar[][] x,ArrayList<String> listOfEdges) {
282.        double[] edgeCapacityArrayMinMax = new double[listOfEdges.size()];
283.        System.out.println("");
284.        for (int demand = 0; demand < h.length; demand++) {
285.            for (int path = 0; path < pathsListToDemands.get(demand).size(); path++) {
286.                try {
287.                    if (cplexMinMax.getValue(x[demand][path]) > 0) {
288.                        for (int edge = 0; edge < listOfEdges.size(); edge++) {
289.                            if (pathsListToDemands.get(demand).get(path).contains(listOfEdges.get(edge))) {
290.                                System.out.println("pathsListToDemands "+pathsListToDemands.get(demand).get(path));
291.                                edgeCapacityArrayMinMax[edge] = edgeCapacityArrayMinMax[edge] +cplexMinMax.getValue(x[deman
    d][path]);
292.                                System.out.println("edgeCapacityArrayMinMax "+edgeCapacityArrayMinMax[edge]+" demand
    "+demand+" path "+path+" edge"+ edge+"\n");
293.                            }
294.                        }
295.                    }
296.                 } catch (UnknownObjectException e) {
297.                    e.printStackTrace();
298.                } catch (IloException e) {
299.                    e.printStackTrace();
300.                }
301.            }
302.        }
303.        return edgeCapacityArrayMinMax;
```

```
304.    }// END public static void FindTotalEdgeCapacity
305.    public static double[] FindTotalEdgeCapacityMinCost(IloCplex cplexMinCost,
    ArrayList<ArrayList<String>>pathsListToDemands, double h[], IloNumVar[][] x2, ArrayList<String> listOfEdges) {
306.        double[] edgeCapacityArrayMinCost = new double[listOfEdges.size()];
307.        for (int demand = 0; demand < h.length; demand++) {
308.           for (int path = 0; path < pathsListToDemands.get(demand).size(); path++) {
309.              try {
310.                 if (cplexMinCost.getValue(x2[demand][path]) > 0) {
311.                    for (int edge = 0; edge < listOfEdges.size(); edge++) {
312.                       if (pathsListToDemands.get(demand).get(path).contains(listOfEdges.get(edge))) {
313.                         edgeCapacityArrayMinCost[edge] = edgeCapacityArrayMinCost[edge] +cplexMinCost.getValue(x2[dem
    and][path]);
314.                       }
315.                    }
316.                 }
317.              } catch (UnknownObjectException e) {
318.                 e.printStackTrace();
319.              } catch (IloException e) {
320.                 e.printStackTrace();
321.              }
322.           }
323.        }
324.        return edgeCapacityArrayMinCost;
325.    }// END public static void FindTotalEdgeCapacity
326. }
327.
```

9.13StringProcessing.java

Processes string variables by removing certain unwanted characters from the string.

```
1.  package sdnGraph;
2.  public class StringProcessing {
3.     String[] removeDelimterSourceRouter(String string, int k) {
4.        String[] stringSplit = string.split("[&\\=]");
5.        return stringSplit;
6.     }
7.     String[] removeDelimterSourceTP(String string, int k) {
8.        String[] stringSplit = string.split("[&\\=]");
9.        return stringSplit;
10.    }
11.    String[] removeDelimterDestRouter(String string, int k) {
12.       String[] stringSplit = string.split("[&\\=]");
13.       return stringSplit;
14.    }
15.    String[] removeDelimterDestTP(String string, int k) {
16.       String[] stringSplit = string.split("[&\\=]");
17.       return stringSplit;
18.    }
19.    String[] removeDelimterNode(String string, int k) {
20.       String[] stringSplit = string.split("[&\\=]");
21.       return stringSplit;
22.    }
23. }
```

9.14 URL.java

Removes delimiters from string.

```
1.  package sdnGraph;
2.  public class StringProcessing {
3.          String[] removeDelimterSourceRouter(String string, int k) {
4.          String[] stringSplit = string.split("[&\\=]");
5.          return stringSplit;
6.          }
7.          String[] removeDelimterSourceTP(String string, int k) {
8.           String[] stringSplit = string.split("[&\\=]");
9.          return stringSplit;
10.     }
11.     String[] removeDelimterDestRouter(String string, int k) {
12.         String[] stringSplit = string.split("[&\\=]");
13.         return stringSplit;
14.     }
15.         String[] removeDelimterDestTP(String string, int k) {
16.             String[] stringSplit = string.split("[&\\=]");
17.         return stringSplit;
18.          }
19.         String[] removeDelimterNode(String string, int k) {
20.             String[] stringSplit = string.split("[&\\=]");
21.         return stringSplit;
22.     }
23. }
```