Control Logic Distribution trade-offs in Software-Defined Wireless Networks

Ph.D.Dissertation

by

Mohammed Osman

Advisor: Dr. Josep Mangues-Bafalluy

Tutor: Prof. Miquel Soriano Ibáñez

Departament d'Enginyeria Telemàtica

Universitat Politècnica de Catalunya

Barcelona, April 2022

Abstract

In current mobile networks, data is increasing day by day due to massive Machine Type Communication (mMTC), emerging AR (Augmented Reality)/VR (Virtual Reality) applications and UltraHD (Ultra-High-Definition) or 360-degree streaming video and many more. The ever growing data demand in mobile networks requires high capacity to meet user satisfactions. 5G and beyond 5G networks are promised to handle the huge volume of data. To transport the large volume of data, a cost-effective high capacity transport solution is a key. In that flows, 5G and beyond 5G networks are moving towards much higher frequencies (e.g., mmWave) that provide more larger bandwidth, hence, reduces the coverage of the cells, which makes 5G and beyond 5G networks more denser. In order to manage the large number of cell sites (known as small cell), centralized network like SDN (Software-Defined Networking) can provide huge advantages by controlling and managing the SCs in an efficient way by enabling programmability in the networks.

The SDN architecture separates the data and the control planes of the networks. It logically centralizes the control of a network in a central point that is an SDN controller, which acts as a brain of the network and is in charge of telling each network node how to forward incoming packets by installing the appropriate forwarding rules. One of the main advantages it brings is programmability through this single entity (the logical controller) with which network management applications must interact to apply their policies. Through agreed-upon APIs, the network managers can exploit the full potential of SDN.

SDN generally assumes ideal control channels between the SDN controller and the network nodes, which may be true in a certain controlled context, data center, for instance, where control packets exchanged between the SDN controller and the forwarding nodes experience almost no loss or very low loss. This may not be the case in challenging environments (e.g., wireless condition) that are becoming more common due to dense deployment of small cells (SCs) with reduced coverage in 5G and beyond 5G deployments to meet the capacity demand. In 5G and beyond 5G use cases, cost-effective wireless transport networks are required to connect the SCs. In this context, mmWave technology is a good player to connect the SCs, as mmWave

provides larger radio spectrum chunks that in turn provide larger bandwidth and higher data rate.

To manage the dense deployment of SCs in the mobile networks, on the network management/control front, network programmability and virtualization are also an integral part of 5G and beyond 5G networks. In this regard, to provide end-to-end connectivity, management and orchestration of all the segments of the networks, ranging from RAN (Radio Access Network), to the transport network, and to the core, is vital. On the transport networks side (the main focus of the dissertation), SDN plays an important role, as SDN enables programmability and virtualization in the networks.

Though SDN provides huge flexibility in network management by splitting the control plane from the data plane, it has some limitations in the wireless networks context, as the separation of the control plane from the data plane introduces extra points of failure in the SDN-based control (e.g., control communication channel failure, SDN controller failure). In wide area network (WAN) scenarios, where in-band channels (e.g., microwave or mmWave links) are responsible to carry control traffic between the forwarding nodes and the SDN controller, the assumption of the availability of a reliable network may not be possible, as the performance of the wireless link changes with the environmental conditions, which leads to a high risk of experiencing channel impairments, which might cause centralized SDN operation failure by affecting communication between the transport component of SCs and the SDN controller.

Therefore, the reliability of SDN in the wireless context needs to be taken into account. In what follows, we have investigated the performance of centralized SDN during unreliable control communication channel conditions. In the dissertation, we propose that the forwarding node is responsible to inspect the reliability of the control communication channel. We also propose a metric named control packet loss ratio (CPLR) to measure the reliability of the control communication channel as well as the control plane. As control packet loss is the major concern to inspect the reliability of the control communication channel, from the point of view of a network node, the failure of the SDN controller is equivalent to the failure of the control communication channel. Thus, we introduce a local agent in the node that is coupled with a monitoring framework to detect the unreliability of the control

communication channel. The experimental results show that during a high loss regime, SDN fails to operate, as control packets that are exchanged between the SDN controller and the forwarding nodes experience high losses.

To recover the SDN control from failure, the dissertation also presents a hybrid SDN (Hybrid Software-Defined Networking) scheme that explores the benefits of centralized and distributed operations depending on control communication channel conditions. Our hybrid SDN approach combines both centralized and distributed modes in the same node to form a hybrid control plane architecture. We introduce a local agent in the node that is composed of a monitoring framework to detect reliability of the control communication channel and a decision module that embeds a novel control logic switching algorithm to make the decision whether to operate in a centralized or distributed mode. The monitoring framework periodically gathers control communication channel status by measuring CPLR and based on information gathered by the monitoring framework during various states, the decision module predicts the network conditions in advance and dynamically switch the mode of operations from centralized to distributed if required. We evaluate the proposed solution under a variety of unreliable network conditions (e.g., link impairments, control packet loss) to investigate the operational performance of the hybrid SDN during high loss conditions. The experimental results show that the proposed hybrid SDN solution substantially improves the aggregated throughput and latency, particularly when control channel packet loss ratios increase, which in turn keeps the network operational in hard conditions where centralized SDN would result in a non-operational network.

Our proposed solution shows huge potential in wireless transport solution of 5G and beyond 5G networks where a more denser networks is formed by deploying SCs or even in the scenario of non-terrestrial networks (NTNs) that is an increasing attraction in 5G and beyond 5G scenarios. Besides capacity, 5G and beyond 5G networks are committed to provide Ultra-Reliable Low-Latency Communication (URLLC), which requires high reliable communication channels to ensure exchange of control messages with the relevant nodes on time. But in wireless condition, the assumption of reliable communication channel may not be the case due to environmental condition. In such condition, advantages of SDN, which is a key player in 5G and beyond 5G notworks in control/management front, may not be exploited properly as

the SDN controller loses the control of the network. Our hybrid SDN, by integrating centralized and distributed operations can show robustness in such scenario by switching mode of operation. Our solution, by integration of both operations in the same node able to exploit the advantages of centralized operation and at the same time brings robustness to the network by quickly react to failure of the centralized operation.

Acknowledgements

I would extend my sincerest gratitude to my supervisor Dr. Josep Mangues-Bafalluy for his enormous support and guidance during the journey. He is a person who presents complex things in an easy way that helps me to understand several complex things. During the discussion, he used to discuss things from different perspectives that pushed me to think more constructively. His guidance helped me to choose the right direction and successfully complete the thesis.

I would like to acknowledge Dr. José Núñez-Martínez for his valuable guidance during the early stage of the journey. In the very beginning, he mentored me for a decent start and to choose the tools to formulate the thesis.

I would like to express my gratitude to Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA) that has provided me with a world-class environment to learn and overcome my shortcomings. In CTTC I found some colleagues with whom I spent really good time and used to discuss different aspects of the research work.

I would also like to convey my special thanks of gratitude to all of my mentors throughout my life who inspired me and guided me in different parts of my life.

Any attempt at any level would not be possible to complete successfully without the support of my family members. I would like to express my heartiest gratitude to my parents for their support during hard times. I would also like to thank my wife Lamia Naznin who stayed beside me and gave me mental support throughout the journey. Last but not least, my two little angels are the inspiration of every part of my life.

Contents

List of Figures

List of Tables

Abbreviations

AFRO	**A**utomatic **F**ailure **R**ecovery for **O**penFlow
API	**A**pplication **P**rogramming **I**nterface
AS	**A**utonomous **S**ystem
AR	**A**ugmented **R**eality
ARP	**A**ddress **R**esolution **P**rotocol
BFD	**B**idirectional **F**orward **D**etection
CCMM	**C**ontrol **C**hannel **M**aintenance **M**odule
CN	**C**ore **N**etwork
CORONET	**Co**ntroller Based **Ro**bust **Net**work
CPLR	**C**ontrol **P**acket **L**oss **R**atio
CPU	**C**entral **P**rocessing **U**nit
DSL	**D**igital **S**ubscriber **L**ine
FatTire	**Fa**ul**t** **T**olerat**i**ng **R**egular **E**xpression
FIB	**F**orward **I**nformation **B**ase
FIFO	**F**irst **I**n **F**irst **O**ut
GRE	**G**eneric **R**outing **E**ncapsulation
Hybrid SDN	**H**ybrid **S**oftware- **D**efined **N**etworks
LLDP	**L**ink **L**ayer **D**iscovery **P**rotocol
LOS	**L**ine-of-sight
LRS	**L**egacy **R**outer **S**erver
LSA	**L**ink **S**tate **A**dvertisement
LSDB	**L**ink **S**tate **D**atabase

LTE	**L**ong **T**erm **E**volution
mmWave	**M**illimeter **W**ave
MPLS	**M**ulti-**P**rotocol **L**abel **S**witching
NBI	**N**orth**b**ound **I**nterface
NETCONF	**N**etwork **C**onfiguration **P**rotocol
NG-SDH	**N**ext **G**eneration **S**ynchronous **D**igital **H**ierarchy
NLOS	**N**on-line-of-sight
NS	**N**etwork **S**imulator
NTN	**N**on-**t**errestrial **N**etwork
OAM	**O**peration, **A**dministration and **M**aintenance
OF-Config	**O**pen**F**low **C**onfiguration Protocol
OFP	**O**pen**F**low Protocol
OLSR	**O**ptimized **L**ink **S**tate **R**outing
O-RAN	**O**pen **R**adio **A**ccess **N**etwork
OSPF	**O**pen **S**hortest **P**ath **F**irst
OVSDB	**O**pen vSwitch **D**atabase **M**anagement **P**rotocol
QoS	**Q**uality **o**f **S**ervice
RAN	**R**adio **A**ccess **N**etwork
REST	**Re**presentational **S**tate **T**ransfer
SBI	**S**outh**b**ound **I**nterface
SBP	**S**DN **B**ased **P**ath
SC	**S**mall **C**ell
SCS	**S**DN **C**apable **S**witch
SCT	**S**olitary **C**onfinement **T**ree
SDN	**S**oftware-**D**efined **N**etworking
SPF	**S**hortest-**p**ath-**f**irst
TCP	**T**ransmission **C**ontrol **P**rotocol
TLS	**T**ransport **L**ayer **S**ecurity
URI	**U**niform **R**esource **I**dentifier

URLLC	**U**ltra-**R**eliable **L**ow-**L**atency **C**ommunication
VLAN	**V**irtual **L**ocal **A**rea **N**etwork
VM	**V**irtual **M**achine
VR	**V**irtual **R**eality
WAN	**W**ide **A**rea **N**etwork
WDN	**W**ireless **D**istributed **N**etwork
WiFi	**Wi**reless **Fi**delity
WiGig	**W**ireless **Gig**abit

Dedicated to my family

Chapter 1

Introduction

This chapter summarizes the general motivation behind the research work that has been carried out to tackle a research problem in today's communication networks. This chapter also presents main goal of the research work, the outline of the dissertation and a brief description of the contents of each chapter.

1.1 Motivation

With the improvement of new network technologies, Software-Defined Networking (SDN) [37] provides a new network paradigm as well as new network architecture to handle the programmability and the data growth in current and future data networks. Unlike legacy networks, where the forwarding plane and control plane coexist on the same node, the SDN architecture decouples the forwarding plane and the control plane. As a result, to build highly scalable and flexible networks to adapt to rapid changes in business requirements, it is necessary to substantially increase programmability, automation of the networks and more control over the networks.

On the other hand, in order to meet the anticipated growth of data demand of mobile networks, a significant evolution in mobile transport networks have been performed and the trend is set to continue for the foreseeable future. Due to the limited availability of radio spectrum, current mobile networks cannot provide the required throughput to the increased mobile traffic volumes. By deploying small

cells (SCs) [27], [29] with shorter cell radii, the capacity of the mobile networks can be increased by spatial reuse of the radio spectrum. Wireless technology (e.g., mmWave) to interconnect the SCs could be a good choice to provide a cost-effective mobile transport solution (including backhaul and fronthaul). The densification of mobile network infrastructure with SCs deployment requires proper control over the network and at the same time proper utilization of resources. The adaptation of SDN to wireless mobile transport networks can improve the management and control over the networks and at the same time provide better resources allocation in the mobile transport networks, by acquiring a global view of the networks as well as the centralized control over the networks.

In addition, SDN provides huge advantages for managing networks by splitting the control logic, which was previously tightly bound to the forwarding entity. Even though SDN provides better manageability and programmatic flexibility to innovative network applications, there are still important issues to solve, such as network failure handling, particularly that due to control plane failure. Therefore, the separation of the control plane from the forwarding plane leads to additional points of failure, i.e., the network administrator will have to pay additional attention to the control plane to handle the SDN controller failure. The SDN controller failure or control channel(s) failure may be really harmful for centrally managed SDN. The SDN controller acts as a brain of the network to manage several forwarding devices. So, in order to keep SDN alive, connection between the control plane and the forwarding plane must be consistent, i.e., communication between the control plane and the data plane must be fully operational. And in fact, this is what is assumed for high-capacity wired networks (e.g., data centers or campus networks), but this may not be the case for dense mobile networks and all-wireless transport networks with in-band control. In in-band control networks, data and control packets are carried out by the same network and in wireless contexts, the channels are more prone to impairments due to environmental conditions.

The motivation of the research work lays on the trade-off between centralized control and distributed control during unreliable network conditions. This research work tries to find the solution by implementing a hybrid control plane architecture where both centralized and distributed control co-exist in order to provide higher

reliability, resiliency and to improve the performance of the network during unreliable control plane conditions by alternating the control logic. In order to drive SDN-based networks in case of impairment, as well as to maintain operational resiliency, several kinds of research have been performed. Most of the researches are concentrated on impairment at the data plane level. Very little attention is paid to the control plane failure as well as control channel failure, or even degraded channel condition. On the other hand, in wireless networks, the radio links are more likely to be affected by environmental conditions and interference, which leads to a high risk of experiencing channel impairments.

1.2 Goal of The Thesis

The main objective of the dissertation is to design a network architecture to preserve simple network management by conceiving centralized network control and also to provide robustness during unreliable network conditions. To formulate the thesis, the main goals are defined as:

- Investigating performance of centralized SDN during unreliable conditions.
- Inspecting the performance of distributed operation during unreliable conditions.
- Defining a network metric to determine the status of the control communication channel as well as that of the control plane.
- Designing a network architecture to preserve the benefit of both centralized and distributed operations and to mitigate their limitations during different use cases.
- Designing and implementing an algorithm to perform network operation switching depending on the reliability of the control communication channel.

1.3 Outline of The Thesis

The dissertation has been organized into nine chapters.

- Chapter 2 presents the background of the work that includes a brief overview of SDN and OpenFlow. This chapter also includes an overview of various deployment models of hybrid SDN and also an overview of mobile transport networks.

- Chapter 3 describes the state-of-the-art solutions to prevent centralized SDN from failing or to handle them when they appear. We have categorized the related works to formulate the main building blocks of our work.

- Chapter 4 specifies the problem statement, which remains unsolved by the research community, and also describes the validity of the research question.

- Chapter 5 presents our proposed hybrid control plane architecture. This chapter also illustrates details of the hybrid node architecture, which is the main building block of our proposed hybrid control plane.

- Chapter 6 depicts a novel network operation switching algorithm that periodically infers the status of the control communication channel and takes network operation switching decisions in advance based on the reliability of the control communication channel.

- Chapter 7 starts with a comparison between existing frameworks that could be used to implement the solution. Finally, the chapter focuses on the framework that has been used to implement the proposed solution.

- Chapter 8 mainly focuses on the performance evaluation of the proposed approach under different use cases. This chapter also describes the evaluated scenarios and experimental setup.

- Chapter 9 concludes with the main findings of this research work and also describes future research directions.

Chapter 2

Background

The mobile transport networks lay between Radio Access Network (RAN) and Core Network (CN). Softwarization and virtualization of the network's control from a central point (e.g., SDN) may provide huge flexibility to manage the transport networks in an efficient way. On the other hand, combination of centralized control and distributed control (e.g., hybrid SDN) may introduce better reliability and robustness to the networks.

In this direction the background chapter introduces a generic overview of SDN, Hybrid SDN and mobile transport networks, which are the technologies that set the framework for this thesis.

2.1 Introduction to Software-Defined Networking

Software-Defined Networking (SDN) [37] is an emerging networking architecture in today's communication networking paradigm, which allows programmability in networking and also provides more granular controls over the networks. SDN brings openness in networking by exposing a network management API (Application Programming Interface) that also allows building virtual networks. The principal idea of separation of the control plane and the data plane provides the networks more manageability, adaptability and cost-effectiveness. Decoupling of the control plane from the data plane makes the network move from distributed control, like OSPF

(Open Shortest Path First) [51] to centralized logic. In this case, network intelligence is logically centralized in the software-based SDN controller. Moreover, the SDN controller maintains a global view of the whole network from network topology to the state of the network resources, like switches. Thus, traffic shaping and applying routing policies become more flexible compared to legacy networks. For instance, if a forwarding link is congested, the SDN controller can dynamically adjust new routing policies to avoid the congested route. One more beneficial aspect of SDN is that of centralized control over multi-vendor environments [37]. Devices like switches and routers from any vendor that is OpenFlow-enabled [49] can be controlled by the SDN controller and hence significantly reduces the operation and management complexity of a network.

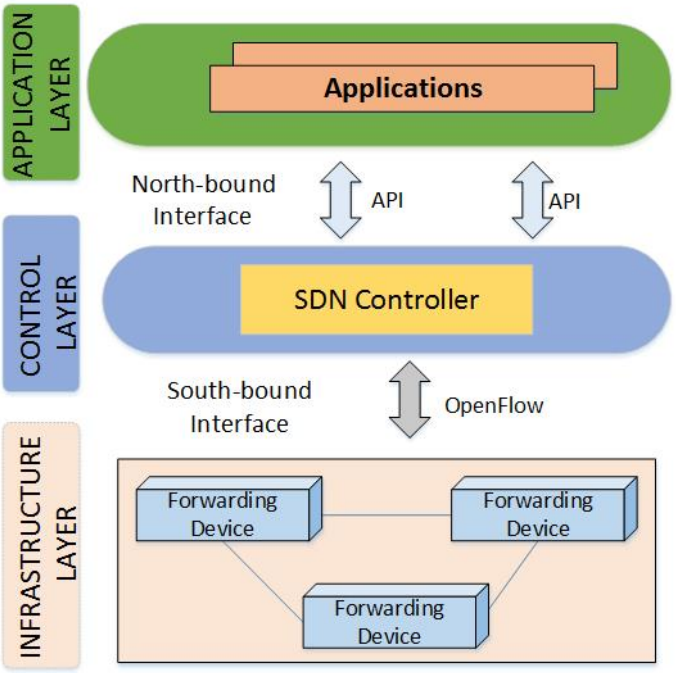

FIGURE 2.1: SDN Architecture

A logical overview of the SDN architecture has been depicted in Figure 2.1. The SDN architecture consists of three layers which are the application layer, the control layer and the infrastructure layer. For fulfilling user requirements as well as business needs, corresponding application resides in the SDN application layer. The SDN application layer communicates with the control layer by the northbound interfaces which provide services access points in various forms, for example, an Application Programming Interface (API) [71]. Several SDN applications like dynamic access

control, seamless mobility and migration, server load balancing and network monitoring can access and control the infrastructure layer devices (e.g., switch) through the control layer. The control layer is the bridge between the infrastructure layer and the application layer. The control layer maintains communication with the application layer and the infrastructure layer through the northbound and the southbound interfaces, respectively. The control layer is not only responsible for providing relevant information to the SDN applications, but also to collect network status and send the update of packets forwarding rules to the infrastructure layer devices. Several popular open-source SDN controller frameworks, such as NOX [7] , POX [17], Floodlight [3], OpenDayLight [13], Beacon [1], ONOS [10], and RYU [19] are available. The infrastructure layer is responsible for forwarding packets based on rules set up by the control layer. This layer mainly contains forwarding devices (e.g., Open vSwitch [16]) and network media. The SDN controller exchanges control information with the infrastructure layer devices via southbound interfaces using southbound protocols. NETCONF (Network Configuration Protocol) [5], OVSDB (Open vSwitch Database Management Protocol) [12], and OpenFlow [49], [14] are some examples of southbound protocols, which may comprise control but also management functions. Additionally, several OpenFlow-compliant commercial products have appeared in the market, as well as open-source ones, such as Open vSwitch [16]. As a consequence, OpenFlow is the most popular candidate among the other southbound protocols. With the introduction of SDN, designing and operating a network become significantly less complex for the network carriers and companies, since the infrastructure layer becomes vendor-independent.

2.1.1 OpenFlow Protocol

OpenFlow [49], [14] is the first standardized communication protocol between the control plane and the forwarding plane (also referred to as southbound protocol) in the SDN architectures. The main purpose of this protocol is to provide communication interfaces to the control and the infrastructure layer devices and at the same time to allow direct access and manipulation of the configuration stored in the forwarding devices (e.g., switches and routers).

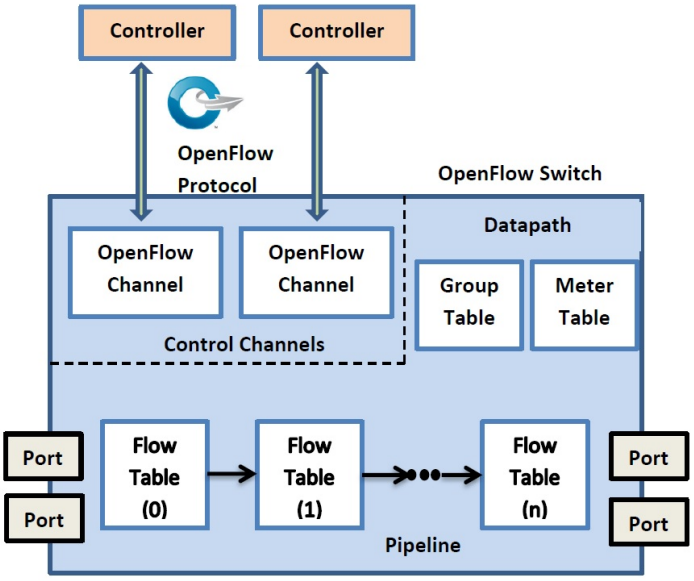

FIGURE 2.2: Components of OpenFlow Switch [15]

OpenFlow resides in both the control layer and the infrastructure layer to maintain communication between these two layers. The OpenFlow principle is based on three basic components (see Figure 2.2): (i) the data plane – which mainly consists of OpenFlow-compliant switches (ii) the control plane - which consists of one or more SDN controllers, and (iii) the secure control channel – which connects the data plane and the control plane [31]. The manipulation and population of flow entries of the flow tables in the OpenFlow-compliant forwarding devices are managed by the SDN controller through the control channels. Such channels may use plain TCP (Transmission Control Protocol) or can be encrypted using TLS (Transport Layer Security). The infrastructure layer devices (e.g., Open vSwitch) forward packets according to flow table entries.

2.1.1.1 OpenFlow Messages and Events

The OpenFlow protocol supports three types of messages that are listed below:

1. SDN controller to forwarding device message
2. Asynchronous message

3. Symmetric message.

The SDN controller initiates controller-to-forwarding device messages to manage or inspect the state of the forwarding node (e.g., Open vSwitch). While asynchronous messages are initiated by the forwarding node to give update to the SDN controller about network changes as well as forwarding node's state. On the other hand, Symmetric messages can be initiated either by the SDN controller or by the forwarding node. For instance, while entry of a new forwarding node is detected by the SDN controller, the SDN controller sends *Hello* message, which is a symmetric message and then followed by *Feature and Configuration* message that are SDN controller-to-forwarding device message (see Figure 2.3).

Different types of OpenFlow messages that are exchanged between the SDN controller and the forwarding nodes are described briefly in the Table 2.1.

2.1.2 Forwarding Device

In SDN architectures, the forwarding device forwards packets according to rules imposed by the SDN controller into its flow tables. The Open vSwitch [16] is one of the good candidates in this context. Open vSwitch is an open-source software switch to support OpenFlow and can be used as a virtual switch in virtualized environment. The Open vSwitch provides standard and vendor-independent management interfaces and at the same time enables the forwarding functions through programmatic control. The OpenFlow switch database contains one or more flow tables or group tables which contain set of rules imposed by the SDN controller. The SDN controller manipulates i.e., add, delete and modify the flow entries reactively or proactively using OpenFlow protocol.

TABLE 2.1: OpenFlow messages [15]

Message Type	**Message(s)**	**Description**
Symmetric	*Hello*	Exchanged between forwarding nodes and the SDN controller during bootstrap. OpenFlow versions are negotiated.

	Echo Request	Verify liveliness of the SDN controller and the forwarding nodes connection.
	Echo Reply	A reply message to Echo Request.
	Error Message	Mostly used by the forwarding node that indicates failure of a request that has been initiated by the SDN controller.
SDN controller-to-forwarding nodes	*Handshake*	OpenFlow *Features Request/Reply* messages to determine forwarding nodes' capability and information about forwarding nodes' ports.
	Configuration	The SDN controller sets and queries configuration parameters in the forwarding nodes (e.g., Timeout of flow rules).
	Flow Table Configuration	Flow tables are numbered and the SDN controller configures dynamic state in the flow tables.
	Modify-State	The SDN controller modifies state of the forwarding nodes. Add, delete or modify flow entries, for instance.
	Multipart	OpenFlow message that carries large amount of data and does not fit into single OpenFlow message is encoded by using multipart messages.
	Packet-Out	Policies that are set by the SDN controller to forward packets are sent to the data plane devices.
	Barrier	The SDN controller receives notifications of a completed operations and also ensure message ordering.
	Role Request	If the SDN controller wants to change its role, it sends role request.

	Bundle	The SDN controller can create, destroy and commit bundle messages.
	Set Asynchronous Configuration	Defines whether the SDN controller should receive a given asynchronous message that is generated by the data plane device.
Asynchronous	*Packet-In*	Triggered by the forwarding nodes to send a packet to the SDN controller for policy making,
	Flow-Removed	The data plane nodes instruct the SDN controller about removing of flow entries from its flow table.
	Port-Status	The forwarding nodes notify the SDN controller about topology change in the network. For instance, if link between two nodes goes up/down or port goes up/down.
	SDN Controller's Role Status	The forwarding node notifies the corresponding SDN controller when the SDN controller's role is changed by the forwarding node in multiple SDN controllers environment.
	Table Status	The SDN controller is notified by the forwarding nodes when table state is changed.
	Request Forward	In multiple SDN controllers environment, the data plane device informs other SDN controllers about the modification of the state of groups and meters.

	SDN Controller's Status	When status of a SDN controller changes the forwarding node sends status change message to all of the SDN controllers to which the forwarding node is connected.
	Error	The forwarding node sends *Error* messages to the SDN controller while an error is detected by the forwarding node.

2.1.2.1 Interfaces

In OpenFlow-enabled forwarding devices, OpenFlow ports are network interfaces for receiving and forwarding packets. Packets are received on an ingress port and go through a pipeline processing and then may be forwarded to an output port. There are three types of ports specified in OpenFlow switch specification [15] which are as follows:

- Physical Port: Physical ports are simply physical interfaces of the devices (e.g., Ethernet interfaces). In case of virtualized deployment of OpenFlow switches over switch hardware, OpenFlow physical port may represent a virtual slice of the corresponding hardware interface.
- Logical Port: Encapsulation of a packet and mapping different physical ports may be possible at these ports. Logical ports specify link aggregation groups, tunnels and loopback interfaces.
- Reserved Port: Responsible for generic forwarding actions. For instance, sending packets to the SDN controller, flooding or forwarding using traditional routing or switching. The required reserved ports that must be supported by the forwarding device are a follows:
  - ALL: Represents all the ports that can be used for forwarding of a specific packet.

– CONTROLLER: Represents the control communication channel with the SDN controller. When the port is used as an egress port, the packet is encapsulated in a *Packet_IN* messages and the packet is sent to the SDN controller using OpenFlow protocol. On the other hand, if the port is used as an ingress port, this identifies a packet originating from the SDN controller.

– TABLE: This port submits the packet to the first flow table for the OpenFlow pipeline processing and only valid in an output action, in the list of actions of a *Packet_Out* message.

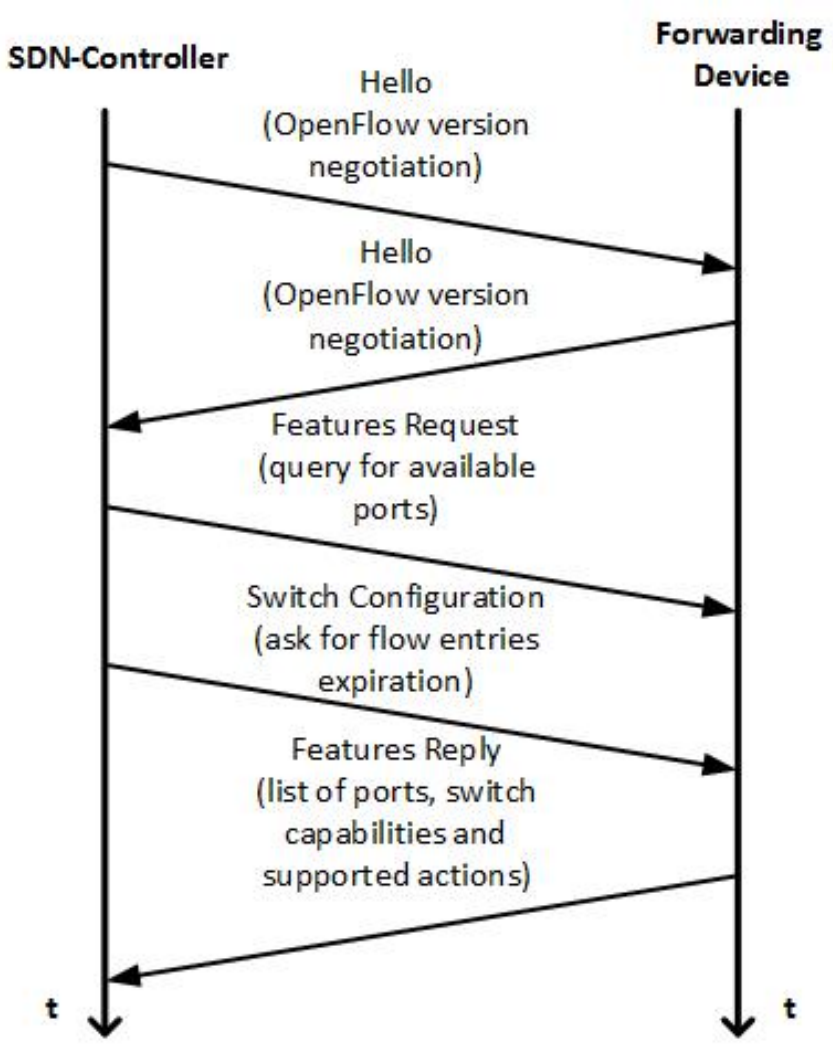

FIGURE 2.3: SDN controller and forwarding device handshaking during bootstrapping

- IN_PORT: Represents the packet ingress port and can also be used as an output port to send the packet out through its ingress port.
- ANY: When no port is specified, *ANY* as the port number allows to apply OpenFlow request instance to any and all ports.
- UNSET: To specify the output port that has not been set in the *Action-Set*.
- LOCAL: The local port enables remote entities (e.g., the SDN controller) to interact with the forwarding nodes via in-band connection rather than separate control network.
- NORMAL: Allows forwarding using the traditional non-OpenFlow pipeline of the forwarding devices.
- FLOOD: Allows flooding of packets using traditional non-OpenFlow pipeline of the forwarding nodes to all standard ports but not to the ingress port.

2.1.2.2 Flow Table Structure

A flow table is a set of flow entries that are stored in memory of a forwarding device. Each flow entry consists of several fields (see Figure 2.4) to apply matching and action to incoming packets.

FIGURE 2.4: Flow entry components of a Flow Table [15]

- Match Fields – check for matching with incoming packets. It comprises ingress port, packet header and metadata specified by a table.
- Priority – prioritize the matched flow entry.
- Counter – updated while packet header become matched.
- Instructions – a set of actions to take care of incoming packets.
- Timeouts – determines validity of a flow entry.

- Cookie – set by the SDN controller for filtering flow statistics, flow modification and deletion.
- Flags – defines the way of managing flow entry.

2.1.2.3 Packet Processing in OpenFlow-compliant Devices

Matching: When the forwarding devices (e.g., Open vSwitch) that are OpenFlow-enabled, receive a particular packet, they extract the packet header of the incoming packet to find the match with match field of the flow table(s). The forwarding nodes must contain at least one flow table and can contain more than one flow table. After receiving a packet, the forwarding node starts searching in its flow table(s) for the match and the search always starts at the first flow table i.e., search for the match against the flow entries stored in flow table 0 (zero). If the switch contains multiple flow tables and if no match is found in the first flow table (e.g., table 0) then the node starts searching for a match on the next flow table, and so on. This process is known as *OpenFlow pipeline* [15] process (see Figure 2.5). Pipeline processing happens in two stages, ingress and egress processing. Distinguishing between two stages is done by numbering the ingress and egress tables. Ingress tables always have lower numbering than the first egress table. Pipeline processing always starts with ingress processing from the first flow table. When the outcome of the ingress processing is to forward packet(s) to the output port, the forwarding device performs egress processing if valid egress table is configured as the first egress table, otherwise packet is simply forwarded out of the switch.

Table-miss: The table-miss flow entry of the flow table defines how to process the packet that has no match entry in the flow tables. For instance, the incoming packet that has no match may be dropped, or forwarded to the SDN controller. The table-miss flow entry rule has lower priority (in most cases it is zero (0)) in the flow table, so that the incoming packet can be matched against the higher priority rules, if there is any match. Otherwise the packet will be processed by the table-miss flow entry rule. The table-miss flow entry does not exist by default in the flow table. The controller may add this rule to the flow table of the forwarding devices during their bootstrap process and may further modify the rule if needed.

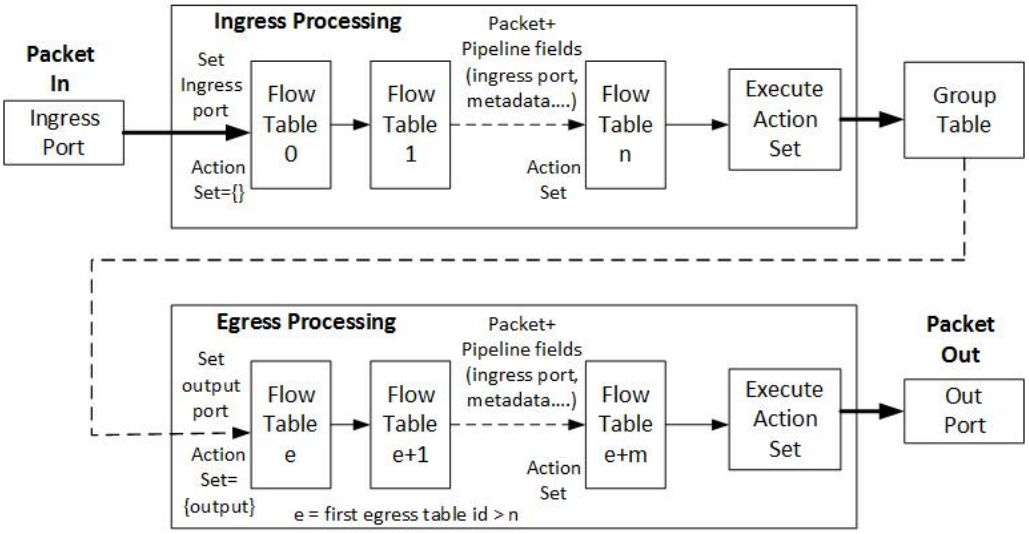

FIGURE 2.5: Pipeline processing for per packet flow [15]

While there is no match for incoming packet(s) in switch flow table(s), the table-miss flow entry is triggered, which specifies the action for the unmatched packet(s). Depending on the rules set by the SDN controller for unmatched packet(s), the unmatched packet(s) could be dropped by the forwarding node or the packet(s) could be sent to the SDN controller through the CONTROLLER reserved port (see section 2.1.2.1) using *Packet_In* (see Table 2.1) message for forwarding decision. If the packet is sent to the SDN controller, then the SDN controller is responsible to apply a policy to forward the packet. The SDN controller sets the priority, instruction, timeout, cookie and flag in the flow rule(s). The SDN controller also instructs the forwarding nodes to install the flow rule(s) into flow table(s) of the nodes and the nodes simply forward the packet(s) according to the instruction set by the SDN controller. When the next packet (same type) arrives, the forwarding node again searches for the match in its flow table(s). When there is match for the packet in its flow table(s), the flow entry with highest priority must be selected for that match. The Figure 2.6 depicts the flow chart of packet flow in an OpenFlow-compliant forwarding node.

Actions: Each flow entry of the flow table(s) contains a set of instructions that are executed while an incoming packet matches with any of the flow entries of the flow table(s). A set of *actions* that are enlisted in *actions-list*, are then executed in a cumulative way. The execution of *actions* from the *actions-list* follows a sequential

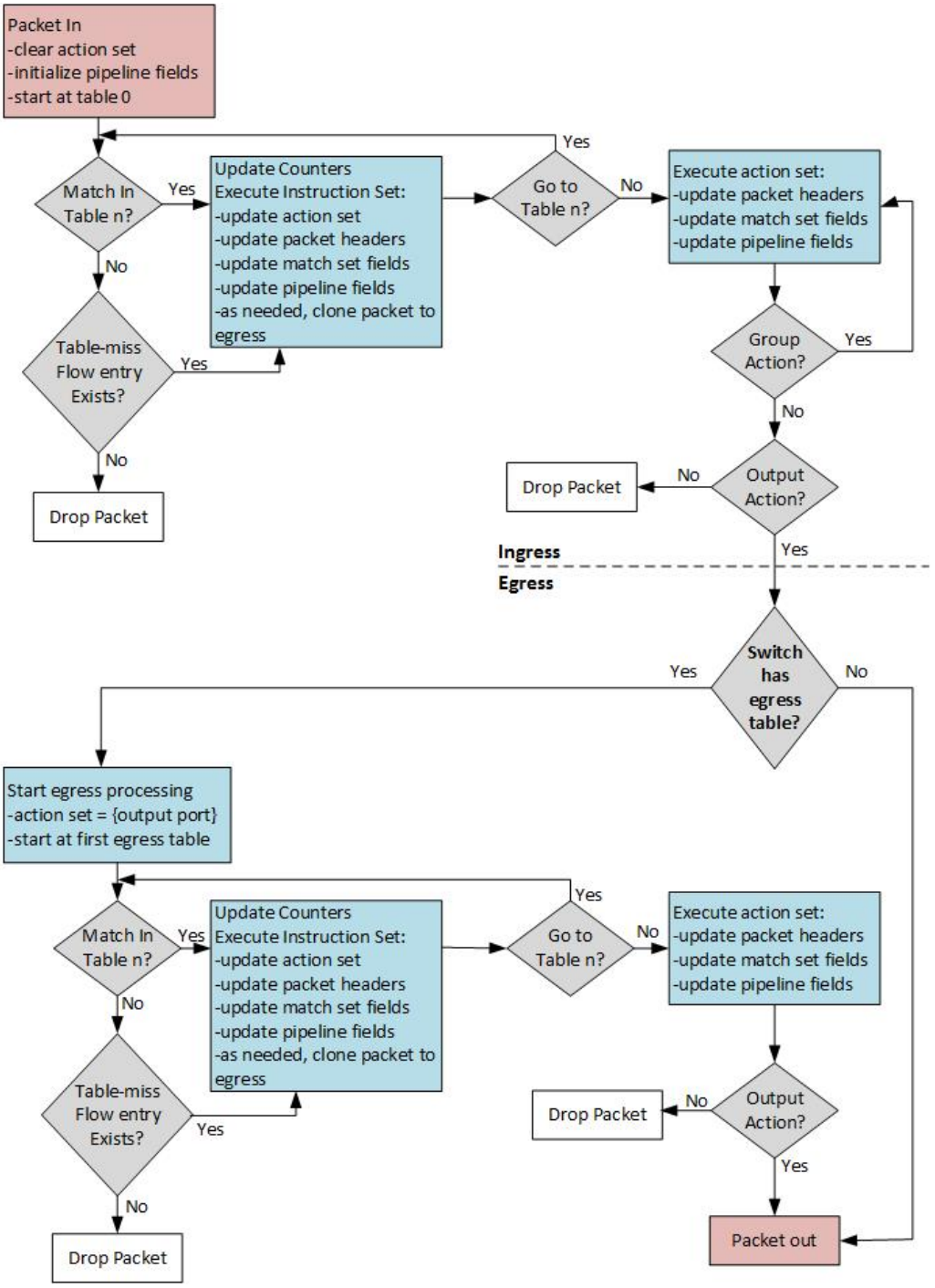

FIGURE 2.6: Flow chart of per-packet flow [15]

process. The Table 2.2 refers to a list of *actions*.

TABLE 2.2: OpenFlow Actions

Actions	**Description**
Output:*port_no*	Forwards a packet to a specified OpenFlow port for egress processing.
Group:*group_id*	Packets are processed through the specified group.
Drop	This action does not require explicit representation. Packets without any output action set or no group action must be dropped.
Set-Queue:*queue_id*	Sets the queue id for a packet and it is useful to provide basic Quality-of-Service (QoS).
Push-Tag/Pop-Tag:*ether-type*	This action push/pop tags (e.g., VLAN tags) to/from the packet header.

2.1.3 OpenFlow Control Channel

The OpenFlow control channel enables the communication between OpenFlow compliant forwarding nodes and the SDN controller. The OpenFlow control channel is generally encrypted using Transport Layer Security (TLS) or maintains TCP connection between forwarding nodes and the SDN controller. The OpenFlow-enabled forwarding nodes may support single or multiple control channels to have a connection with single or multiple SDN controller. The SDN controller configures and manages the forwarding devices via OpenFlow control channel. Moreover, the forwarding nodes also communicate with the SDN controller through the control channel for policy making of the incoming packets, update the SDN controller about network events or any ambiguity in the forwarding nodes state. The OpenFlow control messages that are exchanged between the SDN controller and the forwarding devices have been described in Table 2.1.

2.1.3.1 Connection URI

The SDN controller connection is identified by the forwarding nodes by a unique *Connection URI (Uniform Resource Identifier)*. Two forms of *Connection URI* are present.

1. protocol:name-or-address:port
2. protocol:name-or-address

The *protocol* field defines the transport protocol used for connection setup between forwarding nodes and the SDN controller. To set up a connection the acceptable value for the *protocol* field is *TLS* or *TCP*. The *name-or-address* field defines the hostname or IP address of the controller. The *port* field is the transport port on which the SDN controller listens to the connection. The default value of the transport port is 6653. An example of a typical *Connection URI* can be given as *tcp:192.168.10.1:6653*.

2.1.3.2 Connection Setup

The SDN controller and the forwarding nodes must establish a connection through either a user-specified transport port or the default OpenFlow transport port, 6653. The forwarding node initiates a standard connection according to *Connection URI*, which is a TLS or TCP connection. However, the forwarding nodes may allow the SDN controller to initiate the connection setup and accept the connection (TLS or TCP) set by the SDN controller.

During first establishment of OpenFlow connection, the forwarding devices and the SDN controller exchange *OFPT_HELLO* message, which includes OpenFlow protocol version supported by both sides. If the OpenFlow protocol version is negotiated successfully by the forwarding device and the SDN controller, the connection setup gets successful and standard OpenFlow messages can be exchanged over the connection. On the other hand, while OpenFlow version negotiation fails, the recipient replies with a *Hello Failed* error message and terminates the connection.

2.1.3.3 Connection Maintenance

The underlying transport mechanism of OpenFlow connection is TLS or TCP. Therefore, OpenFlow connection maintenance mostly depends on the TLS or TCP, i.e., detection of connection interruption mostly relies on TCP timeouts and TLS session timeouts. While the connection is broken between the SDN controller and the forwarding nodes, the re-connection attempt should be taken by the originator (either forwarding node or the SDN controller) of the connection. The connection originator keeps attempting to reconnect the other party until a new connection is established or until the *Connection URI* of the other party is removed from its configuration.

Processing of OpenFlow messages by the forwarding device may take longer time as OpenFlow messages are processed out of order [15]. Therefore, the SDN controller must keep the connection alive for too late reply except for the *echo replies* message. In case of late reply of *echo replies*, the SDN controller and the forwarding node may terminate the connection. However, this feature may be disabled by setting the timeout large enough [15]. Moreover, if the SDN controller can not process incoming OpenFlow messages rapidly, the SDN controller stops serving that connection to induce TCP flow control to stop the sender.

2.1.3.4 Connection Interruption

In case of control connection failure with the control plane, as a result of *echo* request timeouts, TLS session timeouts or any other impairments, the forwarding node, depending on its configuration immediately enters into *fail secure mode* or *fail standalone mode*. In case of *fail secure mode*, the packets that are destined to the SDN controller are dropped and the flow entries installed by the SDN controller expire depending on their timeout value. On the other hand, in case of *fail standalone mode* (available on hybrid nodes), the forwarding node acts as legacy Ethernet node and processes all the incoming packets using the *OFPP_NORMAL* reserved port.

2.2 Hybrid Software-Defined Networking Overview

The idea of hybrid SDN is quite imprecise. The idea behind hybrid SDN is to retain advantages of SDN (i.e., centralized) and the legacy network (i.e., distributed) as well. Though SDN eases operation and management of the networks, some limitation of SDN in terms of reliability, robustness and scalability are considered as SDN deployment challenges [65]. On the other hand, distributed networks provide better scalability by spreading control decisions over multiple devices. So, the idea of hybrid SDN is to integrate centralized and distributed networks to potentially sum advantages of centralized and distributed networks as well as at the same time to mitigate their corresponding limitations.

Hybrid SDN combines both centralized networks and the traditional distributed networks that mitigates the respective challenges of the centralized and the distributed networks. Table 2.3 presents four types of hybrid SDN models proposed in [65], which are *i) Topology-based hybrid SDN ii) Service-based hybrid SDN iii) Class-Based hybrid SDN iv) Integrated hybrid SDN.*

TABLE 2.3: Hybrid SDN Models [65]

Models	**Description**
Topology-based	More likely to be partitioned into centralized and distributed networks zone. In centralized zone all the nodes are logically controlled by the SDN controller. On the other hand in distributed zone, control logics are integrated into forwarding devices. In this context, communication between two zones is needed to forward packet(s) between any pair of source and destination in the network.
Service-based	Different services are provided by SDN and traditional networks separately. For instance, network-wide forwarding can be delegated to traditional networks while SDN can provide edge-to-edge services like enforcement of traffic engineering and access policies.

Class-based	A certain traffic class, e.g., TCP, can get service by SDN controlled classes while traditional networks can provide best-effort service to other traffic.
Integrated	All the network services are provided by SDN and traditional networks protocols are used to interface to node's Forward Information Base (FIB). For example, SDN can control forwarding path by injecting routes into routing system.

2.2.1 Advantages of Hybrid SDN

The canonical SDN (i.e., only centralized) architecture provides programmability in the networks by decoupling network logic from the forwarding devices that in turns eases management of the networks and also enforcement of several network policies (e.g., network security). But deployment of canonical SDN in the whole network at a time would be expensive solution as legacy network devices needed to be replaced by SDN-enabled forwarding devices. On the other hand, hybrid SDN models offer several advantages by containing legacy network devices and SDN-enabled devices in the same network.

A comparison between legacy network, canonical SDN and hybrid SDN is described in the Table 2.4.

TABLE 2.4: Comparison between different networking approaches

Attributes	**Legacy Network**	**Canonical SDN**	**Hybrid SDN**
Deployment Cost	Low, as existing network devices not needed to be replaced.	High, legacy network devices required to be replaced by SDN-enabled devices.	Moderate, legacy network devices can be used alongside with SDN-enabled devices.
Network Operation	Fully distributed.	Fully centralized.	Centralized and distributed.

Programmability	Not possible.	Fully programmable network. The SDN controller controls all the forwarding devices in the networks.	Partially programmable network. Some devices that are SDN-enabled are controlled by SDN controller while rest of the devices are operated in traditional fashion.
Protocols	Traditional protocols.	SDN-based protocol. OpenFlow protocol for instance.	Traditional protocol and SDN-based protocol.
Network Management	Difficult as there is no programmability.	SDN provides fine-grained control over the network by the feature of programmability that eases network management.	A portion of the network that requires fine-grained control, are managed centrally while the rest of the network uses traditional networking.
Reliability	High, in a sense that only one fault domain exist (e.g., data plane fault).	Low, as three fault domains can be identified (e.g., data plane fault, control plane fault and control channel fault).	High, based on deployment model network operation can be switched when one operation fails [54].
Robustness	High, as quickly reacts to failure.	Low, as fully dependent on centralized control, so during impairments all devices depend on SDN controller for policy making.	High, as a portion of the network is controlled centrally, so reaction time is lower during failure.

2.2.2 Different Deployment Models of Hybrid SDN

Hybrid SDN approaches are aimed to combine both traditional network and canonical SDN in the same network and offers a range of advantages that have been summarized in section 2.2.1. The deployment of the hybrid SDN principle can fall into two major categories:

- Deployment of SDN-enabled forwarding devices
- Deployment of hybrid forwarding devices

Deployment of SDN-enabled Forwarding Devices

In this category, SDN-enabled forwarding devices are placed alongside with legacy devices in order to form hybrid SDN. In this approach, small portion of the network contains SDN-enabled devices and are controlled centrally by the SDN controller to get fine-grained control for that portion of the network. On the other hand, rest of the portion of the network contains legacy IP devices and are operated in traditional network fashion. This approach is beneficial for a gradual upgrade of the network to adopt SDN. But this approach requires translation of complex distributed routing tables into SDN rules and vice-versa in order to maintain communication between SDN-enabled devices and the legacy devices.

Deployment of Hybrid Forwarding Devices

This deployment model focuses on integrating both centralized operation and distributed operation into the forwarding nodes. In order to enable SDN functionalities into the legacy devices, a hardware module can be integrated into the legacy devices so that SDN-based protocols can be used alongside legacy distributed protocols. According to [33], both legacy network protocols and SDN-based protocols can be operated in legacy devices by installing a hardware named *SDN shim*. Installation of hardware like SDN shim enables following hybrid SDN principles in the network. This deployment model is more concerned about reliability and robustness of the network. This approach recovers network connectivity during impairment by switching network operation [54]. This deployment model is of main interest in this dissertation.

2.3 Overview of Mobile Transport Networks

The mobile transport network infrastructure provides connectivity between the RAN and the CN (core network). A typical mobile access network consists of many base stations. On the other hand, the CN sites are very few. In this context the transport network provides transparent delivery of mobile network originated traffic by interconnecting large number of base stations to the core sites. Due to geographical issues, several network elements and processing requirements between access network and the core/control network, the mobile transport network is subdivided into two major domains, i) the access domain and ii) the aggregation domain (see Figure 2.7).

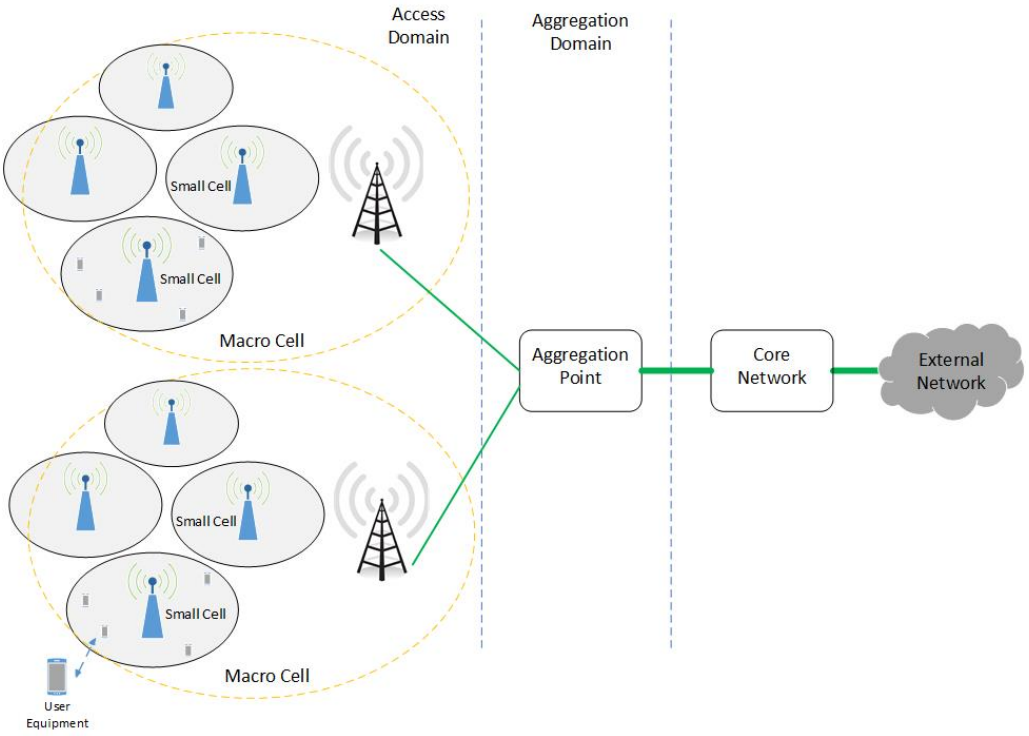

FIGURE 2.7: Transport network to connect SCs

2.3.1 The Access Domain

In the access domain, a number of small cells are interconnected to an access gateway (which may be co-located with the macro-cell). The access domain technologies are microwave links, DSL (Digital Subscriber Line), plain Ethernet and NG-SDH (Next

Generation Synchronous Digital Hierarchy). Among them, directional microwave links are the most popular option in such mobile transport links [26].

2.3.2 The Aggregation Domain

The aggregation domain aggregates traffic from the access domain. Hence, high capacity transport technology is required to support large number of traffic flows. This domain has traditionally adopted plain optical transport network and IP/MPLS (Multi-Protocol Label Switching) solutions to meet the demand. The aggregation domain connects the access domain to the core domain of the mobile networks.

2.3.3 Transport Solution to Connect Small Cells

Data traffic in mobile networks is growing exponentially day by day. In contrast, availability of radio spectrum is not enough in mobile networks like LTE (Long Term Evolution), to provide capacity to the growing demands [30]. The idea of small cells is to enhance coverage area of macro-cells by adding more base stations with reduced coverage. The cell with shorter radius is named as small cell (SC). The macro-cell site usually acts as a gateway site aggregating the traffic of a set of SCs and to connect SCs to the aggregation domain. By deploying SCs overall system performance can be enhanced by offloading some traffic to the SCs. In this context, the SC architecture can greatly increase capacity by reusing radio spectrum [30].

The number of SCs in a given macro-cell coverage can rise up to certain numbers that require large numbers of transport connections between SC sites to the access gateway. Offering the connectivity between SCs and the access gateway in a cost-effective way is a great challenge. Two types of transport technologies, namely wired and wireless, can be deployed to connect SCs to the access gateway.

2.3.3.1 Wired

Due to capacity concerns of the mobile networks, fiber-based technology can be a good option for the mobile transport solution. Fiber technology not only provides

capacity but also provides better reliability in wired transport solution, since there are no interference or NLOS/LOS issues. Though a fiber-based transport solution has some advantages, it has some drawbacks as well. The main drawback of fiber-based transport solution is its implementation cost. Laying fiber to connect all the SCs in a high density area would not be very cost-effective.

2.3.3.2 Wireless

Wireless transport solution to connect small cells can be divided into line-of-sight (LOS) and non-line-of-sight (NLOS). LOS links require a direct path between a transmitter and a receiver without any obstacle. On the other hand, refraction, diffraction and reflection are acceptable in signal propagation between a transmitter and a receiver in case of NLOS link. Carrier frequencies under 6 GHz are suitable for NLOS. In dense urban areas where high rise buildings create obstacles, NLOS offers better adaptability. Though NLOS links have advantages in urban areas, they have capacity limitations [29]. As LOS and NLOS have some drawbacks and advantages, depending on the necessity a better wireless transport solution can be made by composing LOS and NLOS. In this context, to implement a potential transport solution, the preference has been given to licensed sub-6 GHz for NLOS link and unlicensed 60 GHz band for LOS link [27].

Chapter 3

Related Works

This chapter focuses on previous works that are specific to the research problem that the thesis tackles.

Though SDN offers better manageability of the networks and at the same time provides flexibility to meet the current business demands over operational networks, it has some limitations in terms of network reliability. In traditional networks, the forwarding device (e.g. router) maintains a mechanism to react to topology changes as well as to link or node failures. By running a distributed protocol (e.g. OSPF [51]) on it, every node gets information about all nodes in the network by exchanging LSAs (Link State Advertisement). Such distributed information database (e.g., LSDB (Link State Database)) exploits to offer reliability in legacy networks. But in SDN, the forwarding device (e.g. switch) has no equivalent mechanism to react to network's impairments, which turns the device into a black hole if the forwarding device loses connection with the SDN controller. For example, in the event of link failure in the data plane, the forwarding device fully depends on the SDN controller to forward packet(s) through a new forwarding path.

Furthermore, in SDN, the SDN controller maintains a global view of the network, which is an advantage of SDN in terms of better resource allocation. But the SDN controller becomes a single point of failure. However, failure of the SDN controller is very much crucial in SDN, as the SDN controller is the brain of the network and the forwarding devices explicitly need forwarding decisions from the SDN controller to handle new incoming packets. If we consider the wireless networks with in-band

control scenario, handling SDN control unreliability becomes a relevant problem to maintain the network operational in case of failure.

With the goal of solving SDN limitations, several studies have been done so far. Two types of recovery mechanisms were conceived to react to failure in carrier-grade networks, which can be referred to as restoration mechanisms and protection mechanisms [64]. A protection mechanism is most likely proactive, i.e. resources are precomputed and reserved before failure occurs and it does not require additional signaling. On the other hand, a restoration mechanism is reactive, i.e. resources are assigned after a failure occurs and it requires additional signaling. In [62], both mechanisms were studied for SDN, which shows that failure recovery time using protection mechanisms is smaller than failure recovery time in restoration mechanisms in carrier-grade large-scale networks.

In the following subsections, an overview of research works done in the field of SDN that are most related to reliability of SDN are included. Addressing the reliability issue in SDN, three major fault domains can be identified, which are *data plane* fault, *control channel* fault and *SDN controller* fault. Work-related to every domain describes one by one in the next subsections. Moreover, this research work also focuses on hybrid SDN [65] approaches to tackle unreliability of centralized SDN by integrating centralized and distributed operations in the same network. Works related to hybrid SDN are also included in this chapter.

3.1 Reliability in Canonical SDN

The term reliability in networking can be defined as maintaining and providing an acceptable level of services during network failure. Reliability in carrier-grade networking can be achieved by assigning backup paths during a network failure. In SDN-based networks, there are potential points of failure not present in legacy distributed networks (i.e., with a distributed control plane). Specifically, we can identify three types of fault domains [44]:

1. *Data Plane Fault:* Network element(s) or port(s) associated to network element(s) failures.

2. *Control Plane Channel Fault:* Channel between the SDN controller and the data plane element(s) fails or degrades.

3. *SDN Controller Fault:* The SDN controller fails.

3.1.1 Data Plane Fault

In SDN networks, failure recovery due to data plane faults requires modification of flow entries in the flow tables for the corresponding affected path. The two recovery mechanisms, *protection* and *restoration*, are explained more in the next few subsections.

3.1.1.1 Protection Mechanism

In SDN, a protection mechanism relies on a proactive technique to quickly react to failures without intervention of the SDN controller, as shown in [62], [64], [43], [32]. Using the proactive technique, the SDN controller pre-installs rules into the forwarding devices (e.g. Open vSwitch) and also backup rules to avoid black holes in the network during link or node failure. This mechanism requires large numbers of flow rules for various kinds of flows, which in turn may overflow the memory space of the forwarding devices.

An approach presented in [32], utilizes protection scheme to overcome single link or single node failure. In this proposed approach, no additional signaling overhead is needed. A *crankback* technique is used to notify failures in the networks. When a link failure occurs between two nodes, the detecting node tags the data packet and using a *crankback* approach backtracks the data packet along the primary path up to the source node (also known as reroute node) that can determine an alternative path to the destination. As long as the first tagged packet is received by the reroute node, all the subsequent packets coming from the source node will be forwarded through the precomputed backup path to the destination. This approach requires to send the tagged packet to the source node to notify the source node about the link failure. But during the unreliable condition (e.g., degraded channel) the tagged

packet may also get dropped and the source node will never be aware of the link failure in the primary path.

In [43], a protection mechanism adopted to meet *Transport Applications* demand of network failure recovery time below 50 ms. In their proposed approach, a monitoring function is implemented in the data plane to reduce the processing load on the SDN controller. To detect link or node impairment in the data plane, an *Operation, Administration and Maintenance (OAM)* message generator is included in every switch. In contrast to Link Layer Discovery Protocol (LLDP)-based network monitoring, it sends OAM probe messages to the switches to discover data plane links and to maintain the network topology. Upon detection of the link failure, packets are rerouted through a backup path that is installed previously. This mechanism requires an extension of the OpenFlow protocol to process OAM packets.

A protection mechanism also considered in [62]. To overcome packet loss regarding flow entry restoration delay during link failure, the protection mechanism is adopted in this study. The group table concept specified by OpenFlow Switch Specification 1.3.1 [15] is considered in this paper to implement a protection mechanism. To perform fast failure recovery without the intervention of the SDN controller, the group table concept plays a vital role during a network failure. Unlike flow table, the group table comprises a set of group entries. A group entry contains group ID, group type and action buckets. An action bucket contains the alive status of the corresponding actions associated to one or more group types. The actions in the bucket are applied to the packets that are sent to the group type. A group can be linked to one or more action buckets. While network failure is detected by the *Bidirectional Forward Detection (BFD)* [42] technique, the forwarding path in the action bucket of the group table is declared unavailable by changing the value of the alive status. If the action bucket is declared unavailable, the packets are treated according to the next available bucket, which contains predetermined actions.

One of the main issues in a protection mechanism is that switch memory limitation to store a significant amount of flow table entries. By addressing this issue, authors from [57] present FatTire (**Fa**ul**t** **T**olerat**i**ng **R**egular **E**xpressions), a solution to optimize the number of backup flows as well as backup-links to overcome the limitation of switch memory to store more flow tables. FatTire is a new language for writing fault-tolerant network programs, which specify a set of legal paths with the

association of fault-tolerance requirements for those paths without the SDN controller intervention. FatTire creates an avenue for the programmer to set policies at path level that must be enforced all times, even at the time of network failure. It normalizes the policies by combining the separate policies into a single policy. By doing this, the number of backup paths can be reduced in the switch database.

Authors from [44], presented CORONET (**Co**ntroller Based **Ro**bust **Net**work), a system to recover SDN from multiple data plane link failures. CORONET provides a scalable and efficient fault tolerance system with multi-path support. It periodically collects topology information through its *Topology Discovery* module and receives asynchronous events, such as link or node failure events. A *Route Planning* module computes multiple routing paths based on topology information. Routing paths are calculated based on *VLAN growing* algorithm that creates multiple link-disjoint shortest routing paths using Dijkstra's algorithm.

3.1.1.2 Restoration Mechanism

In contrast to protection mechanisms, restoration mechanisms require the SDN controller intervention to handle network failure recovery. A restoration mechanism is somewhat related to reactive techniques. It is usually slower than the protection mechanism to compute a new route during link or node impairments in the networks, as this mechanism does not precompute the backup route in advance. While a link or node goes down, the forwarding device interacts with the SDN controller to get information about the new route. The SDN controller defines the new route in terms of flow rule and instructs the forwarding device to install the rule to restore the connectivity.

An approach regarding a restoration mechanism presented in [45]. Automatic Failure Recovery for OpenFlow (AFRO) illustrated in this paper works on two modes of operations: record mode and recovery mode. In record mode, AFRO records all the events of the SDN controller, such as *PacketIn*, *FlowMod* and *FlowRemove*. Once the failure in the network is detected, AFRO enters the recovery mode. It first spawns a new SDN controller instance that runs in an emulated environment that comprises the network topology eliminating the failed element. The new SDN controller, known as *shadow-controller*, is then filled up by AFRO with the recorded

PacketIn messages processed by the original SDN controller before the occurrence of the impairment. Once the replay ends, the *shadow-controller* contains a new forwarding state and pushes the flows to the forwarding devices.

In [62], the time taken by the restoration mechanism to restore connectivity was observed. In this paper, failure recovery using a restoration mechanism performed by the close intervention of the SDN controller during failure. Upon detection of a link failure, the affected nodes inform the SDN controller about the link failure in the data plane. Then, the SDN controller instructs the affected nodes to remove affected flow entries and install new flow entries to restore the connectivity. This is a slow process and packet(s) may be lost during the time interval from failure detection to completion of flow restoration.

One more approach named predetermined restoration illustrated in [61]. According to the authors of the paper, the administrator will provide all the paths in the SDN controller to reach the destination and set the priority among the paths. As soon as the SDN controller detects the link failure in the data plane, it immediately instructs the forwarding nodes to remove the affected flow entries. The SDN controller calculates new shortest path and instructs the forwarding nodes to install and establish a new path. This approach requires a static route to the SDN controller to send control traffic.

3.1.2 Control-Channel Fault

In the SDN paradigm, the SDN controller manages data plane nodes (e.g. switch) through secure control channels. The data plane nodes communicate with the SDN controller through these channels and the SDN controller instructs the data plane nodes to install flow rules through these channels. It is therefore important to highlight that failure of the control channel(s) will be hazardous in a centrally managed network like canonical SDN.

To prevent SDN from unexpected control channel failures, [70] presented Resilient-Flow, that restores the control channel through alternate paths (path redundancy)

in the presence of channel failure. A module named Control Channel Maintenance Module (CCMM) is implemented in the OpenvSwitch and the SDN controller. CCMM enables the forwarding nodes to detect control plane channel failures and its restoration through an alternative path. To detect control channel failure, CCMM monitors link status by exchanging heartbeat packets between two neighboring CCMMs (it can be CCMMs of two neighboring switches or switch and SDN controller). Furthermore, the CCMM also maintains flow entries of the forwarding node that connect the forwarding node to the SDN controller. Upon detection of the control link failure, the CCMM restores the control channel via an alternative control channel. To calculate path from the data plane elements to the SDN controller, the CCMM exchanges topology map with the neighboring CCMMs. After calculating path, CCMM does restoration of control channel by maintaining flow entries in the data plane element which connect the data plane element to the SDN controller. In this approach if the link between the SDN controller and the node that connects other nodes in the network to the SDN controller goes down, the whole network becomes non-operational, as the SDN controller loses control of the network.

3.1.3 SDN Controller Fault

In an SDN context, network functionalities (e.g. forwarding or routing decision) are centralized in the SDN controller. So, in an SDN architecture, SDN controller failures would have an adverse effect on the operation of the networks. For the sake of simplicity, initial SDN design and implementation focused on a single SDN controller. But the SDN controller may crash due to software bugs or hardware failures, which causes performance degradation of centralized networks .

Though the idea of multiple SDN controllers may lead to the solution of SDN controller failure, several issues must be taken into account to ensure correct network behavior. The OpenFlow version 1.3 supports the attachment of multiple SDN controllers with forwarding devices. But some inconsistent issues, like *Event Ordering*, *Unreliable Event Delivery* and *Duplication of Commands* are still a matter of concern.

The introduction of Ravana in [41] could be a solution to overcome the inconsistency concerns of multiple SDN controllers in SDN. Ravana proposes an extension

of OpenFlow interfaces to handle SDN controller failure. To ensure consistency during SDN controller failure, Ravana uses a two-stage replication protocol across the SDN controllers. Each stage involves adding event-processing information to the replicated in-memory log that keeps track of processed/unprocessed events and the log is shared among the controllers. In the two-stage replication procedure, the very first stage involves in reliably replicating events in the log and the second stage ensures event-processing is completed or not. In this approach one master and a slave controller is used to overcome SDN controller failure. The master replica of the SDN controller decides event ordering in the first stage and then indicates which event should be processed in the second stage. On the occurrence of master failover, the slave SDN controller resumes transaction for unprocessed events from the shared log. Ravana ensures event ordering, correct event processing and execution of commands for exactly once during the SDN controller failure but it requires the extension of OpenFlow.

The distributed SDN controller presented in [63], could be the solution for the aforementioned SDN contrller fault. HyperFlow [63], is an event-based control plane, which is logically centralized but physically distributed. HyperFlow supports OpenFlow but it requires slight modification on existing control applications. It provides scalability by sharing consistent network-wide view among the SDN controllers. HyperFlow uses publish/subscribe messages to replicate SDN controller events to other SDN controllers. To facilitate failure detection at the control plane, each SDN controller periodically advertises itself through the control channel.

In [36], an integrated SDN principle is proposed. In this approach, a centralized SDN controller controls data traffic forwarding and OLSR (Optimized Link State Routing) is used to route OpenFlow control traffic between the SDN controller and forwarding devices. A module named *OLSR-to-OpenFlow (O2O)* is presented to configure control rules that are used to forward OpenFlow control packets. The O2O module configures control rules by inspecting the IP routing table handled by the OLSR daemon. When a packet is needed to be destined to the SDN controller, the control rule(s) is/are used to forward the packet to the SDN controller and the SDN controller sets routing logic, which is installed into the flow table of the forwarding nodes. Moreover, the O2O module periodically controls the liveliness of the SDN controller. During the SDN controller failure, the O2O module deletes

all the rules from the flow table of the forwarding nodes and dumps all the OLSR routing table, i.e., as long as the SDN controller failure happens, the OLSR takes the control of packet routing. This approach requires translation of complex routing tables into flow rules.

3.2 Work Related to Hybrid SDN

In order to leverage the advantages of the centralized network in today's traditional networks, one approach is deployment of SDN-enabled devices besides the legacy devices to form hybrid SDNs. The other approach is installing a hardware module into legacy forwarding nodes. In the following sections, existing hybrid SDN approaches are illustrated that are related to the second approach to form hybrid SDN.

Reference [28] describes a hybrid SDN architecture for Wireless Distributed Network (WDN) where link-state information is processed by the SDN controller in a centralized manner. For instance, based on link-state information, the weight of each link is calculated by the SDN controller and then the SDN controller broadcasts the calculated weight to the forwarding nodes. Based on the information received from the SDN controller, each forwarding node maintains their routing table that specifies the route with minimum total link cost to go from one node to any other node in the network. The goal of the approach is to distribute computational complexity between the forwarding node and the SDN controller where forwarding nodes determine path in a distributed manner and the SDN controller provides information for decision making.

The approach in [59], presents hybrid IP/SDN network, which is a coexistence of both regular IP forwarding and SDN forwarding. A hybrid IP/SDN node combines an SDN capable switch (SCS), an IP forwarding engine and an IP routing daemon. In that way coexistence of distributed operation for regular IP traffic and centralized operation for SDN-based path (SBP) are maintained. Based on the requirements, different applications can get services from the centralized or the distributed operation.

3.2.1 Controller Platforms for Hybrid SDN

The control plane is a key part of the centralized networking context. The control plane is responsible for computing path for providing end to end connectivity. In hybrid SDN, the controller also plays an important role in controlling both legacy and SDN-capable devices. In the following sections, the existing controller platforms for hybrid SDN are explained.

The work [40] presents a hybrid SDN model that contains both programmable and legacy network nodes in the same network to retain some advantages of an SDN network for path selection and network performance. A network controller named *Telekinesis* is presented in their work, which provides control over both SDN and legacy paths using OpenFlow. This approach requires modification in legacy forwarding nodes in order to offer SDN functionality in legacy nodes. By modifying the legacy nodes, remote manipulation of forwarding entries on a legacy node can be done using OpenFlow, the *Telekinesis*, which acts as a hybrid network controller, can then improve path diversity by providing control over the legacy nodes.

In order to control and manage the hybrid SDN, a management framework named HybNET is presented in [47]. In this approach, SDN-capable forwarding nodes are managed by the SDN controller for carrying out network control and management operations. On the other hand, the legacy devices are used for forwarding, and they are controlled by HybNET. HybNET provides a common configuration interface for both SDN-capable switch and legacy switch by translating legacy network configuration into OpenFlow configuration. In this approach, for managing legacy devices, network virtualization is achieved through VLANs. The HybNET framework translates the VLAN configuration into OpenFlow rule. In the HybNET framework, the legacy network configuration remain hidden from the SDN controller, that in turn converts the global view of the network to an SDN view (i.e., the SDN controller only view the SDN-capable switch in the topology). On the other hand, the global view of the topology (includes legacy switch and SDN-capable switch) is maintained by the HybNET.

In the work [35], SYMPHONY integrates both legacy and centralized control planes in legacy devices. In this framework, a distributed routing protocol and a Legacy

Router Server (LRS) are used to maintain communication between forwarding devices and the SDN controller. In this architecture, LRS maintains the information related to network connectivity, topology information, for instance, and also provides seamless connectivity between centralized networks and legacy networks. On the other hand, the SDN controller contains some modules for path computation, ARP (Address Resolution Protocol) handling, etc.

In [48], the SDN Hybrid Embedded Architecture (SHEAR) presents a hybrid network architecture where a small number of SDN-enabled forwarding devices co-exists with legacy nodes in the same network. By minimizing the number of SDN-enabled switches in the networks, SHEAR simplifies traffic engineering and also provides a loop-free network. The legacy control plane learns about topological information while the SDN-enabled nodes are used as "observability points" that gather the data plane information and acknowledge the SHEAR controller about relevant data plane events. This approach leverages a legacy distributed protocol to detect impairments in the data plane and outsources the recovery of the failure to the SHEAR controller.

3.3 Remarks

This chapter presents a comprehensive study on reliability issues in SDN by categorizing them into three fault domains, i.e., data plane fault, control channel fault and SDN controller fault. The study shows that very little attention has been given to the control plane failure by the research community. There are still some reliability issues like control channel failure or even degraded control channel in wireless context that need to be improved. By addressing limitation of existing approaches, this dissertation investigates reliability of SDN in wireless networks and proposes a solution to overcome SDN during failure. The next chapter describes limitations of the existing approaches. Based on limitations of the exiting approaches, a research question is stated in the next chapter. It also includes validity of the research question that is tackled in the dissertation.

Chapter 4

Problem Statement

This chapter illustrates the research question that has been tackled in this work and also the validity of the research question. Section 4.1 states the specific problem statement that is answered in this dissertation. Section 4.2 demonstrates the limitations of existing approaches, which are described in chapter 3. This section also describes the novelty of our research work that has been carried out to tackle the problem statement. Finally, section 4.3 discusses about how worthwhile is the tackled question in the dissertation from a technical point of view and the contribution of the work to the research community.

4.1 Research Question

As new radio technologies are explored in higher frequency ranges, efficient and dense deployments of SCs become increasingly relevant as they improve the capacity of mobile networks through spatial reuse of the radio spectrum. However, and as proven in current deployments, laying fiber to each base station or SC may not be cost-effective, hence the need to explore wireless backhauls/midhauls, or even, fronthauls. This is particularly relevant in architectures such as Open Radio Access Network (O-RAN) [8]. In this context, wireless mesh networking may offer such a cost-effective solution. On the other hand, to ripe the benefits of such wireless transport networks, proper control and management is key. The adaptation of SDN

in such networks may bring better resource allocation as well as better control by maintaining a clear view of the whole network. But SDN itself has some limitations in terms of reliability in a wireless networking context.

By addressing reliability issues of SDN in wireless contexts, the main goal of the dissertation is to answer the following research question

- Can ***hybrid SDN***, where nodes have certain ***autonomy*** to take control plane switching decisions, improve the ***reliability*** showed by ***centralized*** SDN models in terms of ***control and management*** to manage ***all-wireless multi-hop mesh networks***?

In what follows, we provide a definition of the most relevant aforementioned concepts when defining the research question of this thesis.

Hybrid SDN: In this dissertation, the term hybrid SDN refers to the network where centralized and distributed operation can both be supported by a network node. By retaining logically centralized control over the network, SDN provides simple network management and better resource allocation and improved network flexibility. On the other hand, distributed network operation provides robustness to network failure by quickly reacting to failures of the centralized operation.

Autonomy: We want to find the most appropriate trade-off between centralization and distribution of control and management decisions depending on the network conditions (traffic load, wireless channel impairments etc.). A centralized decision is useful because it maintains a global view of the network based on which decisions can be taken that, in turn, offer better network configuration and management. But gathering and processing this information takes some time. On the other hand, decentralized decisions are faster to take (e.g., fast recovery), but they may result in less optimal configuration and management. Our goal is to inspect control plane reliability from the point of view of the node that experiences the impairments, and so, the node will autonomously decide whether to operate in a centralized or distributed way.

Reliability: In this dissertation, the word reliability refers to protection against any network impairment to provide non-stop packet forwarding/routing. In wireless transport networks, SDN is a key player to manage and control the transport

networks in a centralized way. On the other hand, wireless channels are more likely to be affected by the interference and the environmental condition, which in turn makes wireless channels to experience more link impairments. Therefore, to keep the network alive, reliability of the SDN-based wireless transport networks must be improved.

Control and management channel: In SDN, control and management channels are used to control and manage the forwarding devices in a centralized manner. The difference between control and management is often small, but at a high-level, control would be related to path setup and modification (i.e., what mostly OpenFlow does) and management would be related to setup up the network, such as IP addresses configuration and node configuration in general (e.g., transmission power, a channel for transmission) before establishing paths through the control. This is what OF-CONFIG [9], NETCONF [5], and RESTCONF [18] do.

The control and management channels could be either out-of-band or in-band. Out-of-band channels are separated from data traffic channels. On the other hand, in-band channels share the same links and network environment in general with the data plane traffic, this may also have implications. So, we focus on this problem because it is of fundamental importance for the correct operation of the networks.

All-wireless multi-hop mesh networks: Wireless multi-hop mesh network is a cost-effective solution for the mobile transport networks to provide connectivity between SCs and to connect SCs through a gateway (which may be co-located with the macro-cell, for instance) to a certain aggregation point. The aggregation point connects the SCs to the core network with a desired QoS (Quality of Service) level (see Figure 2.7). Deployment of SCs within the coverage of a certain macro-cell provides higher capacity by reusing radio spectrum and also provides better coverage in that specific area.

4.2 Validity of The Question

Since the first instantiation of SDN architecture done in data centers and campus networks (for research experimentation), most of the works were done initially focused on wired networks and making the network work under this new paradigm. In

this context, it is in general assumed that there is a really-low (almost zero) latency with really high capacity (almost infinite, in practical terms) connections between the SDN controller and the forwarding devices, and so, the SDN controller has a perfect view of what is happening at the dumb forwarding plane devices.

However, as this paradigm becomes mainstream and moves to other environments, this initial assumption must be relaxed. In our case, we consider SDN for an in-band solution in multi-hop wireless mesh networks. We focus on the in-band solution, as in multi-hop in-band solution, communication channels jeopardize the control packets in wireless transport networks because of dynamicity of the wireless environment. Therefore, a new solution is needed in terms of reliability of control/management communication channels in wireless contexts.

The state-of-the-art of the thesis emphasizes on reliability in SDN where most of the focus is on data-plane reliability. However, separation of the control plane from the data plane raises another point of failure (e.g., SDN controller failure or control/-management channel failure or even degraded channel) in the networks, which must be considered. But the reliability of the control plane, especially control channel failure studied much less. On the other hand, the performance of SDN during degraded channels has not been studied much to the best of our knowledge. Moreover, wireless network technologies are of great interest nowadays due to cost-effectiveness in certain deployments. Furthermore, the increasing relevance of non-terrestrial networks (NTNs) [58] may also bring additional value to this work. In this context, wireless links are highly susceptible to interference and environmental conditions that cause wireless link impairments very often. Therefore, reliability of the control plane in wireless SDN is a major concern, but it has rarely been studied. Furthermore, wireless networks offer a cost-effective transport solution for mobile networks where SDN is a key player for controlling and managing it. Therefore, a more comprehensive study on SDN-based wireless transport networks as well as their reliability in an SDN context is required.

Moreover, the previous works only focus on one type of control plane failure at a time. These approaches can not handle controller fault or control channels fault or even degraded channel at the same time. Our scheme provides a solution for all types of control plane failure simultaneously.

4.3 Is It a Worthwhile Question?

Future beyond 5G/6G deployments will be much denser as new much higher frequencies are explored, with deployments of SCs everywhere for offering the required capacity (particularly in urban environments, but also in other less accessible ones, e.g., rural). To reach the capillarity required, the wireless transport networks will play a key role closer to the edge because wireless backhauling/midhauling/fronthauling may be more cost-effective than the fiber-based transport due to the unavailability of fiber everywhere. This acquires more importance in scenarios related with underserved regions and/or NTNs.

In addition, we would like to exploit the programmability that new networking paradigms, like SDN, bring. Therefore, it is a key factor to guarantee that the control and the management information reliably reaches the relevant network entities at the right moment, or alternatively, if this is not possible, the decision may be taken in a distributed fashion. Otherwise, the vision of such dense network will not be realized. Therefore, solving the research question we are proposing in the right way is fundamental for reaping all the benefits that SDN brings in the wireless networking field.

Chapter 5

Hybrid Control Plane Architecture

In this chapter we first illustrate our proposed architecture to form the DenseNet-hybrid SDN infrastructure that adopts wireless mesh networks to provide transport connectivity between SCs. Second, we describe hybrid node architecture where both centralized and distributed operations coexist in the same node and an integrated local agent based on a monitoring framework that periodically monitors the channels for occurrence of any impairment. Finally, we describe the interaction between components inside the forwarding nodes during centralized and distributed operations.

5.1 DenseNet-Hybrid SDN infrastructure

The idea of SCs is to complement macro-cells by adding more base stations with smaller coverage, but higher end-user rates. The macro-cell site usually acts as a gateway site for aggregating the traffic of a set of SCs. The transport network connects SCs to the aggregation point and also to the core network.

Due to capacity concerns of mobile networks, fiber-based technology can be a good option for the mobile transport networks. But due to implementation cost (e.g., laying fiber to all the SCs) in a high density area or for some specific scenarios may not be very cost effective. On the other hand, a wireless mesh networks can provide

a cost-effective solution. SC deployments can greatly increase capacity by reusing radio spectrum. In this context, SDN, by maintaining a global view of the network can provide huge advantages for managing the transport networks and can provide better resource allocation. Even though SDN provides better manageability of the network, in wireless contexts where control messages are sent over in-band channels, performance of SDN may degrade due to channel impairments, which may in turn generate losses of control messages. As Figure 5.1 illustrates, we propose a hybrid control plane architecture for DenseNet. In this way, our architecture attempts to preserve the benefits of both worlds (i.e., centralized and distributed control). Specifically, we propose to maintain a centralized control logic to preserve the benefits of canonical SDN (i.e., simple network management, programmability) under reliable control plane conditions, whereas the distributed control plane is in charge of acting under unreliable conditions to quickly react to failures that avoid the inefficient use of a centralized control logic. Our hybrid control plane architecture proposes changes to the architecture of data plane nodes while preserving the architecture of centralized controllers with respect to canonical SDN models.

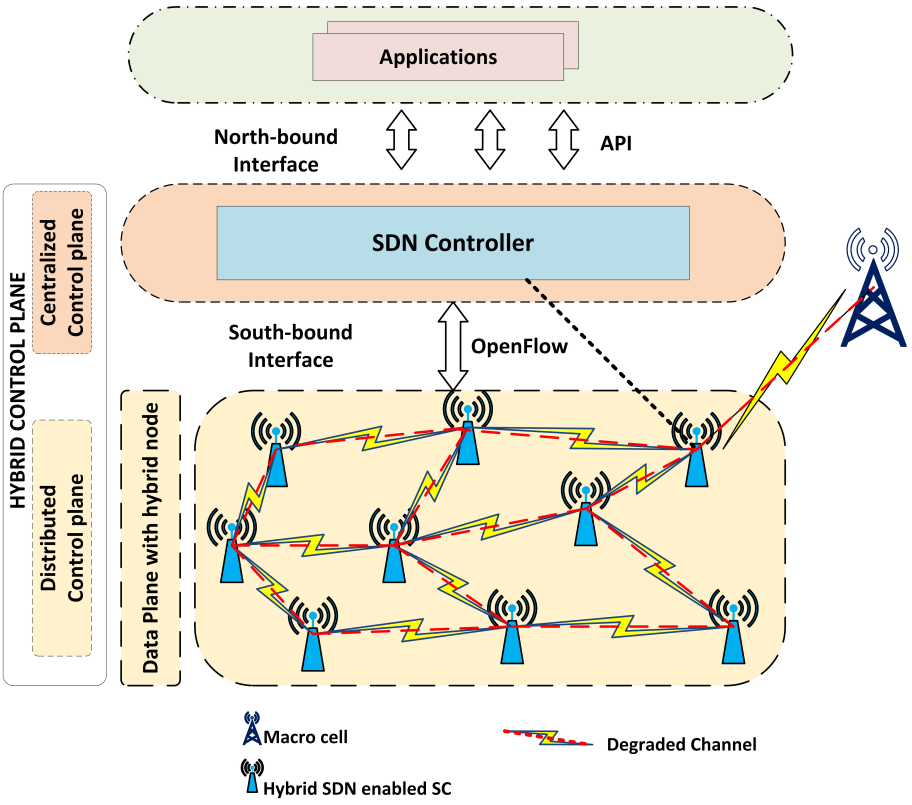

FIGURE 5.1: Hybrid control plane architecture for DenseNet

5.2 Hybrid Node Architecture

In-band SDN solution simplifies deployment of SDN as it does not require a separate network. But it introduces new challenges while deployed in multi-hop wireless networks as condition of the wireless channel degrades significantly by the environmental conditions. In such a case, control packets that are exchanged between the SDN controller and the forwarding nodes experience losses due to unreliable channels. To overcome canonical SDN from failure during unreliable conditions, we propose a hybrid control plane architecture where we change the architecture of forwarding nodes that combines both centralized and distributed operations. We also argue that the forwarding nodes will be responsible for inspecting control communication channel condition and will also switch the mode of operation from centralized to distributed and vice versa depending on the reliability of the control channel. In what flows, we introduce the high-level architecture illustrated in Figure 5.2, which aims to quickly adapt to changes in the control plane by combining both centralized and distributed control logic in the same node, thus, creating a hybrid node as data plane device. The hybrid node is divided amongst the data plane forwarding pipe and a control logic switching algorithm to decide the operation of the data plane forwarding pipe. In what flows, we describe the architecture of the main components embedded in a data plane node. Last but not least, we describe the work-flow between the aforementioned building blocks and the centralized SDN controller. The architectural design of the hybrid node has been presented in [53], [54].

5.2.1 Implementation Details

We have adopted the Open-Source Hybrid IP/SDN networking (OSHI) framework that has been designed in [59, 60, 11]. This framework allows nodes to concurrently run a distributed control plane and a centralized control plane. To attain this goal, the hybrid node embeds an SDN Capable Switch (SCS), such as Open vSwitch, an IP-based forwarding engine (i.e., the one provided by the Linux kernel), and an IP routing daemon based on Quagga to calculate distributed routes (see Figure 5.2). The SCS is connected to the physical network via the physical interfaces while the IP forwarding engine is connected to the SCS via a set of internal virtual ports

endowed in the SCS. In this way, the hybrid node features two types of forwarding, namely IP forwarding (for distributed operation) and SDN forwarding (for centralized operation).

5.2.1.1 Interaction Between Physical and Virtual Interfaces

Interaction between physical and virtual interfaces of the SCS is vital during the presence of distributed operation. While distributed operation is active in the node, the incoming packets (on the physical interface) need to be traversed through the virtual port to the IP forwarding engine for policy making by the IP routing daemon. After accomplishment of policy making, the packet is again sent back to the physical

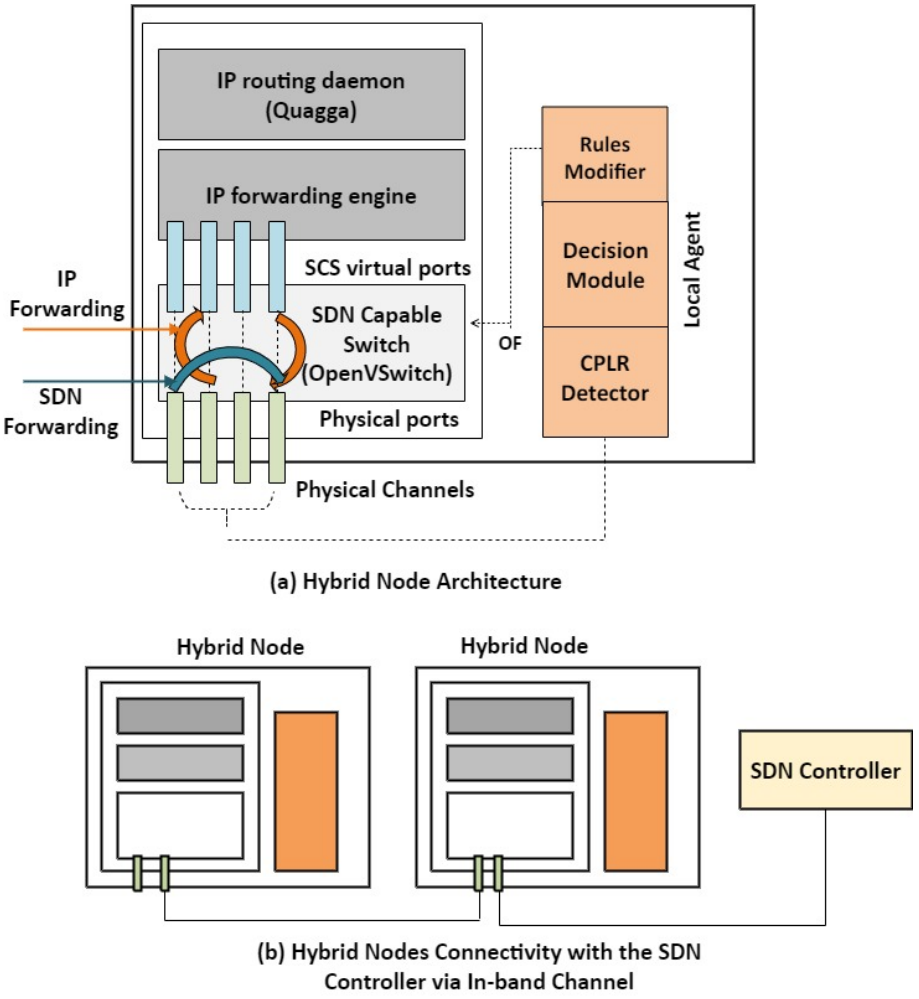

FIGURE 5.2: Hybrid Node Architecture and Connectivity

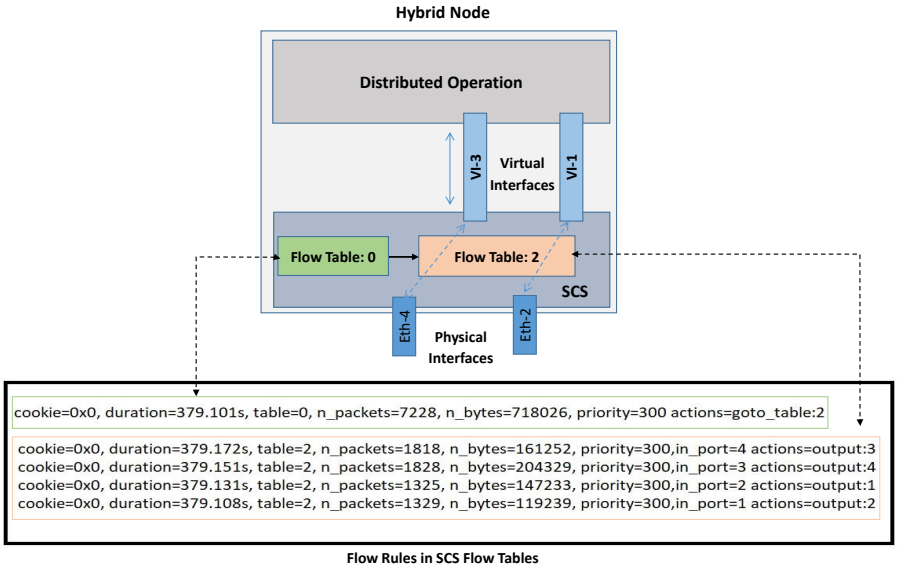

FIGURE 5.3: Flow rules that interconnect physical-virtual interfaces

interface through the virtual interface of the SCS (see Figure 5.3). In that direction, Figure 5.3 describes the rules that bridge between physical interfaces to virtual interfaces and vice versa. During distributed operation, while a packet is received by the SCS's physical interface, the packet is directed to the flow table-2 by following the flow rules in the flow table-0 (see the first rule in Figure 5.3). Here, we have leveraged the benefits of multiple flow tables functionality of Open vSwitch, which is described in the section 2.1.2.3. In our configuration, the flow table-2 of the SCS contains the rules to maintain interaction between physical and virtual interfaces. During initial configuration of SCS, which is an Open vSwitch, these rules are fetched into the SCS flow table. By following the rules in the flow table-2, packets are sent from physical interface to the IP forwarding engine via virtual port and vice versa. For instance, the second rule shows in the Figure 5.3 forwards packets that are received on physical interface number 4 (i.e., $in\_port = 4$) to IP forwarding engine via virtual interface number 3 (which is *action=output:3*). The IP plane then takes necessary routing decisions and using the third rule that shows in the Figure 5.3, the IP plane forwards the packets to the corresponding physical interface of the SCS via the virtual interface. Then the packets are forwarded to the intended destination via the physical interface.

5.2.1.2 Traffic Classification: IP Traffic or SBP Traffic

The flow entry of the main flow table (i.e., table-0) of SCS distinguishes traffic that needs regular IP forwarding or SDN-based Paths (SBPs). VLAN (Virtual LAN) IDs are used to distinguish between packets that need to be processed by a distributed control plane (i.e., IP forwarding) and packets that have to be processed by SDN controller (i.e., SBPs). In our case, by default, packets come to the ingress port of SCS with a tagged VLAN ID. The flow entry embedded into the SCS flow table-0 contains rules to take the necessary steps to forward the incoming packets with VLAN ID.

```
cookie=0x0, duration=133.161s, table=0, n_packets=0, n_bytes=0, priority=302,dl_vlan=10
actions=goto_table:1

cookie=0x0, duration=133.160s, table=1, n_packets=0, n_bytes=0, priority=302,dl_vlan=10
actions=CONTROLLER:65535
```

FIGURE 5.4: Flow rules that distinguish packets for SBPs

Figure 5.4 describes, the flow entry rules for the incoming packets with a VLAN ID. After bootstrapping of the nodes, during initial communication between the SDN controller and the nodes, the SDN controller installs a rule into nodes' flow table (i.e., table-0) to handle incoming packets with a VLAN ID. The SDN controller also installs a *flow-miss* entry rule into SCS flow table-1. The *flow-miss* entry rule is used by the SCS to forward the incoming packets to the SDN controller when no matching rule can be found for the incoming packets in SCS flow table.

When a packet comes to an ingress port of the SCS with a VLAN ID, the packet is matched against the flow entry rule of the flow table-0 with priority 302 (see Figure 5.4). Then, by following the OpenFlow pipeline processing the packet is directed to the another flow table (i.e., table-1). In the OpenFlow pipeline processing, the flow entry can only direct packets to the flow table with higher flow table number. If the corresponding flow table does not contain any flow entry for further redirection of packets to another flow table, then the OpenFlow pipeline processing stops at that flow table. The incoming packets are then processed according to the action associated in the flow entry of that flow table. From Figure 5.4, it can be noticed that while a packet is directed to the flow table-1, if there is no matching rule for the

packet, the incoming packet is then forwarded to the SDN controller using *flow-miss* entry rule (e.g., using instruction, action=CONTROLLER:65535). Then, the SDN controller installs rules to forward the packet to the intended destination. In this way, the SDN operation is preserved and SBP is maintained.

In order to process the incoming packets according to the distributed operation, the VLAN tag is removed. While a packet comes to the ingress port of the SCS, the packet is matched against the flow entry rule embedded into the flow table-0 with priority 301. The flow entry rule then removes the VLAN ID from the incoming packets by following instruction (*action=pop_vlan*) set in the rule. Then, the packets are directed to the flow table-2 by following OpenFlow pipeline processing. The flow entry rules integrated into the flow table-2 forward the packets to the IP forwarding engine for distributed policy making (for details see section 5.2.1.1).

```
cookie=0x1, duration=137.839s, table=0, n_packets=66, n_bytes=6472, priority=301,ip,in_port=3,dl_vlan=10
actions=pop_vlan,goto_table:2

cookie=0x0, duration=535.160s, table=2, n_packets=204, n_bytes=15816, priority=300,in_port=4 actions=output:3
cookie=0x0, duration=535.150s, table=2, n_packets=75, n_bytes=7174, priority=300,in_port=3 actions=output:4
cookie=0x0, duration=535.140s, table=2, n_packets=721, n_bytes=61380, priority=300,in_port=2 actions=output:1
cookie=0x0, duration=535.131s, table=2, n_packets=723, n_bytes=77960, priority=300,in_port=1 actions=output:2
```

FIGURE 5.5: Flow rules to enable IP forwarding

Figure 5.6 describes the flow chart of the processing of packets in the SCS flow table depending on the network operation mode (i.e., distributed or centralized). In the case of centralized mode of operation, the VLAN ID is not removed from the incoming packets and the packets are directed to the flow table-1. If there is a matching rule for the packet, the corresponding action is taken against the packet. Otherwise, the packet is forwarded to the SDN controller for policy making. On the other hand, while distributed operation prevails in the network and is responsible for distributed policy making, the VLAN ID is removed from the packets and the packets are directed to the flow table-2. The actions set that are integrated into the flow table-2 are then executed to perform IP routing.

5.2.1.3 Integrated Local Agent

In our work, the forwarding node is responsible to inspect impairments, i.e., control plane failure, link failure or even degraded communication channel between forwarding nodes and the SDN controller. For this reason, we included a local agent in the data plane node in order to enable centralized or distributed control depending on the reliability of the control communication channel between data plane devices and the centralized SDN controller. The local agent is composed of *i) CPLR detector, ii) decision module, and iii) rules modifier*. The module *CPLR detector* is based on a monitoring framework that continuously infers the reliability of the control plane by periodically monitoring the status of the control communication channel. In this work, the metric is based on determining the packet loss ratio of the control communication channel, i.e., node-to-controller communication channel. The resulting metric referred to as *Control Packet Loss Ratio (CPLR)* (as well as the slope of the $CPLR$ vs. t curve) that determines the status of the control communication channel between the data plane node and the SDN controller. To calculate CPLR of a link,

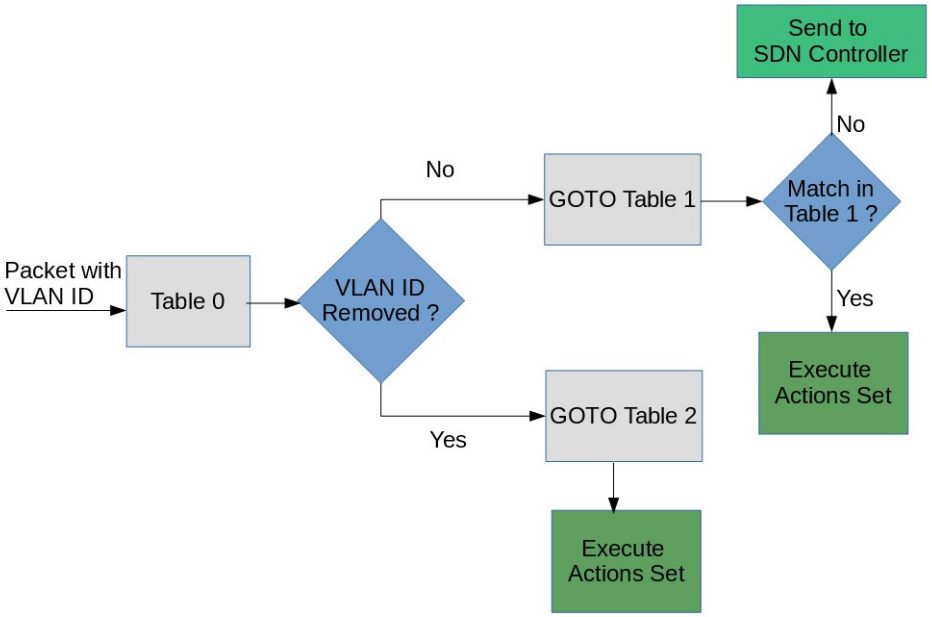

FIGURE 5.6: Packet processing in the SCS flow table

we first measure the number of sent and dropped control packets on each link of a node by using Traffic Control (TC) [21] tool, which is a Linux utility. The *CPLR detector* contains a small piece of software that calculates the percentage of loss of packets of each link, which in turn gives us CPLR value of each link. Depending on the reliability of the control communication channel, which is determined by the *CPLR* value of the links, as well as the CPLR trend, the decision module integrated into the data plane nodes (see Figure 5.2) decides about the activation of the distributed operation from the centralized operation and vice versa. In this sense, it embeds a control logic switching algorithm (see chapter 6) that selects the mode of operation of the network nodes by detecting trends that describe the quality of the control communication channels. The prediction is based on the various measurements gathered from the centralized or the distributed control plane logic, which is active in a given data plane node. Under a high-loss regime, the decision module triggers the action to perform network switching from centralized to distributed control plane operation. Then, the *rules modifier* of the local agent pushes some predefined rules to activate the distributed operation in the node that is decided by the decision module. When the network switching operation happened because of the failure of the centralized operation, the module *CPLR detector* of the local agent keeps monitoring the control communication channel as well as the control plane. While the control plane as well as control communication channel performance improves and the *CPLR detector* of the local agent characterizes the control channel as a reliable-enough medium by measuring the *CPLR* of the channel, the decision module again triggers the action to switch back network operation to centralized mode.

5.3 Data Plane Node Forwarding Pipe and Network Operation

This section describes how the data plane forwarding pipe works in a node during presence of centralized or distributed network operation.

5.3.1 Interaction Between The Data Plane Node Forwarding Pipe, The Local Agent, and The Centralized SDN Controller

In our setup, by default packets are tagged with a VLAN ID to enable the use of a centralized control plane logic embedded in the SDN controller (see Figure 5.7). In the following, we summarize the centralized operation.

1. When a packet with VLAN ID is received by a physical interface of the SCS, the SCS conducts a lookup in its flow tables to find a match for the current packet.

2. If a match is found in one of the flow tables, the packet is then forwarded according to the rule installed previously by the SDN controller into that flow table. Otherwise, the incoming packet is forwarded to the centralized SDN controller for defining policies for appropriate handling of the incoming packets.

3. The centralized SDN controller installs the necessary OpenFlow rules to serve the current incoming packet. Thus, the forwarding data plane node simply follows the instructions set by the SDN controller to forward a packet.

4. The *CPLR detector* module of the local agent periodically monitors the control communication channel as well as the control plane by inspecting CPLR.

5.3.2 Interaction Between The Data Plane Node Forwarding Pipe, The Local Agent, and The IP Forwarding Engine

In our setup, when the distributed operation is prevailed in the network, the VLAN ID is removed from the incoming packets to interact the data plane forwarding pipe to the distributed operation (see Figure 5.8). In the following, we summarize the data plane forwarding pipe under distributed mode of operation.

1. While the distributed operation is active in the given data plane node and a packet with VLAN ID is received by a physical interface of the node, the VLAN ID header is removed from the packet.

2. The packet is then forwarded to the IP forwarding engine via virtual interface for policy making in distributed fashion. The virtual interface acts as a bridge between the SCS and the IP forwarding engine. In the IP forwarding engine, the distributed protocol (OSPF) is integrated for policy making.

3. When the policy making for the packet is done by the distributed protocol, the packet is then again forwarded to SCS via virtual interface.

4. The SCS again encapsulates the VLAN ID before the packet leaves the SCS egress port.

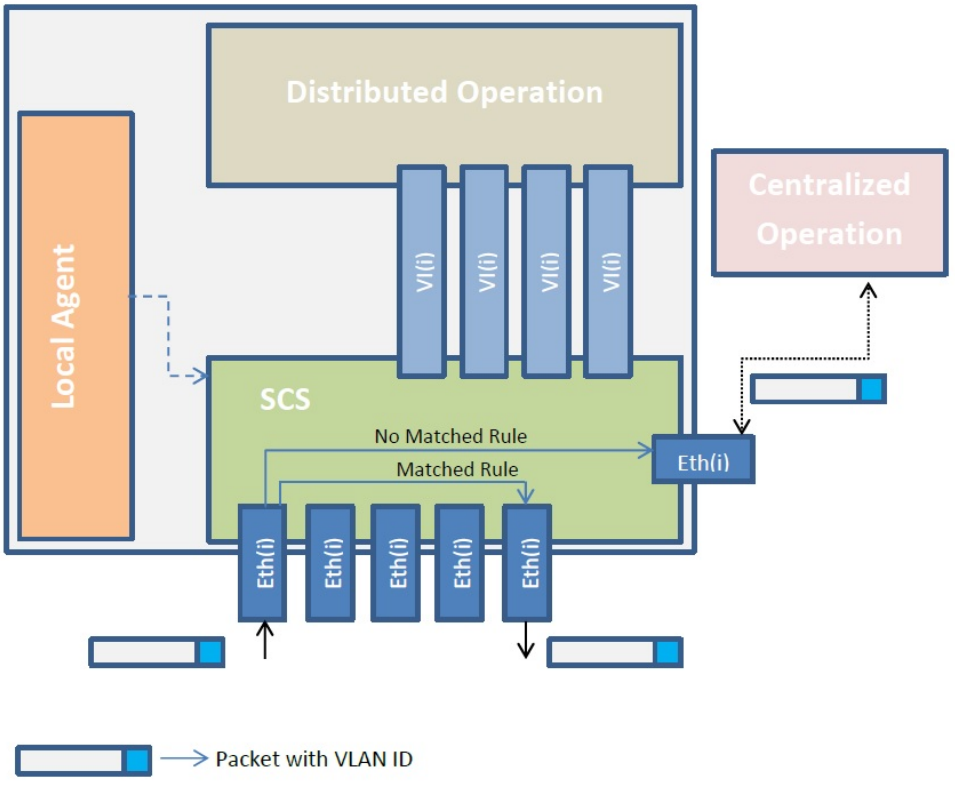

FIGURE 5.7: Interaction-Data plane forwarding pipe and centralized operation

5. The *CPLR detector* module of the local agent periodically monitors the control communication channel by inspecting CPLR of the channel.

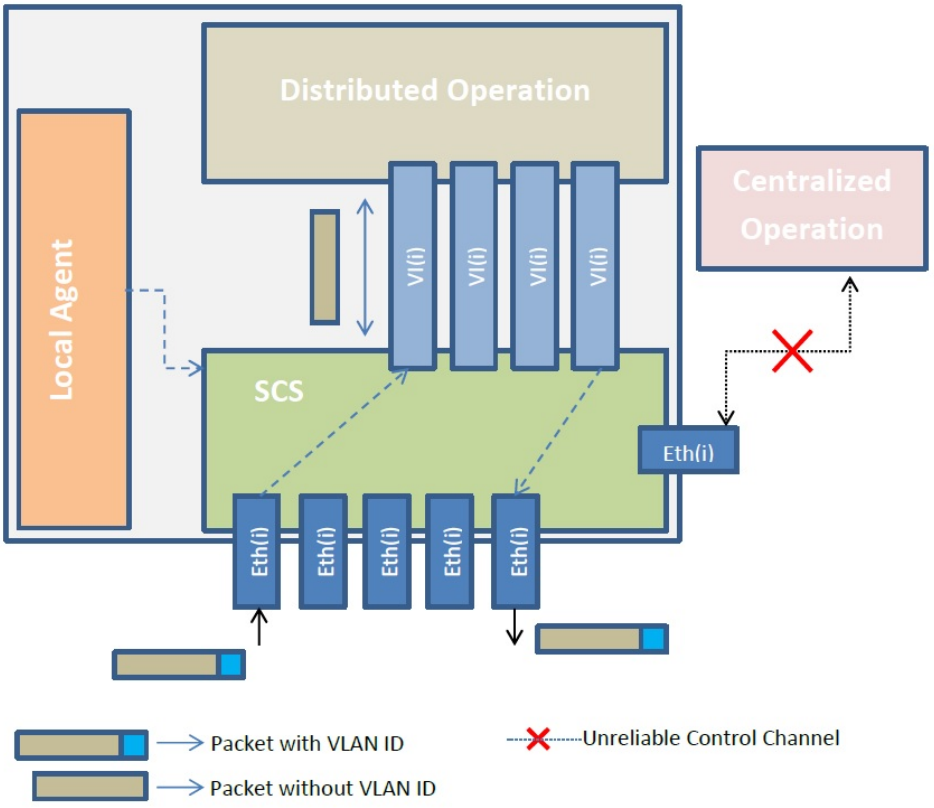

FIGURE 5.8: Interaction-Data plane forwarding pipe and distributed operation

Chapter 6

Network Control Logic Switching Algorithm

This chapter begins with a description of the operation of (centralized) canonical SDN and it also describes how the data plane failure is recovered in this case. Control plane failure has a huge impact on recovering canonical SDN from data plane failure and, at some points, canonical SDN becomes non-operational under control plane failure. Section 6.3 of this chapter describes the conceived novel network switching algorithm that decides on network operation switching between centralized and distributed mode and vice versa in our hybrid control plane architecture during control plane impairments. This chapter also includes all the steps that are followed to switch between network operation modes.

6.1 Canonical SDN Operation

In our canonical SDN setup, which is only centralized, when a new packet comes to the ingress port of a node (e.g., Open vSwitch), the node looks up in its flow tables for a flow entry match for the packet. If no match is found in the flow tables, the node sends the packet as a *PacketIn* message to the SDN controller using the *flow-miss* entry rule. Then, the SDN controller calculates the shortest path for the packet using shortest-path-first algorithm (SPF) (i.e., Dijkstra algorithm) and, using

PacketOut message, the SDN controller instructs the forwarding node to install flow rules into the flow table of the nodes. The forwarding nodes keep the rules for an infinite period unless the nodes get further instruction from the SDN controller to modify or delete the rules. With the subsequent flow of packets, the forwarding nodes simply forward the packet according to the rules installed previously with the intervention of the SDN controller.

6.1.1 Canonical SDN Operation with Perfect Control Plane and Imperfect Data Plane

In case of change of network topology (for instance, the link between two nodes goes down), the nodes that get affected inform the SDN controller immediately about the impairment by sending *PortStatus* messages. In our setup, when the SDN controller receives *PortStatus* messages, it instructs, by means of *FlowMod* messages, the affected nodes as well as all the nodes in the affected path to delete all the previous rules from their flow tables except the *flow-miss* entry rule. The *flow-miss* entry rule is not deleted because using this rule the forwarding node sends new packets to the SDN controller when the matching rule can not be found in the forwarding table of the node. When the forwarding nodes in the affected path get instructions from the SDN controller to delete rules, the nodes perform deletion of rules according to the instructions of the SDN controller. As the rules are deleted from the flow table of the nodes, when a packet comes to the ingress port of the nodes, the nodes again send the packet as a *PacketIn* message to the SDN controller using the *flow-miss* entry rule. The SDN controller then again sets a new policy for the incoming packets and instructs all the nodes in the new path to install new rules in their flow table. The nodes then forward subsequent packets according to the new rules.

6.1.2 Canonical SDN Operation with Imperfect Control plane and Imperfect Data Plane

To keep canonical SDN operational in case of data plane impairments, control messages (e.g., *PortStatus*, *FlowMod*, *PacketIn* and *PacketOut* messages) and also topology discovery messages (e.g., LLDP) need to be exchanged between the SDN controller and the forwarding devices regularly. The exchange of control messages require having a low loss, as in canonical SDN scenario the SDN controller is the central point that makes the policies for incoming packets and the forwarding devices follow the instructions set by the SDN controller to forward the packets. During imperfect control plane conditions, loss of control messages due to degraded control channel or failure of control channel, or even, failure of the SDN controller, may hamper the correct operation of canonical SDN, as the control messages can not reach the SDN controller from the forwarding devices and vice versa. As the SDN controller and the forwarding devices maintain TCP connections, lost control messages are retransmitted, which may protect SDN operation from failure in a low-loss regime. However, the high-loss regime imposes several retransmissions of lost packets or even loss of a packet completely after several retransmissions, which substantially degrades the performance of canonical SDN or it even makes it non-operational.

6.2 Distributed Operation

Under distributed network operation, network intelligence (i.e., routing decisions) is distributed among the network nodes (e.g., routers) and tightly bound to each network node. A distributed routing protocol (e.g., OSPF) running in a single area within the same Autonomous System (AS) determines the route to each destination. In our distributed network scenario, OSPF version 2 (OSPFv2) is deployed. OSPF uses link-state information to make routing decisions and the SPF (Shortest Path First) algorithm (Dijkstra algorithm) is used to calculate the shortest path to the destination. Each node in the same area runs the algorithm and maintains an individual topological database. *Hello* messages are exchanged periodically between neighbor nodes to acknowledge their existence. Each node in the area floods Link-State Advertisements (LSAs) that advertise node state (i.e., interface or link state)

and also carries information about its neighbors to adjacent routers in the same area.

6.2.1 Distributed Operation During Topological Change with Perfect Channel Conditions

During topological changes in the network, due to link failure or interface of the router becoming unavailable, the OSPF protocol detects the topological change in the network and calculates the new loop-free route. Each node in the network floods LSAs periodically to advertise its state. In case of any impairments, the affected node floods the LSA immediately and whenever the neighbor nodes of the affected node receive a copy of the LSA, the nodes again flood the copy of the LSA. However, if a node receives a duplicate copy of the LSA, the node simply discards the LSA. By receiving LSAs, the other nodes in the area update their topological database and the network topology diagram becomes stable again. The OSPF protocol then calculates the new loop-free route to the destination.

6.2.2 Distributed Operation During Topological Change with Imperfect Channel Conditions

In case of imperfect channel conditions while there is a topological change in the network, the affected node floods the LSA, and all the nodes in the area receive a copy of the LSA and update their routing table. Due to imperfect channel conditions, some LSAs may be lost. Lost LSAs are retransmitted by the source node. LSA flooding brings some benefits during the high-loss regime. In a high-loss regime, LSAs may get lost and a node may not receive the LSA from a neighbor node, but because of LSA flooding, it may receive a copy of LSA from other neighbor nodes. In this way, each node in the network knows about the topological change and updates its individual topological database.

6.3 Control Logic Switching Algorithm

During imperfect control plane conditions, the SDN performance degrades (see section - 8.2) due to loss of control messages and at critical conditions (e.g., high-loss regime), the SDN may become non-operational. On the other hand, legacy distributed operation keeps the network operational by flooding LSAs (see section 8.3), which makes the networks stable during topological change in a high-loss regime.

To recover SDN from failure, we have integrated a control logic switching algorithm (also referred to as the *decision module*) into the data plane nodes (see section 5.2) that autonomously takes network switching decisions by predicting network conditions. The network condition is being predicted based on periodical measurements performed by the module *CPLR detector*, which inspects channel conditions by means of the metric named CPLR. The metric CPLR goes as an input to the algorithm and based on the trend of the CPLR metric over time during different states, the algorithm decides which network operation will be active in the network to perform packet forwarding or routing.

As a consequence of any impairment experienced by the control plane, due to either failure of the SDN controller or a degraded control channel, CPLR values will increase, and an anomaly will be inferred by the network node. Algorithm 1 describes the control logic switching algorithm. All the state decisions have been explicitly reflected for the sake of clarity. This algorithm is periodically run in the local agent of each node in the network. Each of these periods is referred to the Algorithm 1 with subindex $k \in \mathbb{N}$. The local agent will call this algorithm by providing some input parameters, such as the *current_state*, the measured CPLR (calculated by the *CPLR detector* module), and the slope of the *CPLR vs. t* curve, calculated in the previous period (S_{k-1}) and the current one ((S_k)). Based on the input parameters the algorithm predicts network conditions of the upcoming periods and decides which operational mode will be active in the network which comes as an output of the algorithm. If the *current_state* (i.e., centralized or distributed) is different from the output of the algorithm, the local agent integrated into each node performs some actions (see section 6.3.1 and 6.3.2) to apply the network operation switching.

Algorithm 1 Pseudo code of the control logic switching algorithm.

Input: *current_state*,$CPLR$,S_{k-1}, S_k
Output: *next_state* ▷ centralized or distributed

```
1:  procedure DECIDE_STATE
2:      if current_state=centralized then
3:          if (CPLR < CPLR_max) then
4:              if S_k ≤ δ then                          ▷ If low CPLR increasing trend
5:                  next_state=centralized
6:              else
7:                  if S_{k-1} > δ then
8:                      next_state=distributed
9:                  else
10:                     next_state=centralized
11:                 end if
12:             end if
13:         else                                         ▷ CPLR ≥ CPLR_max
14:             next_state=distributed
15:         end if
16:     else                                             ▷ current_state=distributed
17:         if S_k >= 0 then
18:             next_state = distributed
19:         else
20:             if CPLR < CPLR_max then
21:                 next_state=centralized
22:             else
23:                 next_state=distributed
24:             end if
25:         end if
26:     end if
27:     return next_state
28: end procedure
```

There are various design criteria behind this algorithm. Since our goal is to stay as much as possible in the centralized mode and only use the distributed mode as a backup operational mode, the algorithm is more conservative (i.e., it attempts to make sure) when switching from centralized to distributed than vice versa (lines #3–15). The rationale behind this is that the application of network-wide management policies is easier when operating in centralized mode due to easier programmability through the SDN controller APIs and functionality.

Furthermore, switching to distributed operation implies certain control message exchange and processing (section 6.3.1). On the other hand, the algorithm will decide to switch to the centralized mode in a greedier manner (i.e., based on slight improvements in network conditions), for the reasons explained above (lines #16 et seq.), as soon as network conditions improve.

To control its operation, there are two main parameters in the algorithm. First, $CPLR_{max}$ that is the CPLR under which TCP retransmissions are enough to guarantee (even if delayed) the interaction between the SDN controller and the node, though there may be an impact in network performance, as observed in chapter 8. Second, the conservative behavior is represented by parameter δ, the value of the slope (S_k) of the curve *CPLR vs. t* during the current period k, if exceeded, implies a noticeable increasing trend of CPLR (line #6).

Therefore, it is based on an increasing trend and not based on instantaneous CPLR values that the algorithm decides to switch to distributed operation. That is, the algorithm at least takes the increasing slope above δ during two consecutive evaluation periods to switch network operation from centralized to distributed(lines #6–7). However, if the instantaneous value is such that CPLR is above $CPLR_{max}$, which means that network performance cannot be guaranteed (even if with high-losses), the switching decision is taken without waiting for evaluating the slope in the following period (line #13).

When in distributed operation (lines #16 et seq.), the aim is to switch back to centralized when an improvement in the network's conditions is detected. To restore centralized operation in the network, this is coded in the algorithm as measuring a non-positive slope in this period (no need to wait for two periods, as above) (line #19), and the CPLR value is below the one that guarantees that the network can operate in centralized mode ($CPLR_{max}$) (line #20). Under other conditions, the algorithm decides that the problem persists and decides to stay in distributed mode.

6.3.1 Network Operation Switching: Centralized to Distributed

In case of high-loss conditions, when the CPLR of the control communication channel is increasing and the decision module detects that the CPLR curve is sloping upwards or the CPLR value reaches $CPLR_{max}$, the switching algorithm integrated into the local agent of the nodes takes the network operation switching decision from centralized to distributed mode. In order to perform network operation switching from centralized to distributed, the following steps are followed by the local agent:

1. The *rules modifier* module deletes the old rules, except the *flow-miss* entry rule, from the SCS flow tables that were installed with the intervention of the SDN controller. The *flow-miss* entry rule is not deleted as this rule is important while network operation is again restored to centralized mode (see section 6.3.2).

2. The *rules modifier* module installs new rules in the SCS flow table, which include an OpenFlow *POP_VLAN* action, aiming to remove the VLAN tag of the incoming packets to the ingress port of a node and to forward the incoming packets to the IP forwarding engine.

3. Incoming packets arriving at the SCS ingress port are then forwarded to the IP forwarding engine via the internal virtual port, in order to process the packets through the IP routing daemon. In the IP routing daemon, a distributed routing protocol (in our case, Open Shortest Path First (OSPF)) makes the policies to route packets to their intended destination and send the packets to the SCS egress port via the internal virtual port (see section 5.3).

4. On the other hand, the *rules modifier* module of the local agent also updates the SCS flow table by adding another rule that pushes the VLAN tag again to the packets using an OpenFlow *PUSH_VLAN* action before the packets leave the SCS egress port and sends the packets toward the following hop toward the destination host. All these modifications (e.g., deletion and installation) of rules in the SCS flow table are performed without the intervention of the SDN controller.

Whenever the network operation in the node is switched to distributed mode, the node acts as a legacy device where a distributed routing protocol (e.g., OSPF) ensures the policy making and routing of incoming packets. The SDN controller, in that case, does not have any influence over the nodes in the network because of control packet loss due to impairment in the control communication channel.

6.3.2 Network Operation Switching: Distributed to Centralized

During the distributed mode of operation, the module *CPLR detector* of the local agent continues measuring the CPLR (and its slope) of the control communication channel, and when the CPLR goes below $CPLR_{max}$ (and there is a non-increasing CPLR trend), the switching algorithm takes the network operation switching decision from distributed to centralized again. The local agent then restores centralized operation back by following these steps:

1. The *rules modifier* module deletes the rules from the SCS flow table that includes *POP_VLAN* and *PUSH_VLAN* OpenFlow actions that were installed by the local agent during network operation switching from centralized to distributed. When the control plane performance becomes fair again, the SDN controller starts receiving LLDP (Link Layer Discovery Protocol) messages and, by receiving LLDP messages, the SDN controller discovers the network topology again.

2. As during switching from centralized to distributed, all the flow rules installed by the SDN controller were deleted, except the *flow-miss* entry rule, when the SCS ingress port receives a packet, it sends the packet to the SDN controller for policy making using the *flow-miss* entry rule, as there is no matching rule remaining in the SCS flow table. In this way, network operation switches back again to centralized mode and the SDN controller takes control of the network.

Chapter 7

Framework for Experimental Evaluation

This chapter discusses about the considerations behind selecting the tools we selected for the framework used to assess the research statement explained in the chapter 4. The selection of a simulator or an emulator to evaluate the network topology is one of the main components to take into account together with the selection of the SDN controller platform. This chapter begins by describing the reasons behind choosing the emulator Mininet and then illustrates the architectural overview of the emulator. This chapter also explains the architecture of the SDN controller platform that has been used in the research work.

7.1 Network Simulator and Emulator

A network simulator or emulator provides flexibility to evaluate network performances efficiently without having real deployment. It provides an environment to deploy that behaves like an actual network, which eases carrying out realistic evaluations on top of it. The following paragraphs briefly describe the simulator and the emulator.

- Simulator: A simulator is a software program that creates an environment to mimic the behavior and configurations of a real device. Unlike doing experiments on real devices, executing simulation on the simulator to do experiments is flexible, cost-effective, more controllable and scalable. A simulator has its own simulation clock to precisely control the execution of the simulated components and the simulation clock may be faster or slower than real-time [67]. For this reason, several executions of the same experiment as well as the same model, generate the same results. However, the results generated by the simulator deviate from the results generated by the real components if modeling of the real device is not accurate.

- Emulator: An emulator duplicates the hardware and software features of a real component. An emulator utilizes the virtualization technique to emulate components that run real operating systems and application programs [69]. An emulator does not have its own clock to control the execution order of emulated components, rather it uses a real-time clock. All the components running on the emulator share the same kernel and the kernel's CPU (Central Processing Unit) scheduler schedules the execution order of emulated components. Due to uncontrollable events (e.g., the processes running on the system), results generated by an emulator for several executions of the same experiments may differ. However, results generated by the emulator are almost the same as the results generated by the real components.

7.1.1 Simulator and Emulator for SDN

A very relevant tool used by the network research community is Network Simulator 3 (ns-3) [39]. NS-3 supports OpenFlow but it has some limitations to deploy SDN scenarios. EstiNet [2] also supports OpenFlow, and it can be used in both simulator and emulator modes. But the most popular tool used by the SDN research community is Mininet [46], [4]. The following Table 7.1 compares the functionalities and compatibility of the tools to conduct experiments on SDN.

TABLE 7.1: A comparison of SDN tools

	Mininet	**EstiNet**	**ns-3**

Operational Mode	Emulator	Simulator and Emulator	Simulator
OpenFlow support	OpenFlow 1.3	OpenFlow 1.3	OpenFlow 1.3 (using the module OFSwitch13 [34])
Working Principle	Mininet uses the virtualization technique to emulate hosts and uses Open vSwitch to create software switches. It uses the virtual Ethernet pair approach to connect switches and also hosts.	EstiNet utilizes kernel re-entering [68] procedure that allows multiple hosts to run on a single kernel. The EstiNet simulation engine process allows simulating multiple Open vSwitches. Exchange of packets between application programs running on different hosts, go through real TCP/IP layer of the kernel and also go into a tunnel network interface that connects to the simulation engine where each host has its own simulated protocol stack.	In NS-3, the operation of the devices (e.g., Open vSwitch) is simulated by compiling and linking the device´s module (written in C++) with its simulation engine code to form a user-level executable program.

Compatibility with real world SDN controller	The emulated host on Mininet is like a virtual machine that allows running any real applications to exchange information. The virtualization technique provides the flexibility to run the real SDN controller on an emulated host and to control the software switch (e.g., Open vSwitch) by setting up a TCP connection.	The kernel re-entering scheme in EstiNet allows running a real application on a simulated host which in turn gives the flexibility of running the SDN controller in EstiNet without any modification. To create a TCP connection between the switch and the SDN controller, the operation of the switch is simulated inside the simulation engine and let it to create a TCP/IP socket bound to a tunnel interface [69].	In ns-3, a real SDN controller can not readily be run on a node without modification, as ns-3 is a user-level program and a real SDN controller is also a user-level program that can not be compiled and linked together to form a single executable program. As a result, the SDN controller needs to be implemented from scratch as a C++ module.
Openness	Open-Source	Proprietary	Open-Source

In this research work, Mininet has been chosen to deploy the SDN scenario as Mininet is an open-source tool that provides enormous flexibility to create, customize, share and test SDN networks in a very simple way.

7.2 Mininet

Mininet is an open-source network emulator for deploying an entire network on a single PC or on a virtual machine (VM). Mininet provides flexibility to design network topologies and new functionalities by using the high-level programming language Python. Mininet uses a Linux container as an emulated host which is a lightweight VM with individual network interface. The OS-level virtualization features, including processes and network namespaces, provide Mininet with a scalable prototyping environment and also allows managing hundreds of nodes.

In contrast to currently available prototyping environments, Mininet code can easily be deployed on hardware-based networks and testbeds. In that sense, Mininet is an interactive prototyping environment for real networks.

7.2.1 Mininet - Virtual Network Architecture

Four fundamental network elements are available in Mininet to create a topology, which are the following ones:

- Links: Act like a wire that connects virtual interfaces of different nodes to provide connectivity. These are also known as Virtual Ethernet pair or veth pair. Packets that are sent via one interface are delivered to another interface just like a fully functional Ethernet port.
- Host: An emulated host in Mininet is a Linux container, which is simply a shell process moved into its own namespace, from where commands can be executed. Each host has its own virtual Ethernet interface(s). The real application program (e.g., SDN controller) can readily be run on an emulated host.
- Switches: Switches are Linux-based software switches that provide the same packet delivery semantics that would be provided by a hardware switch.
- SDN controller: The SDN controller can be placed on an emulated host within the same emulated network or anywhere in the real network, as long as the

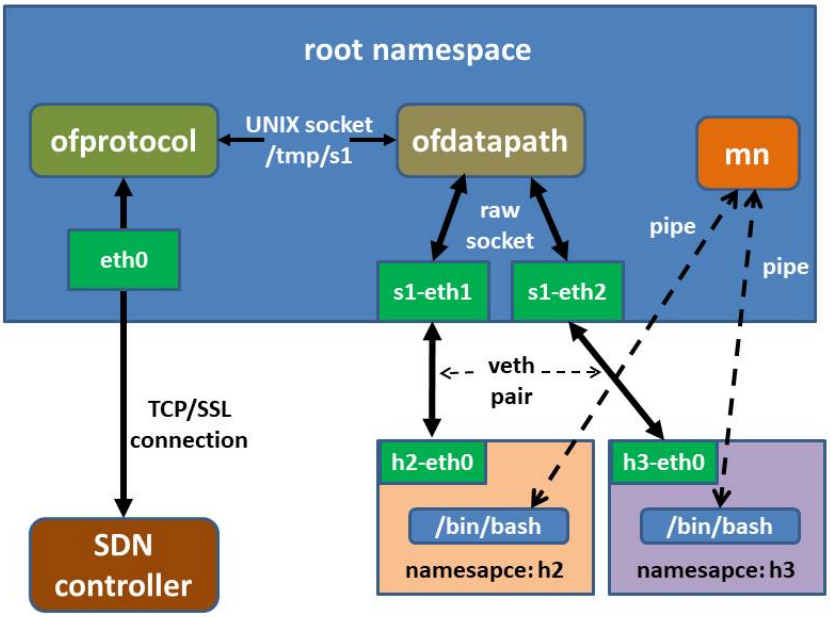

FIGURE 7.1: Mininet Architecture [46]

switches running on the machine have IP-level connectivity with the SDN controller.

The emulation of a simplified virtual network in Mininet has been depicted in Figure 7.1. Figure 7.1 shows that two hosts (labelled as h2 and h3) are connected to a switch (labelled as ofdatapath). The switch communicates with the SDN controller through the OpenFlow protocol using a secure control channel. Hosts are inside separate Linux namespaces with their own virtual Ethernet interfaces that are privileged to run real application programs, like packet sniffer tools (e.g., Wireshark [25]) on each interface to capture packets within the emulated scenario.

7.3 SDN Controller Platform

The SDN controller is a core component in the SDN paradigm, as the SDN controller is the brain of the network where network policies (e.g., routing) are applied in a centralized fashion to manage the underlying infrastructure layer.

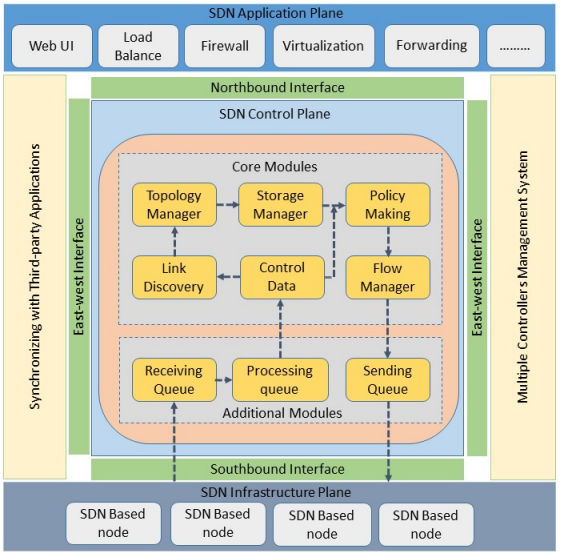

FIGURE 7.2: SDN Controller Architecture

7.3.1 Architecture of SDN Controller

Figure 7.2 depicts a generic architecture of the SDN controller. The modularized design of the SDN controller consists of several modules where each module is responsible to perform a certain functionality and cooperation between modules is maintained to provide complete functionality.

The core of the SDN controller accommodates some basic modules that include the topology discovery module, link discovery module, storage module, policy making module, flow table module and control data module. Furthermore, to provide buffering functionality the core of the SDN controller contains some additional modules like receiving queue, processing queue and sending queue [66], [72]. The core of the SDN controller is surrounded by the interfaces that manage communication between the core of the SDN controller and different layers.

7.3.1.1 Functionality of Core Modules of SDN Controller

As the SDN controller is the brain of the network, it needs to have a total view of the network resources (e.g., nodes and links) in order to control the network and to make strategies. Link discovery, topology discovery and traffic flow handling are the main functionalities of the controller core.

- **Link Discovery Module:** According to OpenFlow technical specifications, the OpenFlow switch should have two initial configurations. Firstly, the IP address and the TCP port of the SDN controller (in case of multiple controllers, IP address and TCP port of the master controller and a pool of IP addresses of the slave controllers) have to be set in the OpenFlow switch. Secondly, a default flow rule has to be pre-installed in the OpenFlow switch to route the LLDP packets to the SDN controller encapsulating in OpenFlow *Packet_In* message [38], [52], [55].

  In the initial stage, the OpenFlow switch searches for the SDN controller in the network and attempts to establish a secure connection with the SDN controller through Transport Layer Security (TLS) protocol to send and receive configuration messages. As a part of the initial handshake, the SDN controller requests the features of the SDN-based forwarding nodes (e.g., Open vSwitch) employing *FEATURE_REQUEST_MESSAGE*. The node responds with *FEATURE_REPLY_MESSAGE*, which contains data-path ID, list of active ports and corresponding MAC address of the ports. After receiving *FEATURE_REPLY_MESSAGE*, the SDN controller knows about the features of the forwarding node.

  To discover links between forwarding nodes in the infrastructure layer, the link discovery module is used by the SDN controller. The link discovery module of the SDN controller periodically sends LLDP messages to the SDN-based nodes (e.g., Open vSwitch) to discover the links between SDN-based switches. The SDN controller uses OpenFlow *Packet_Out* message to send the LLDP packets to the SDN-based node. For instance, while the forwarding node (A) receives the *Packet_Out* message, the node decapsulates the LLDP packets and outputs the packet via every active port except the ingress port. However, the LLDP packet received by node (B) from node (A), is unknown to node (B). Then node (B) encapsulates the LLDP packet in *Packet_In* message and sends it to the controller (see Figure 7.3). On the other hand, the SDN controller also sends LLDP packet to node (B) and by following the same procedures, the SDN controller also gets a *Packet_In* from node (A). After receiving *Packet_In* messages from node (A) and node (B), the link discovery module of the SDN controller learns that node (A) and node (B) are directly connected.

- **Topology Manager:** The topology manager periodically reads link information provided by the link discovery module and creates a topology graph of the whole network by combining all the relevant information of the forwarding nodes. The topology manager creates a topology instance and puts it into the storage manager.
- **Policy Manager:** The policy manager is responsible to make the decision to forward the incoming packets. For instance, the policy manager using the topological information calculates the shortest path based on the shortest path calculation algorithm (e.g., Dijkstra algorithm).
- **Flow Manager:** The flow manager creates the flow rules based on the policy applied by the policy manager and pushes the flow rules to the forwarding nodes by encapsulating them into *Packet_Out* messages.

7.3.1.2 Interfaces

To maintain interaction between different layers, the core of the SDN controller is surrounded by several interfaces which are explained next:

- **Southbound Interface:** The Southbound Interface (SBI) of the SDN controller is responsible to maintain communication with the infrastructure layer. In SDN, OpenFlow is a de-facto standard as SBI. SBI allows the SDN controller to provision physical and virtual network devices in an intelligent way.
- **Northbound Interface:** The Northbound Interface (NBI) provides flexibility to the network administrators to develop their own applications (e.g., load balancing, QoS, firewall) and integrate the applications to the SDN controller. To maintain the interaction with the application layer the SDN controllers support a number of APIs (e.g., REST).
- **East-West Bound Interface:** To manage the cluster of multiple controllers as well as communication between multiple controllers, Westbound Interface can be used. On the other hand, Eastbound Interface can be used to interact with other third-party applications. However, there is no standard interface for these purposes.

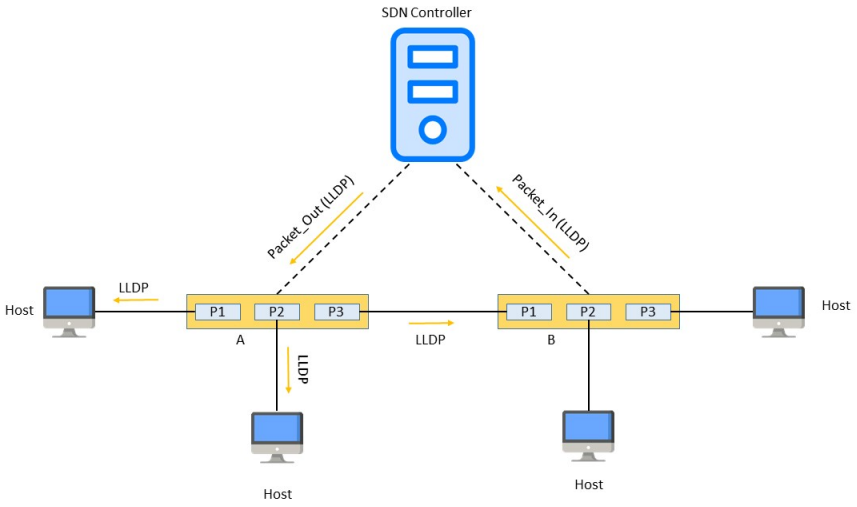

FIGURE 7.3: LLDP protocol methodology

7.3.2 Classification and Comparison of SDN Controllers

The working principle of the SDN controllers is almost the same but the classification criteria of the SDN controllers mostly depends on some properties that include programming language, architecture, application programming interface (API), platform, and interface. Table 7.2 presents a comparison of the most popular SDN controllers.

Table 7.2: Comparison of SDN Controllers.

	Floodlight	**NOX**	**ONOS**	**POX**	**Ryu**	**OpenDaylight**
Programming Language	Java	C++	Java	Python	Python	Java
Architecture	Centralized	Centralized	Distributed Flat	Distributed Flat	Centralized	Distributed Flat
Northbound API	REST, Java RPC, Quantum	ad-hoc	REST, Neutron	ad-hoc	REST	REST, RESTCONF, XMPP, NETCONF
Southbound API	OpenFlow 1.0, 1.3	OpenFlow 1.0	OpenFlow 1.0, 1.3	OpenFlow 1.0	OpenFlow 1.0-1.5, NETCONF, OF-Config	OpenFlow 1.0, 1.3
Supported Platform	Linux, MacOS, Windows	Linux	Linux, MacOS, Windows	Linux, MacOS, Windows	Linux, MacOS	Linux, MacOS, Windows
Interface	CLI, Web UI	CLI, Web UI	CLI, Web UI	CLI, GUI	CLI	CLI, Web UI
Documentation	Good	Limited	Good	Limited	Good	Good

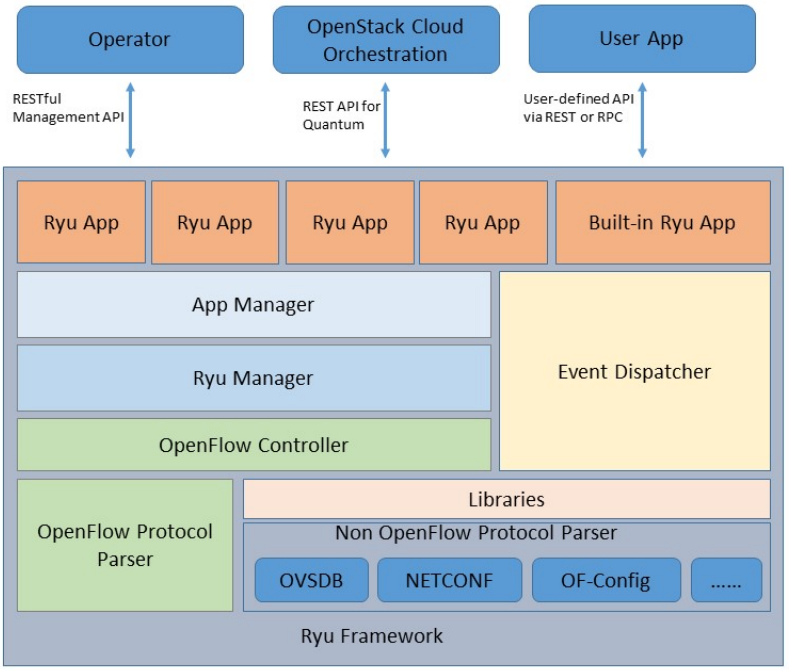

FIGURE 7.4: Architecture of Ryu framework

Among the other SDN controller platforms, we have chosen Ryu for our experimental evaluation as Ryu is an open-source, component-based SDN controller platform that is entirely implemented in Python. Like other SDN controller platforms, Ryu also provides software components with well-defined APIs, which adds more flexibility to the developer to create new applications for network management and control.

7.3.3 RYU Architecture

Ryu architecture provides the developer with a platform that is equivalent to an operating system without any application software installed in it. According to the demand, the developers need to write their own applications based on the framework and API (e.g., REST) provided by Ryu. Figure 7.4 presents the architecture of Ryu framework. The Ryu framework has the following main features:

- **Ryu Libraries:** Ryu enriched with a collection of libraries that allows parsing and building various protocol packets, such as VLAN, MPLS (Multiprotocol

Label Switching), GRE (Generic Routing Encapsulation). Ryu also has libraries to support non-OpenFlow protocols (e.g., NETCONF, OVSDB).

- **Protocol Support:** Ryu brings one important strength by supporting multiple southbound protocols (e.g., OpenFlow, NETCONF, OVSDB, OpenFlow Management and Configuration Protocol (OF-Config) [9]) to manage forwarding devices.

- **OpenFlow Protocol (OFP) Parser and Controller:** In the Ryu framework, the controller is a key component that manages configurations of flow rules in the forwarding devices and also managing events. When an OpenFlow message is received in the receiving queue, the OpenFlow protocol parser is called by the controller to understand the OpenFlow message. The OFP Parser then generates the proper OFP Event and labels it as *OFPxxx*. For instance, a *Packet_In* event becomes *OFPPacketIn*.

- **Event Dispatcher:** After the generation of the proper OFP event, the event dispatcher is called by the controller to construct the event object. The event dispatcher has specifications about how to dispatch an event to the appropriate Ryu app.

- **Ryu Manager:** The Ryu manager is the executable of the Ryu framework. When the Ryu manager is executed, it listens to the specific IP address and specific TCP port (by default:6633). The OpenFlow switch (e.g., Open vSwitch) gets connected to the Ryu manager to this port.

- **App Manager:** The main functional component for all Ryu applications in the Ryu framework is the app manager. All the Ryu applications inherit the app manager´s RyuApp class.

- **Ryu Applications:** The Ryu framework is distributed with multiple applications, like simple_switch, router, firewall. Ryu applications contain user logic and are single-threaded entities that implement various functionalities in the Ryu framework. Each application receives events in its *Event queue*, which is mainly a FIFO (First IN First Out) queue in order to preserve the order of the events. Each Ryu application runs a thread to consume the events from the queue. The *Event processing* thread pops out the event from the queue and

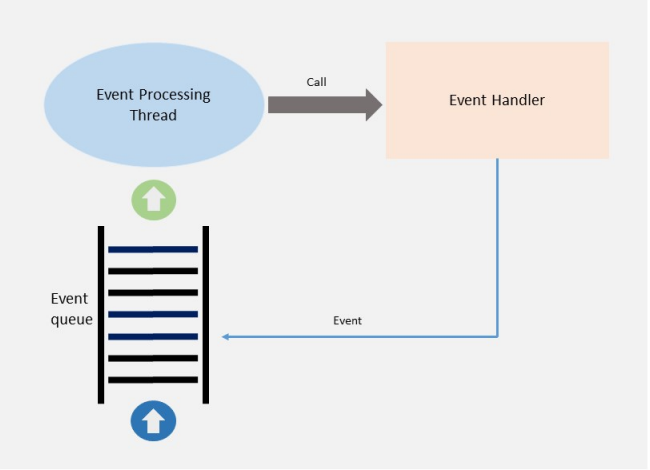

FIGURE 7.5: Functional Architecture of Ryu Application

calls the appropriate *Event handler* for the event. In this context, the *Event handler* works in a blocking mode, i.e., when an event handler is given control, no further event is processed by the *Event processing* thread until the control is returned to the *Event processing* thread. Figure 7.5 presents the functional architecture of Ryu applications.

- **Northbound Interface:** In the northbound interface, Ryu supports a number of APIs (e.g., REST) to interact with the application layer.

7.4 Experimental Setup

Figure 7.6 illustrates experimental setup where custom topologies that include data plane nodes (e.g., Open vSwitch), are created in Mininet. The veth pair, which is one of the fundamental network elements in Mininet, is used to connect the nodes in the topology. The SDN controller (e.g., Ryu) to make the policies for data plane nodes, is placed on an emulated host in Mininet. More explanations of the evaluated scenario are presented in chapter 8.

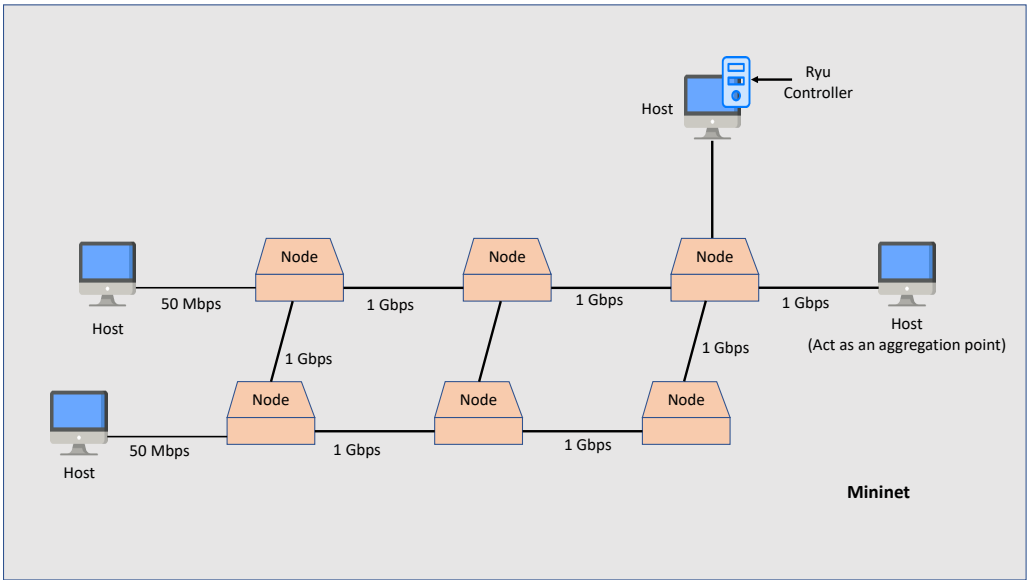

FIGURE 7.6: Experimental Scenario Setup in Mininet

Chapter 8

Scenario Evaluation and Experimental Results

This chapter first presents the performance evaluation of canonical SDN, which is only centralized, under unreliable conditions. The analysis of the distributed mode of operation during unreliable channel conditions is also reported in this chapter. Furthermore, this chapter also includes the performance of hybrid SDN during unreliable conditions. Hybrid SDN switches the mode of operation at the node level depending on the decision taken by the control logic switching algorithm described in section 6.3. The contributions of this chapter have been published in [53], [54].

8.1 Overview of Evaluated Network Scenario

Using Mininet, we evaluated custom topologies of 9-node and 16-node grid mesh networks (see Figure 7.6). The reason for considering mesh networks is that in a mmWave mobile transport network, a mesh topology is an ideal candidate to interconnect the SCs at the transport level, where, for instance, the SC is placed on the street furniture (e.g., a lamppost) [27]. Moreover, we considered an in-band deployment of the SDN scenario, which is realistic when deploying SDN in multi-hop wireless transport scenarios. In contrast to the out-of-band deployment of SDN, where a separate network is required to exchange control messages between the data

plane node and the SDN controller, the in-band solution minimizes the requirement of a separate network by sending control messages over the same channel used for data packets. We considered Ryu as SDN controller platform to make policies for packet forwarding. We developed our Ryu application that contains the user logic for packet forwarding, data plane failure handling and network loop handling.

In the evaluated multi-hop wireless scenario, only one SC is connected directly to the aggregation node (i.e., macro-cell) and acts as a gateway node for other SCs that are in the coverage area of the macro-cell. Optical fiber technology or mmWave technology can be used to connect the gateway SC to the macro-cell base station. The other SCs in the coverage area are connected to each other through wireless links to form a mesh network. The main idea behind this is that mmWave-based transport network is a cost-effective candidate to connect SCs rather than laying fiber to connect SCs to each other. Moreover, mmWave is able to achieve data rates up to gigabit-per-second [56], which is very close to the data rate provided by fiber technology. In this way, the coverage of the macro-cell is complemented with that of SCs. Moreover, the SDN controller is also connected to the gateway SC and communication between the SDN controller and the SCs is maintained via (the transport component) the gateway SC.

In our evaluated SC based transport scenario, the transport node associated with each SC is a hybrid node that connects heterogeneous networks (e.g., sensor networks, wifi networks, and mobile ad-hoc networks) to the macro-cell base station of a mobile network. We use TCLink [24] to emulate high capacity mmWave links (60GHz 802.11ad WiGig links) as wireless transport links and to set their link rate up to 1 Gbps. Multiple TCP flows were generated from different SCs by using iPerf [22], which, on average, resulted in a traffic rate of 50 Mbps each. All the TCP traffic flows generated from each SC are sent toward the macro-cell base station via a gateway SC, which in a realistic scenario may be co-located with a macro base station cell site and SCs deployed in lampposts, for instance.

8.1.1 Parameters Used in The Experimental Evaluation

Table 8.1 explains the parameters used in our evaluations. Extensive evaluation campaigns of the centralized and the distributed network under various unreliable

network conditions for different network sizes allowed tuning the algorithm parameters ($CPLR_{max}$ and δ) to the values reflected in the table. Furthermore, 10 s. was found to present an appropriate trade-off between the processing overhead (much higher if evaluation periods were much shorter) yet being able to detect steady trends based on which the algorithm takes decisions.

Finally, the tested network sizes, CPLR values, and number of impaired links were selected to reflect networks with various complexities and a variety of operational conditions ranging from perfect to really bad, so that the algorithm could show its operation under stress and have its operational limits assessed. All tests were repeated 10 times, and their min, 25-percentile, median, average, 75-percentile, and max values are represented in the boxplots.

TABLE 8.1: The experimental parameters and their corresponding values.

Parameter	**Value**	**Explanation**
$CPLR_{max}(\%)$	15	CPLR value over which the network is not operational in centralized mode.
δ	0.5	Slope that if exceeded for two consecutive periods implies switching to distributed mode.
Period	10	Period (in s.) with which CPLR is evaluated by local agent of each node.
Network size	9, 16	Number of nodes in the grid.
CPLR (%)	5, 10, 15, 20, 25	Average CPLR values generated to test network performance.
Links impairments	0, 1, 2, 3, 4, 5	Number of link impairments (switching links up/down) randomly distributed through the network (yet guaranteeing having a connected graph).

8.2 Canonical SDN Operation: Imperfect Control Channel and Imperfect Data Plane Condition

Several experiments were conducted to observe the throughput and latency performance of canonical SDN during unreliable control channel and data plane conditions. To observe the impact of the unreliable control plane over data plane throughput and latency in a controlled way, we generated losses of control messages that were exchanged between the forwarding nodes and the SDN controller. We used netem [23] in combination with TCLink to add impairments to all of the communication channels (including the channel that connects the SDN controller to the gateway node) in our emulated scenarios.

We also broke data plane links (e.g., the link between two nodes) randomly by maintaining a sequence of link down/up at arbitrary periods of our emulation time. We did so to emulate the extreme instability that could appear in a wireless channel, and, in this sense, it represents a quite complex situation to be handled by the control plane. As with data plane link failure, control messages (*PortStatus*, *FlowMod*, *PacketIn*, and *PacketOut* messages) [14] continue being exchanged between the SDN controller and data plane nodes in order to redirect these control messages to the SDN controller over the degraded channels. Moreover, in our evaluated scenarios, every 5 s., LLDP messages are exchanged between the SDN controller and the forwarding nodes; the SDN controller knows about the topology by receiving LLDP packets from the data plane nodes.

We performed evaluations with different network sizes under unreliable conditions, by injecting various flows in order to understand what were the CPLRs that allowed the control communication channel to operate (even if with difficulties) and those for which it was impossible in most repetitions. Figures 8.1–8.4 depict the data plane performance metrics (aggregated throughput and average latency) during unreliable control plane conditions of canonical SDN for various network sizes (blue curve for three flows and red curve for five flows in the centralized case).

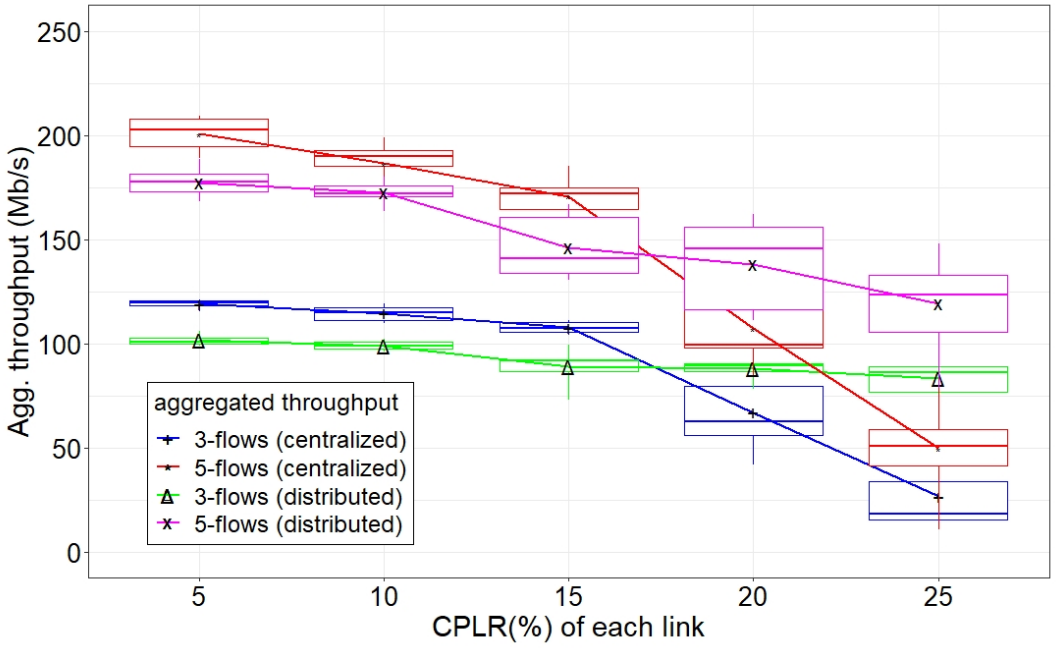

FIGURE 8.1: Data plane performance metric (aggregated throughput) comparison between centralized and distributed operations, during imperfect control channel conditions in a 9-node grid network.

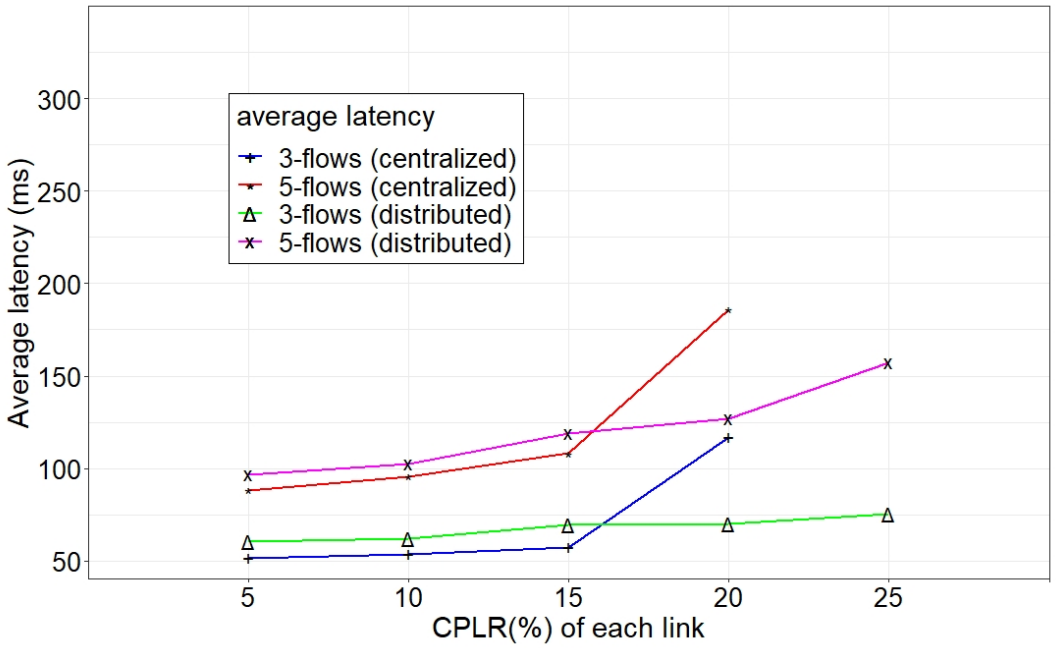

FIGURE 8.2: Data plane performance metric (average latency) comparison between centralized and distributed operations, during imperfect control channel conditions in a 9-node grid network.

We injected three and five TCP flows in each topology and varied the CPLR of each link. We also broke four data plane links randomly during our emulation time. For both network sizes, the results reveal that the increase of CPLR caused degradation of the data plane throughput and substantial growth of the latency in the canonical SDN scenario, which is very high for a CPLR of 25% (not represented). In the evaluated scenario, during the presence of link impairments in the data plane, the affected nodes sent *PortStatus* messages to the SDN controller to inform the SDN controller about data plane link failure.

In this case, due to the presence of faulty control communication links between the SDN controller and the forwarding nodes, the control messages (i.e., *PortStatus*, *PacketIn*, *PacketOut*, and *FlowMod*) were dropped and could not be exchanged on time. Moreover, LLDP messages that were used by the SDN controller for discovering network topology, were also dropped. As the SDN controller and forwarding nodes used a TCP connection to maintain the communication between them, dropped packets were retransmitted again.

Therefore, for low CPLR values (e.g., 10%) throughput was maintained to an acceptable level through TCP retransmission of control messages. However, in the case of high values of CPLR (e.g., 25%), although dropped control packets were retransmitted again, the high-loss of the control packets affected the restoration of a new path at the data plane, which rendered the network unusable, as the impact was the same as if the SDN controller had failed. The impact on the aggregated throughput and latency were also highly noticeable.

To evaluate the impact of control message loss over data plane performance, we generated different numbers of link failures at the data plane with the intention of redirecting control messages to the SDN controller over lossy control communication channels. Figures 8.5–8.8 report the data plane performance of canonical SDN while the network was experiencing an increasing number of data plane link impairments under a certain CPLR value. We repeated the experiments for different network topologies by injecting seven TCP flows in each topology.

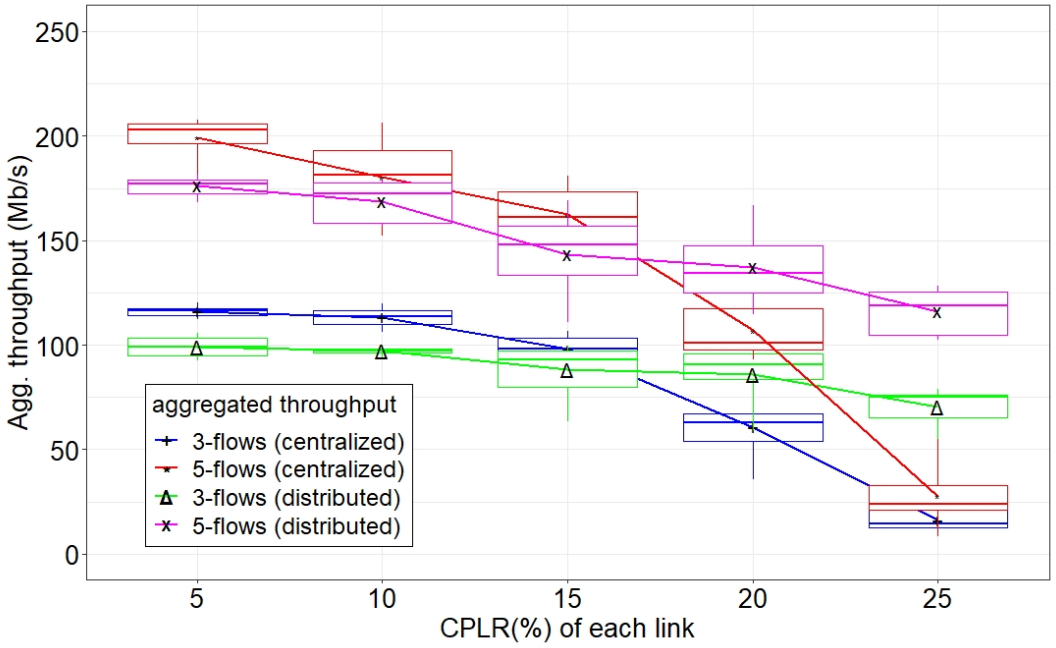

FIGURE 8.3: Data plane performance metric (aggregated throughput) comparison between centralized and distributed operations, during imperfect control channel conditions in a 16-node grid network.

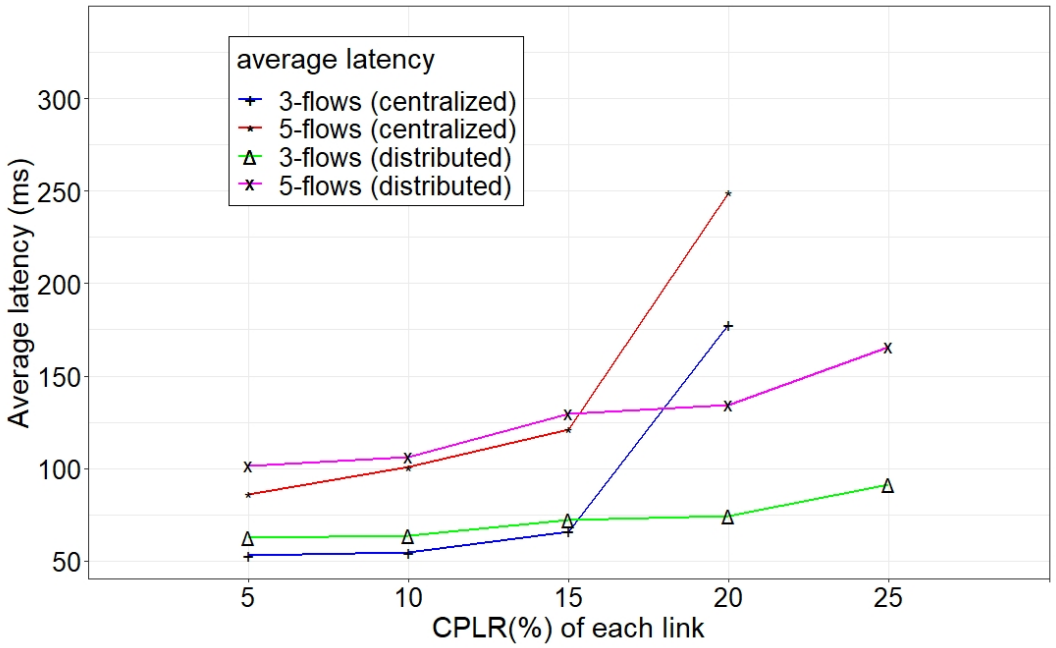

FIGURE 8.4: Data plane performance metric (average latency) comparison between centralized and distributed operations, during imperfect control channel conditions in a 16-node grid network.

We observe that, for both topologies, the aggregated throughput at the data plane substantially degraded, and the latency increased with the increasing number of broken links at the data plane. This was evaluated for various values of CPLR to assess its dependency on varying degrees of data plane link impairments. As CPLR increased, so did the restoration time due to control packet loss and the consequent retransmission of control packets between the network nodes and the centralized SDN controller. For both network topologies, for CPLR values of 25%, the degradation of the aggregated throughput at the data plane was remarkable, and the latency was substantially higher due to the high-loss of control packets, which caused several retransmissions that affected the proper communication between the SDN controller and the forwarding nodes.

As latency is very high during a CPLR of 25%, for both cases, this is not represented in the figures. The network topology is a grid, and the connectedness of the graph is maintained, which may not be the case in a real deployment with lower nodal degrees of the graph. In this sense, the network performance presented in the figures may be considered the best possible case in terms of the path diversity.

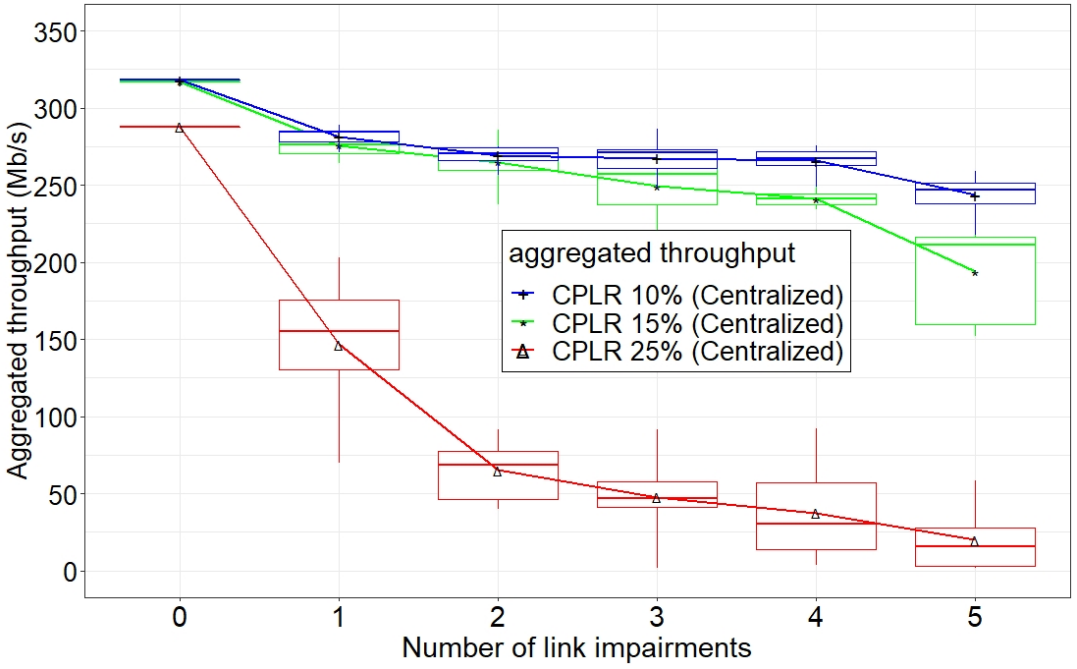

FIGURE 8.5: Aggregated throughput behavior of centralized network operation during data plane link failures for various times under imperfect control channel conditions in a 9-node grid network.

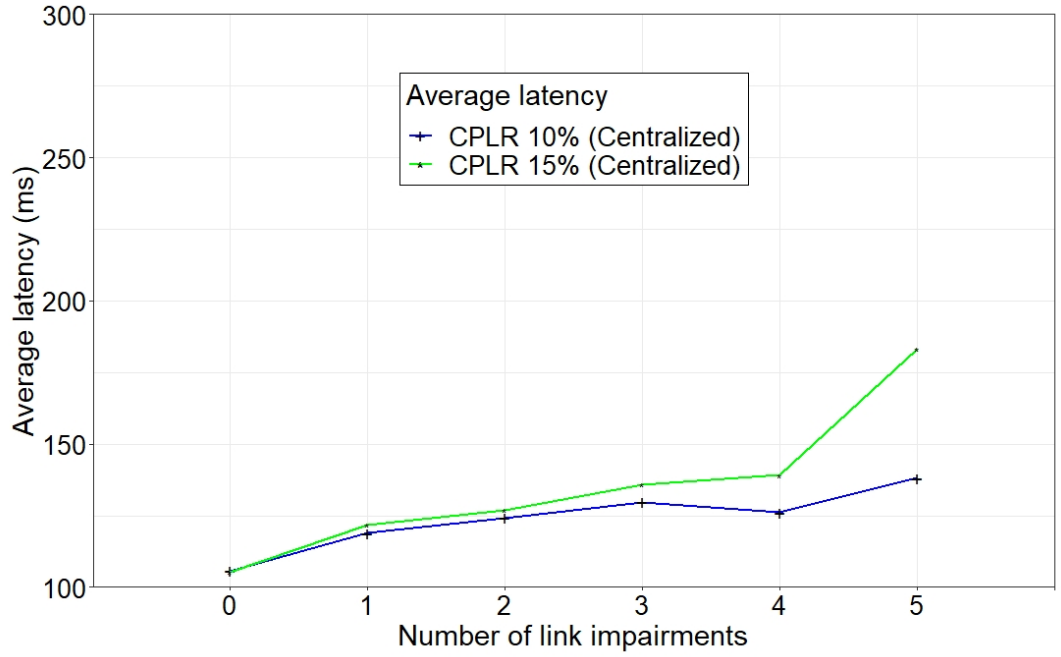

FIGURE 8.6: Average latency behavior of centralized network operation during data plane link failures for various times under imperfect control channel conditions in a 9-node grid network.

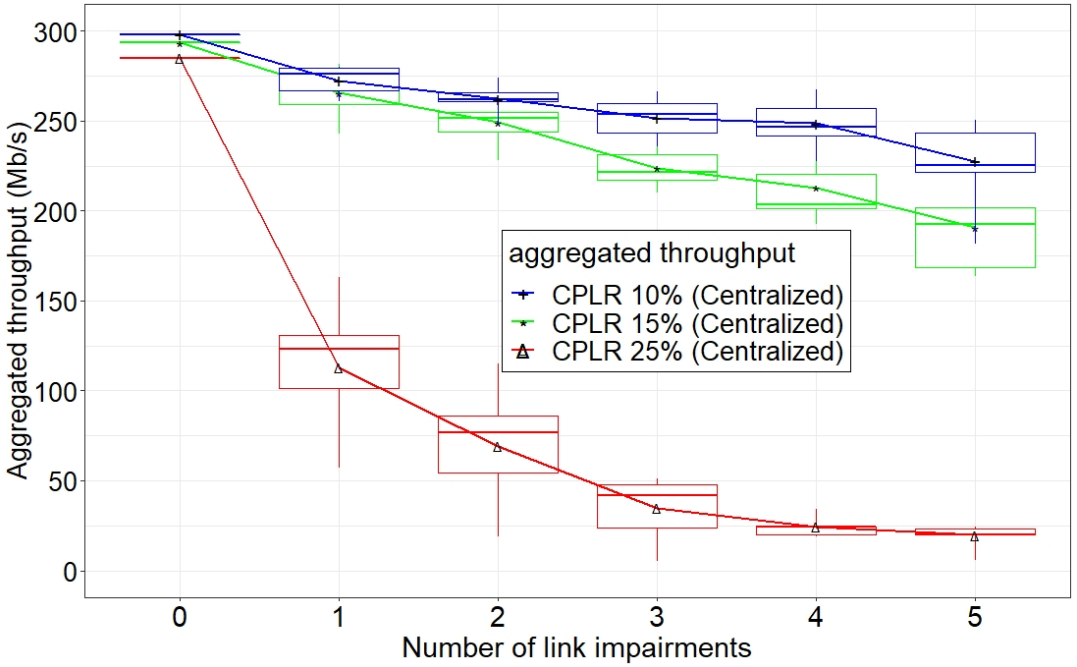

FIGURE 8.7: Aggregated throughput behavior of centralized network operation during data plane link failures for various times under imperfect control channel conditions in a 16-node grid network.

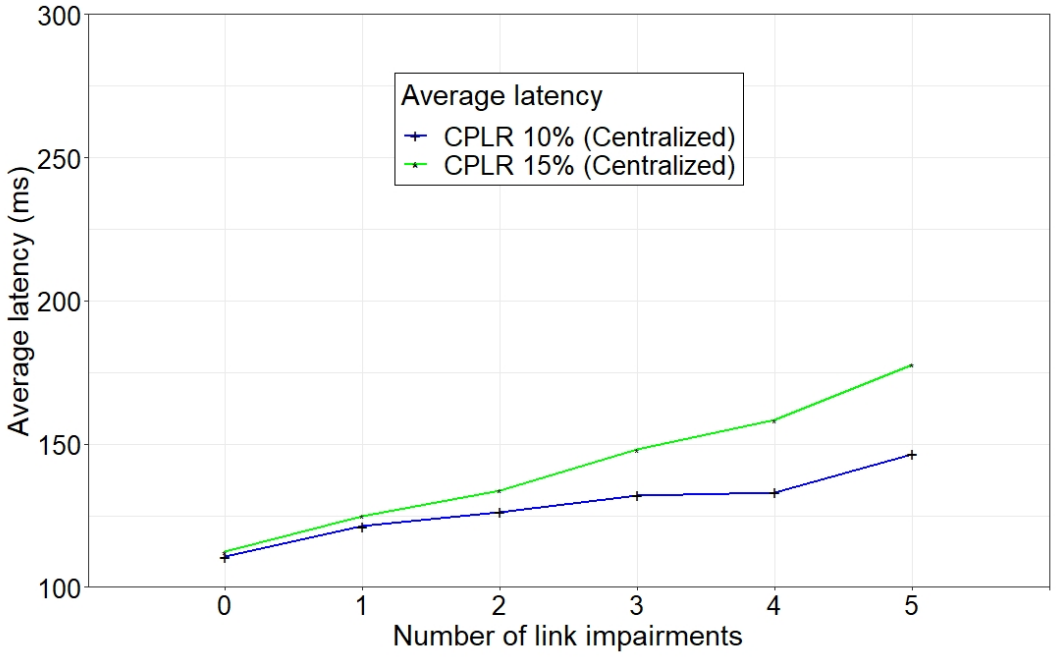

FIGURE 8.8: Average latency behavior of centralized network operation during data plane link failures for various times under imperfect control channel conditions in a 16-node grid network.

8.3 Distributed Operation: Imperfect Channel Condition

We evaluated the same network topologies with the same number of TCP flows in the distributed network. We conducted our experiments in the distributed case where all the links were degraded communication links, and, in a controlled way, we increased the loss of control messages exchanged between two neighbor nodes. In the case of our evaluated distributed scenario, Hello and LSA (Link State Advertisement) messages are periodically exchanged among the neighbor nodes.

In our setup, within every 2 s., Hello messages are exchanged between two neighbor nodes that confirm the availability of the neighboring nodes. On the other hand, every 5 s., LSAs are exchanged by the nodes in order to learn about the topology of the network. During any change in the network topology that may be caused by link failure or the unavailability of a node, the affected node floods updated LSAs

in the same area network. By receiving a copy of updated LSA the other nodes in the same area learn about the changes in the topology.

In our evaluated scenario, we generated losses of these control messages in order to investigate the throughput and latency performance of distributed operation during unreliable conditions. We degraded the control communication channel by dropping control messages, and we also changed the network topology by breaking links in the network. We did so in order to flood the LSA by the affected node in the area and to observe the behavior of the network in terms of network metrics during unreliable conditions.

When there is a change of topology (e.g., link failure happens), the nodes exchange the updated LSAs to learn about the topology changes. However, due to the degraded control communication channels, some control messages are dropped, which, in turn, affects the network convergence time. The dropped control messages are retransmitted again; however, such losses affect the throughput and latency performance, as there is some delay in the reestablishment of routes caused by control packet loss.

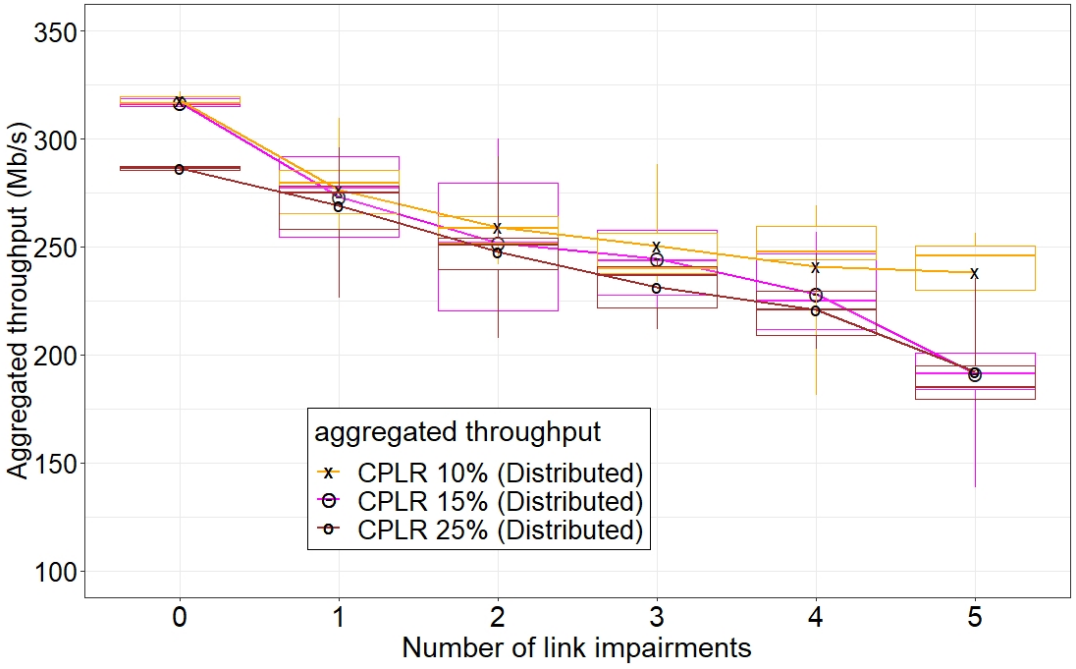

FIGURE 8.9: Aggregated throughput behavior of distributed network operation during link failures for various times under imperfect channel conditions in a 9-node grid network.

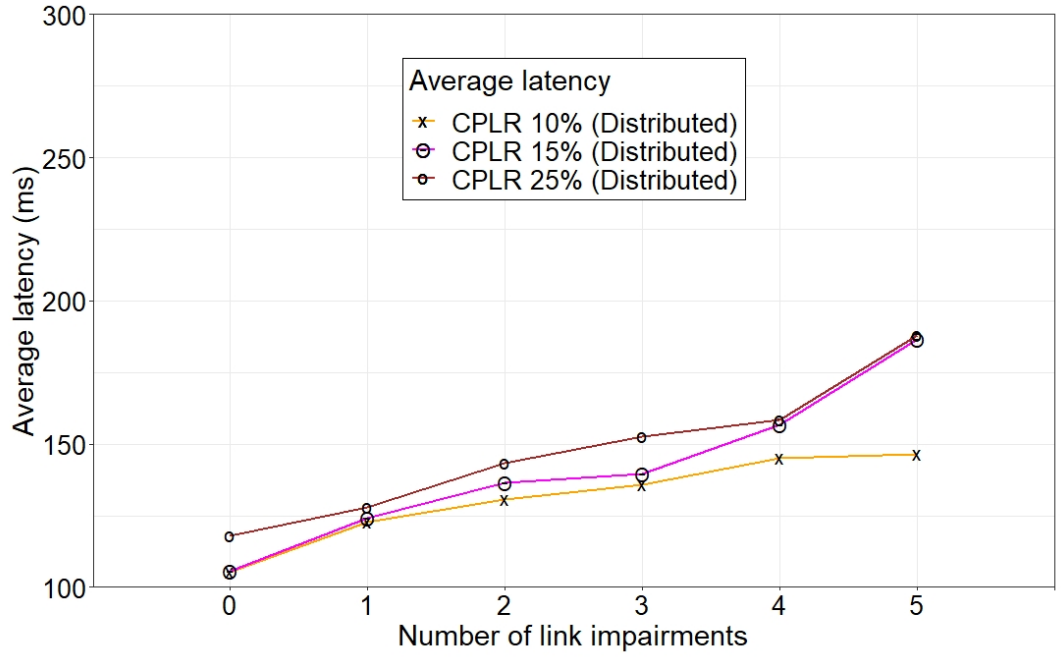

FIGURE 8.10: Average latency behavior of distributed operation during link failures for various times under imperfect channel conditions in a 9-node grid network.

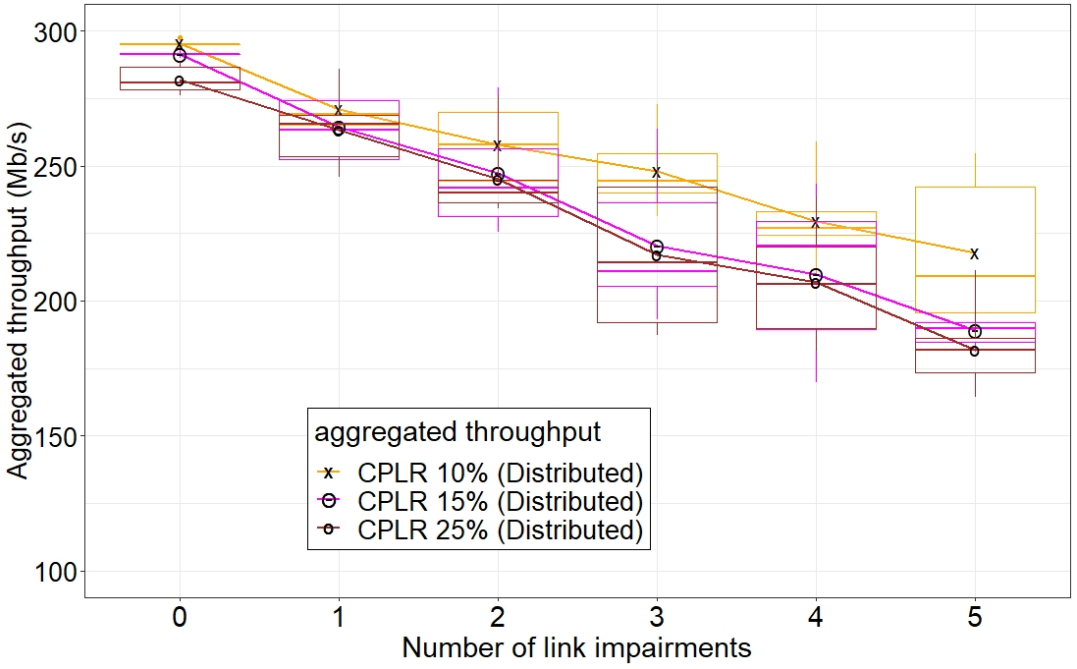

FIGURE 8.11: Aggregated throughput behavior of distributed network operation during link failures for various times under imperfect channel conditions in a 16-node grid network.

From Figures 8.1–8.4, it can be observed that, in distributed network operation (green curve for three flows and purple for five flows in distributed case) and for low CPLR values (e.g., 5%), the aggregated throughput was higher, and the latency was lower because of lower control packet loss and also due to the lower retransmission rate of lost control messages. As CPLR increased, there was a degradation of the aggregated throughput while the latency went upward. With the increase of CPLR values, the loss of control messages increased, which also increased the rate of retransmissions. For this reason, and for higher values of CPLR (e.g., 25%), the aggregated throughput declined while the latency increased.

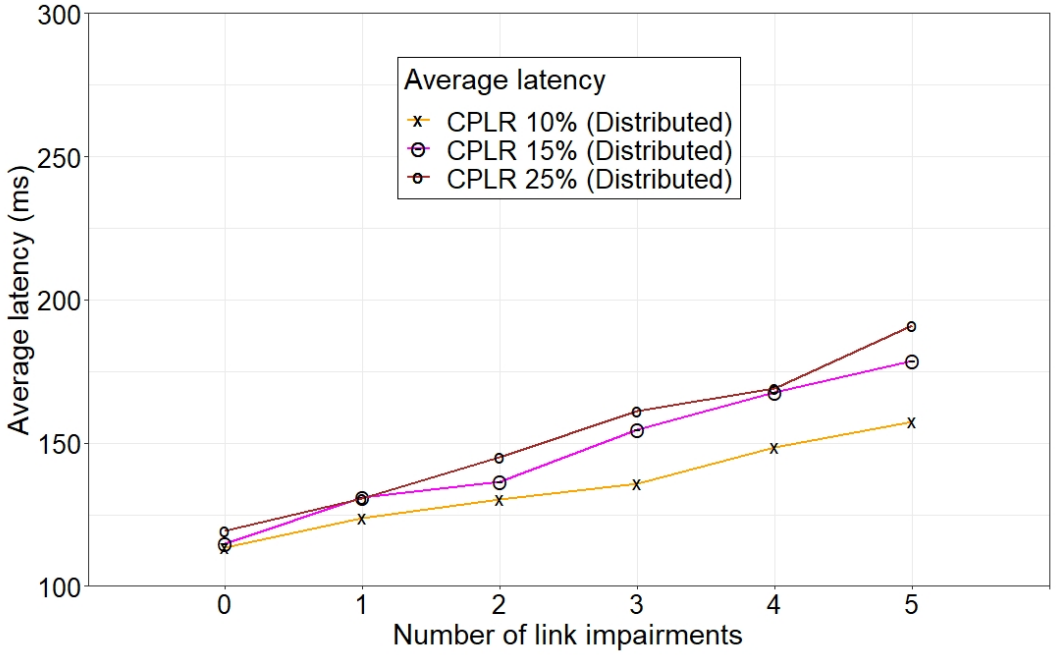

FIGURE 8.12: Average latency behavior of distributed operation during link failures for various times under imperfect channel conditions in a 16-node grid network.

In another experiment, we investigated the performance of distributed operation during topology changes under unreliable channel conditions. In this way, for a certain value of CPLR, we changed the network topology several times by varying the number of broken links. As Figures 8.9–8.12 depict, a high number of link failures under the high CPLR regime (e.g., 25%) caused a decline in the aggregated throughput and an increase in the latency. Due to a high-loss of control messages, during topology changes for several times, the exchange of the updated LSAs between nodes

suffered from losses, which caused more retransmissions and the consequent delay of control messages that affected the throughput and latency of the network.

8.4 Performance Comparison: Canonical SDN and Distributed Networks During Imperfect Channel Condition

To observe the performance of centralized canonical SDN and distributed operation under unreliable channel conditions, for both cases we generated losses of the following messages in a controlled way.

- Centralized case: control messages (i.e., *PortStatus*, *PacketIn*, *PacketOut*, and *FlowMod*) exchanged between the forwarding nodes and the SDN controller.
- Distributed case: control messages (i.e., Hello and LSA) exchanged between nodes that keep track of neighbor availability.

As Figure 8.1–8.4 report, for both network sizes during low CPLR values, centralized operation performed better (in terms of the throughput and latency) than distributed operation due to lower retransmissions of control packets due to low loss of control packets, and the delay of control packets to reach the SDN controller is lower.

When the CPLR increased above 15%, distributed operation outperformed centralized operation by maintaining higher throughput and lower latency, as higher retransmission of control packets was required for centralized operation, and also delay was incurred by the control packets. Moreover, Figures 8.5–8.8 illustrate that, for both network sizes and for high CPLR values (e.g., 25%), while the topology of the network changed very often, centralized operation failed to operate correctly. On the other hand, distributed operation kept the network operational (see Figures 8.9–8.12).

For low values of CPLR, the centralized operation performed well, as lower retransmissions and small delays were experienced by the control packets to reach the SDN

controller, and there was a lower convergence time as the network graph was already known by the SDN controller. However, for high CPLR values, due to the high-loss of control packets, the exchange of control messages substantially affected (due to higher retransmissions and consequent delay) the TCP-based control communication between the SDN controller and forwarding nodes. For this reason, if the CPLR of the links increased, the performance of the centralized operation substantially degraded, as it depends on a centralized SDN controller, which becomes a single point of failure in high-loss conditions.

On the other hand, in the case of distributed mode, while there is a topological change in the networks, LSAs are being flooded. During high-loss conditions if a node fails to receive a copy of the LSA from a neighbor node due to the degraded channel, it might still receive a copy of that LSA from another neighbor node. In this way, the distributed link-state routing can converge during topology changes and shows better performance than the centralized operation under high control packet loss conditions.

8.5 Hybrid SDN Operation

Sections 8.2 and 8.3 illustrate the performance evaluation of the canonical SDN and the distributed networks during unreliable conditions. To investigate the performance of the canonical SDN and distributed network operation, the network performance metrics, i.e., aggregated throughput and average latency, have been analyzed for different network sizes and several injected flows. From our evaluated scenarios, by examining the performance of the centralized and distributed modes during unreliable control communication channel conditions, we characterized the switching point between centralized and distributed operation in our hybrid control plane to the CPLR value of 15% and set it as $CPLR_{max}$.

From the previous sections 8.2 and 8.3, it can be observed that below the point $CPLR_{max}$, centralized operation performed better, and, above it, distributed operation outperformed the centralized operation (see Figures 8.1–8.4). Section 8.4 describes the reasons behind this. In this section, we quantify the gains that hybrid SDN operation offers. In this direction, we emulate the same network topologies by

injecting the same number of flows; however, the difference is that the forwarding nodes are hybrid nodes, where centralized and distributed operations coexist and the nodes can take autonomous switching decision between both modes.

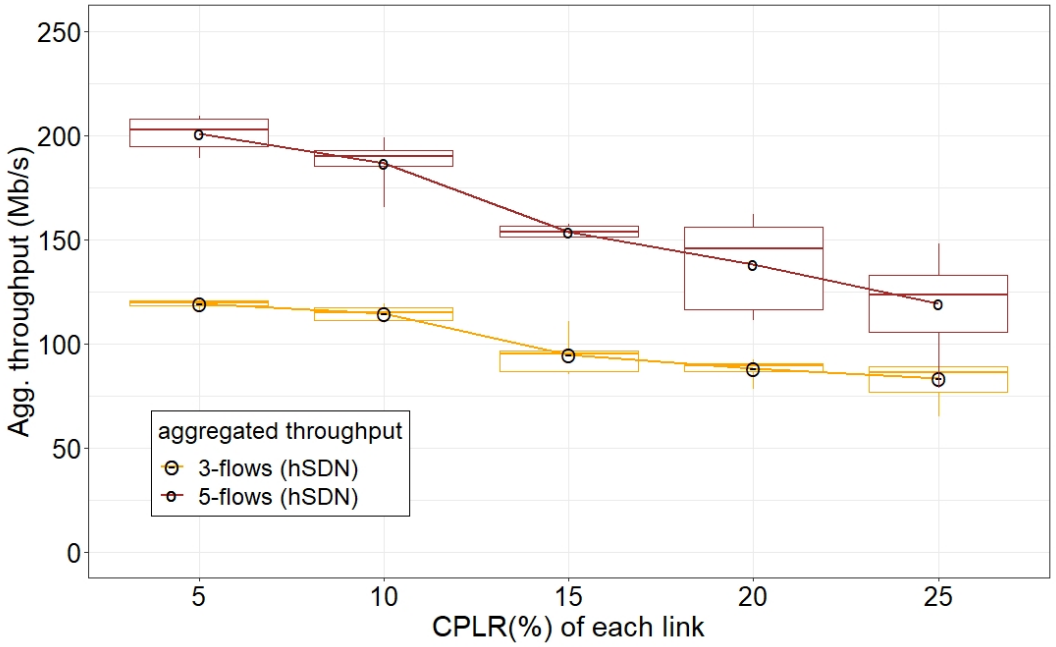

FIGURE 8.13: Aggregated throughput metric of data plane in hybrid SDN for increasing CPLR in 9-nodes topology.

8.5.1 Centralized to Distributed: Recovery Operation

In our hybrid SDN setup, initially the control communication channels are in reliable condition and the network is operated in centralized mode. The module *CPLR detector* of the local agent that is integrated into nodes (see Figure 5.2) periodically monitors the CPLR of the communication links every 10 s. Then, we start increasing the CPLR, and when the CPLR of the links increases, the *decision module* (running Algorithm 1) of the local agent that is integrated into the each node, dynamically takes network operation switching decisions, and the local agent performs network operation switching from centralized to distributed as described in section 6.3.1. In this case, the conceived algorithm predicts the conditions of the control communication channels by inspecting trends of impairments and takes network operation switching decisions in-advance before the centralized SDN operation reaches

the critical conditions. Figures 8.13-8.16 report data plane performance based on aggregated throughput and average latency in hybrid SDN for different topologies. Figures 8.13 and 8.15 depict, that with hybrid SDN, by switching network logic from centralized to distributed, aggregated throughput performance is improved in high-loss regimes as compared to only centralized canonical SDN (to compare please see Figure 8.1 and 8.3). For instance, 28% throughput improvement can be achieved during 20% of CPLR and a higher improvement can be achieved during higher loss periods. On the other hand, Figures 8.14 and 8.16 illustrate that, in hybrid SDN, the average latency can be maintained to an acceptable level during high-loss regimes (e.g., 25% of CPLR), while during the same conditions, the average latency is very very high in canonical SDN (see Figures 8.2 and 8.4).

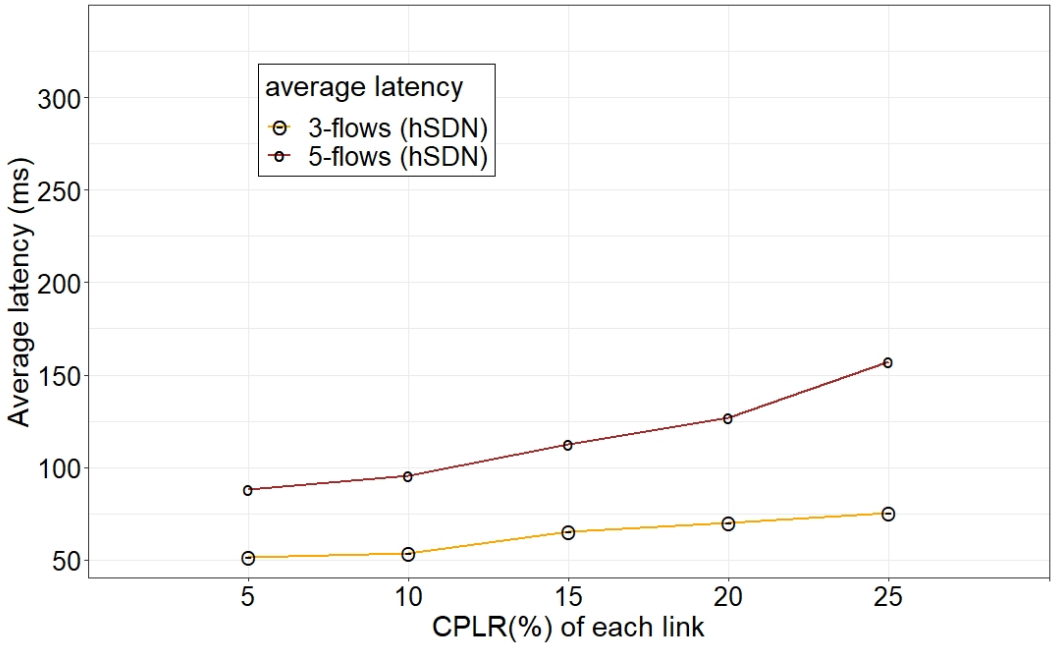

FIGURE 8.14: Average latency metric of data plane in hybrid SDN for increasing CPLR in 9-nodes topology.

8.5.2 Distributed to Centralized: Reliable Control Plane Condition

During unreliable control communication channel conditions, by inspecting trends of the impairments, the *decision module* of the local agent, which is integrated into each

node, takes network operation switching decision to distributed. While distributed mode is present in the node, the *CPLR detector* of the local agent keeps monitoring the control communication channel periodically. As soon as the channel condition became good, which is determined by measuring the CPLR, the *decision module* takes the decision to restore the centralized operation in the network and the local agent again performs network operation switching from distributed to centralized mode by following the procedure illustrated in section 6.3.2.

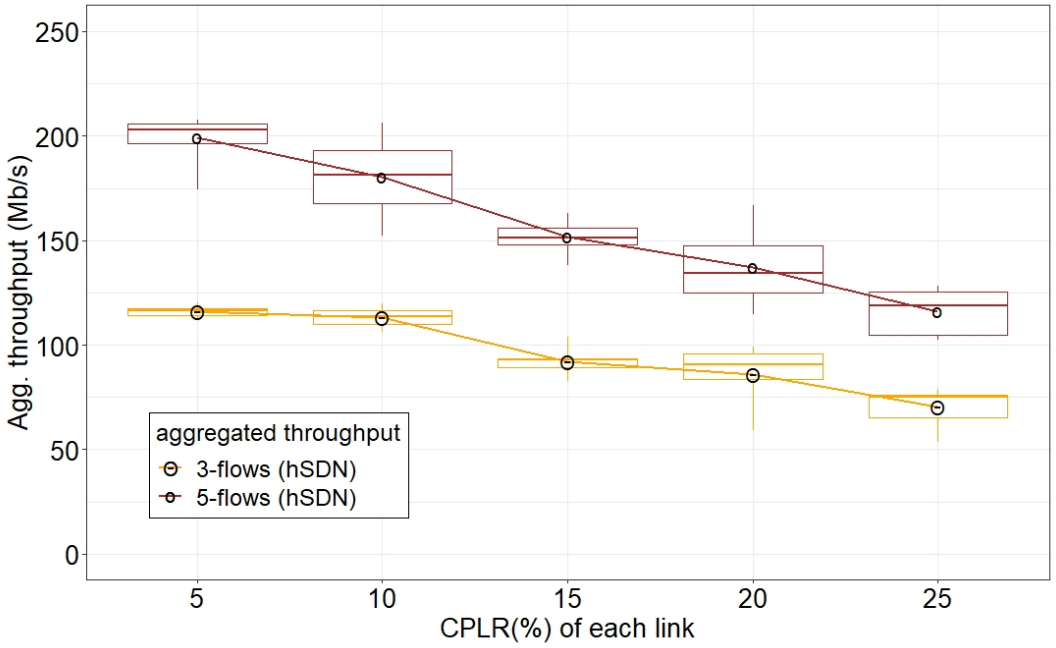

FIGURE 8.15: Aggregated throughput metric of data plane in hybrid SDN for increasing CPLR in 16-node topology.

8.5.3 Network Mode Switching Back-and-Forth: Centralized - Distributed

To illustrate the operation of the control logic switching algorithm, Figure 8.17 depicts the data plane performance during the network mode switching (i.e., centralized-distributed switching back and forth) in our hybrid SDN approach by measuring the trends of impairments of the control communication channel.

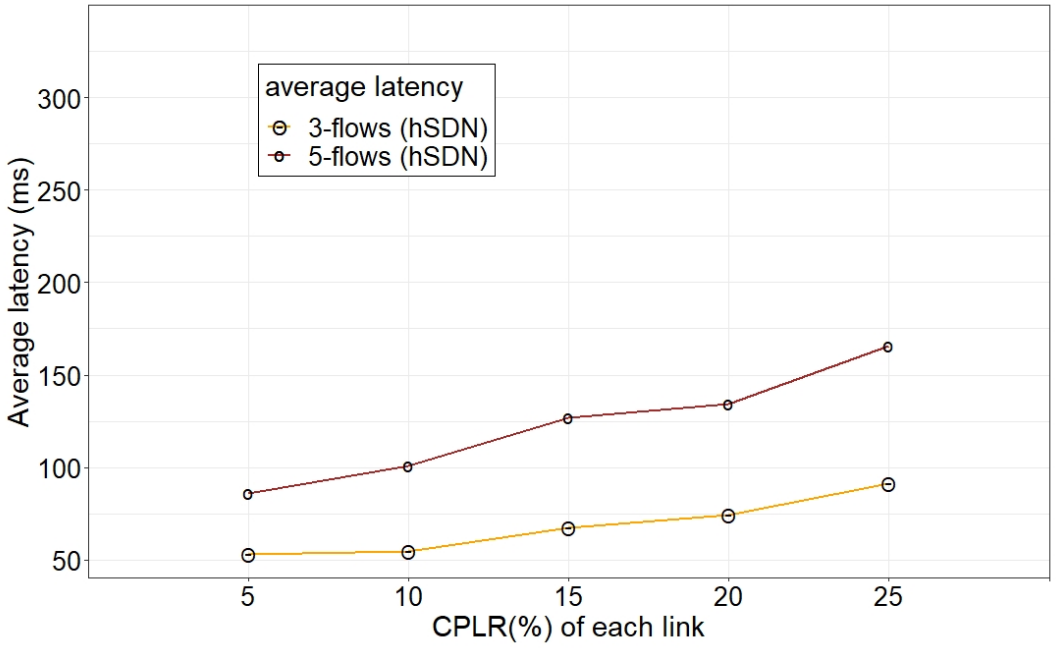

FIGURE 8.16: Average latency metric of data plane in hybrid SDN for increasing CPLR in 16-nodes topology.

Initially, all the nodes in the 16-node grid network are in centralized mode. We begin increasing impairments to the links and, at around 60 s., a high CPLR is measured, and the decision module of the local agent takes a decision based on the integrated algorithm to perform network mode switching and performs switching to the distributed operation. At around 120 s, the performance of the control communication channel becomes reliable (i.e., low loss), and, as a consequence, the CPLR become low, and the local agent restores the centralized operation based on the decision taken by the decision module. By doing so, the SDN controller gains control of the network, and the network is managed in a centralized fashion.

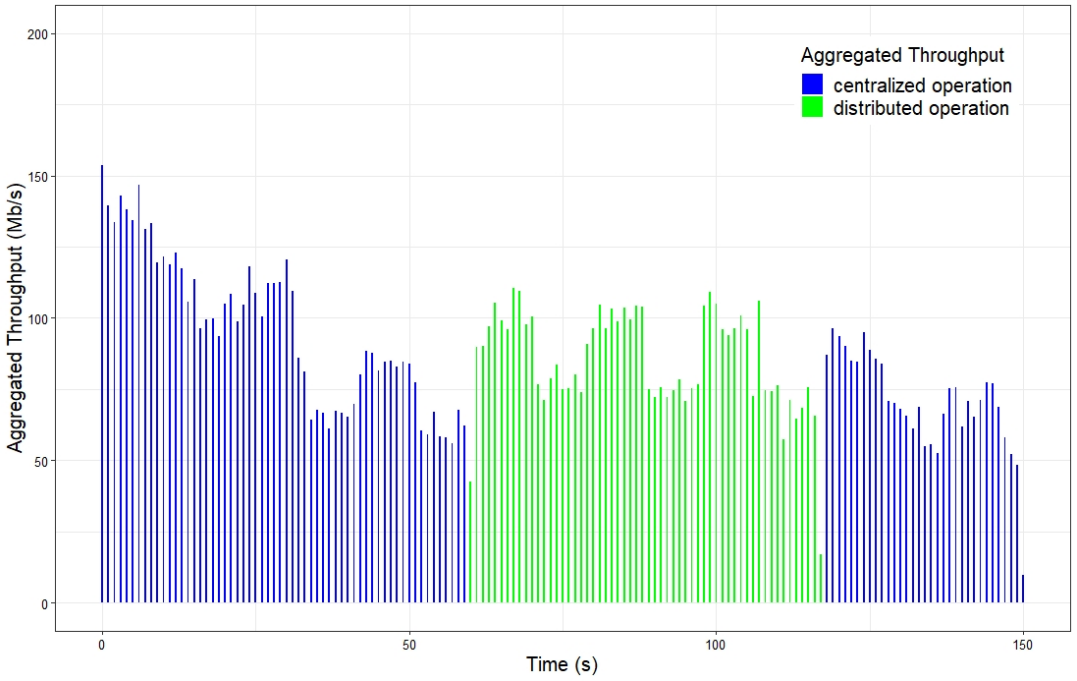

FIGURE 8.17: Aggregated throughput behavior over time during network operation switching.

Overall, these evaluations confirm that the conceived node architecture and algorithm are capable of handling multiple impairments (including those of the control plane) and maintain a stable network operation.

Chapter 9

Conclusions

Given its promise and advantages, SDN is rapidly evolving to adapt to the needs of future networks, which are increasingly embedding software components. Furthermore, the programmability of SDN avoids vendor lock-in, which enhances the interoperability between devices from different vendors, which is increasingly important in a software-oriented network in which the number of stakeholders is increasing. Moreover, the centralization feature of the SDN provides more flexibility in network management, which in turn provides remarkable benefits to manage the increasing amount of heterogeneous traffic in future networks. However, for the correct operation of SDN, it is still required to investigate several aspects of SDN operation during unreliable conditions. The separation of the control plane from the data plane imposes extra fault domains in the SDN paradigm, which brings new challenges in centralized SDN scenarios.

This research work tackles the research question defined in chapter 4 by addressing reliability in SDNs. In this direction, a novel approach has been proposed in the dissertation. This chapter describes the concluding remarks of the research work and its contribution to the research community. Finally, this chapter provides possible future research directions on this topic.

9.1 Concluding Remarks

In chapter 4, we addressed a problem statement, which is answered in this dissertation.

1. In chapter 5, we presented a hybrid node architecture that combines both centralized and distributed operation in the same node. The hybrid node is able to switch between the mode of operations to ripe the benefits of centralized and distributed operations.

2. To find out a appropriate trade-off between centralized and distributed operation, in the chapter 8, a fair comparison between centralized and distributed operations is presented. The comparison reveals that during the low loss regime centralized operation shows better performance. On the other hand, during the high-loss regime, the distributed operation outperforms the centralized operation. The SDN scenario generally assumes to have an ultra-reliable control communication channel between the SDN controller and the forwarding nodes. Nevertheless, in wireless multi-hop scenario control packets are carried by the wireless channel that is vulnerable due to environmental conditions. This research work investigates such scenarios which show that during a high-loss regime the performance of canonical SDN degrades enormously. This work also investigates the performance of the distributed operation under degraded channel scenarios and find that during a high-loss regime distributed operation maintains the throughput to an acceptable level.

3. To improve reliability of SDN in wireless transport networks, our proposed scheme autonomously switches mode of operation based on a control logic switching algorithm integrated into the node. Our proposed hybrid SDN, not only leverages the advantages of canonical SDN (e.g., simple network management, managing heterogeneous traffic) but also mitigates the drawbacks of canonical SDN in terms of reliability. Hybrid SDN indeed keeps the network operational irrespective of the level of control plane impairments by switching network control from centralized to distributed. In fact, our hybrid SDN approach switches network control logic when the CPLR of the control channel reaches an unacceptable level that limits proper communication between the

SDN controller and the forwarding nodes. In our hybrid SDN approach, due to better manageability shows by the centralized SDN, the network operation is targeted to keep in centralized mode as much as possible if the control communication channel condition as well as control plane condition allows. If the distributed operation prevails in a given node due to degraded channel conditions, by measuring the reliability of the control communication channel conditions the centralized control is restored in the networks.

4. The proposed hybrid SDN approach maintains the throughput level equivalent to the canonical SDN during low loss conditions. On the other hand, during high-loss conditions, the throughput level is maintained equivalent to the distributed operation by switching the control logic to distributed which significantly improves the throughput level.

9.2 Summary of Contributions

The main contributions of this thesis are listed below:

- One of the main contributions of the research work is the way the control plane impairments are handled. In this work, we proposed that the data plane node is responsible to measure the reliability of the control communication channels. We proposed that the CPLR is the metric that is used by the nodes to determine the reliability of the control communication channel. As the responsibility has been given to the data plane node, the failure of the controller or the control communication channel, or even degraded control communication, can be detected from the node point of view, which allows handling all kinds of failures of the control plane that may occur and are difficult to handle in centralized SDN.

- In chapter 5, the hybrid node is presented, which is the main building block of our hybrid SDN architecture. The hybrid node concurrently runs the centralized and distributed logic. A local agent has been integrated that is composed of the *CPLR detector* module, *decision module* and *rules modifier* module. The *CPLR detector* module periodically inspects the reliability of the control

communication channel by measuring the CPLR. The *decision module* adopts a network logic switching algorithm that determines the presence of the active logic in a given node. If the decision module takes the decision to switch the network logic, the *rules modifier* pushes some predefined rules to the node´s flow table.

- In chapter 6, a novel algorithm has been proposed that predicts network conditions in advance, based on the analysis of the trends of impairments by inspecting the slope of *CPLR vs. t* curve and dynamically switches the operational mode from centralized to distributed and vice versa depending on the reliability of the control communication channel.

- We have implemented and experimentally evaluated our proposed solution in Mininet, which is the most popular emulator for conducting research on SDN. We have emulated different network topologies that mimic the wireless transport network that may be required in Small Cells (SCs) deployments where fiber is not a cost-effective solution. Moreover, we have injected several flows from the SCs to the macro-cell site (the aggregation point at the transport level). During our emulation period, we varied CPLR values and generated a number of impaired links to operate the networks with various complexities. We have investigated a variety of operational conditions ranging from perfect to really bad control plane conditions, so that the algorithm could show its performance under stress and its operational limits could be assessed.

- It has been shown in chapter 8 that during critical conditions (e.g., high CPLR) canonical SDN operation fails due to lack of controller-device communication. However, our proposed hybrid SDN solution substantially improves network performance in terms of aggregated throughput and latency, particularly when control channel packet loss ratios increase. For instance, 28% throughput improvement for 20% control packet losses and even more during higher control packet losses. Moreover, our proposed solution also brings a significant improvement in latency during high-loss condition, which is very high in canonical SDN. This allows offering a reliable network operation in hard conditions under which a centralized canonical SDN control would result in a non-operational network.

9.3 Future Work

The dissertation not only contributes to the research community to tackle the reliability issue of canonical SDN, but it also opens a window for future research directions:

- In our experiments, we have degraded all the communication channels at a time and all the nodes in the network perform operational mode switching to distributed. It could be worth investigating in more detail in the future scenarios in which there may be heterogeneous nodes in the network, trying to reflect incremental deployments of the technology. In such scenarios, maintaining a complete coherent network view and the interaction between nodes operated in centralized mode and nodes operated in distributed mode smoothly may pose some additional challenges.

- Throughout the work, reliability has been translated into maintaining stable values of throughput and latency despite impairments. But Available Bandwidth (ABW) [50] measurement is one of the significant metrics in SDN to get information about the current load on the links as well as on the network to be able to apply more advanced traffic engineering. In an SDN scenario, to measure ABW, the controller periodically performs polling from the forwarding devices using *PortStatusReq* messages. While delay increases in communication between the controller and the forwarding devices, the ABW estimation error also increases, which has an adverse impact on path selection for some services. This may require further analysis in in-band SDN deployments in wireless multi-hop scenarios for which communication between the controller and the forwarding devices may be repeatedly impaired. The degraded control-communication link may incur delay or even loss of control packets. Adequately handling such measurements, may improve traffic engineering in such transport networks.

- Artificial Intelligence (AI) and Machine learning (ML) are the key technologies adopted by future communication networks for automated network management. It would be worth investigating under what circumstances and to what extent AI/ML-based techniques can improve hybrid SDN network operation

without adding much additional computational complexity to transport nodes also considering the control communication channel impairments.

Bibliography

[1] Beacon. https://openflow.stanford.edu/display/Beacon/Home.

[2] Estinet: Openflow network simulator and emulator. https://www.estinet.com/ns/.

[3] Floodlight. http://www.projectfloodlight.org/floodlight/.

[4] Mininet. http://mininet.org/overview/.

[5] Netconf. http://www.netconfcentral.org/netconf_docs.

[6] Nfv white paper. https://portal.etsi.org/nfv/nfv_white_paper.pdf.

[7] Nox. http://www.noxrepo.org/nox/about-nox/.

[8] O-ran. https://www.o-ran.org/.

[9] Of-config. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf.

[10] Onos. http://onosproject.org/software/.

[11] Oshi home page. URL http://netgroup.uniroma2.it/twiki/bin/view/Oshi.

[12] Open vswitch database management protocol. URL https://tools.ietf.org/html/rfc7047.

[13] Opendaylight. http://www.opendaylight.org/software/, .

[14] Openflow. https://www.opennetworking.org/sdn-resources/openflow, .

[15] Openflow switch specification. https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, .

[16] Openvswitch home page. http://openvswitch.org/, .

[17] Pox. http://www.noxrepo.org/pox/about-pox/.

[18] Restconf. https://tools.ietf.org/html/draft-ietf-netconf-restconf-05.

[19] Ryu. http://osrg.github.io/ryu/.

[20] Small cell virtualization functional splits and use cases, small cell forum release 5.1. http://scf.io/en/documents/159_-_Small_Cell_Virtualization_Functional_Splits_and_Use_Cases.php.

[21] Linux traffic control. URL https://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html.

[22] iperf - the ultimate speed test tool for tcp, udp and sctp. URL https://www.iperf.fr/.

[23] netem - network emulation. URL https://www.wiki.linuxfoundation.org/networking/netem.

[24] Tclink in mininet python api reference manual. URL http://www.mininet.org/api/classmininet_1_1link_1_1TCLink.html.

[25] Wireshark. URL https://www.wireshark.org/.

[26] Infonetics research, mobile backhaul equipment and services forecasts., March 2012.

[27] Small cell millimeter wave mesh backhaul. *InterDigital White Paper*, 2013.

[28] Mehran Abolhasan, Justin Lipman, Wei Ni, and Brett Hagelstein. Software-defined wireless networking: centralized, distributed, or hybrid? *IEEE Network*, 29(4):32–38, 2015.

[29] NGMN Alliance. Small cell backhaul requirements. *white paper, June*, 2012.

[30] Dejan Bojic, Eisaku Sasaki, Neda Cvijetic, Ting Wang, Junichiro Kuno, Johannes Lessmann, Stefan Schmid, Hiroyasu Ishii, and Shinya Nakamura. Advanced wireless and optical technologies for small-cell mobile backhaul with dynamic software-defined management. *IEEE Communications Magazine*, 51 (9):86–93, 2013.

[31] Wolfgang Braun and Michael Menth. Software-defined networking using openflow: Protocols, applications and architectural design choices. *Future Internet*, 6(2):302–336, 2014.

[32] Antonio Capone, Carmelo Cascone, Alessandro QT Nguyen, and Brunilde Sanso. Detour planning for fast and reliable failure recovery in sdn with openstate. In *Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the*, pages 25–32. IEEE, 2015.

[33] Daniel J Casey and Barry E Mullins. Sdn shim: Controlling legacy devices. In *2015 IEEE 40th Conference on Local Computer Networks (LCN)*, pages 169–172. IEEE, 2015.

[34] Luciano Jerez Chaves, Islene Calciolari Garcia, and Edmundo Roberto Mauro Madeira. Ofswitch13: Enhancing ns-3 with openflow 1.3 support. In *Proceedings of the Workshop on ns-3*, pages 33–40, 2016.

[35] Vijaya Durga Chemalamarri, Priyadarsi Nanda, and Karla Felix Navarro. Symphony-a controller architecture for hybrid software defined networks. In *2015 Fourth European Workshop on Software Defined Networks*, pages 55–60. IEEE, 2015.

[36] Andrea Detti, Claudio Pisa, Stefano Salsano, and Nicola Blefari-Melazzi. Wireless mesh software defined networks (wmsdn). In *2013 IEEE 9th international conference on wireless and mobile computing, networking and communications (WiMob)*, pages 89–95. IEEE, 2013.

[37] Open Networking Fundation. Software-defined networking: The new norm for networks. *ONF White Paper*, 2:2–6, 2012.

[38] Dana Hasan and Mohamed Othman. Efficient topology discovery in software defined networks: Revisited. *Procedia computer science*, 116:539–547, 2017.

[39] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.

[40] Cheng Jin, Cristian Lumezanu, Qiang Xu, Zhi-Li Zhang, and Guofei Jiang. Telekinesis: Controlling legacy switch routing with openflow in hybrid networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 20. ACM, 2015.

[41] Naga Katta, Haoyu Zhang, Michael Freedman, and Jennifer Rexford. Ravana: Controller fault-tolerance in software-defined networking. In *Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research*, page 4. ACM, 2015.

[42] D. Katz and D. Ward. Bidirectional forwarding detection (bfd), rfc 5880 (proposed standard). `https://tools.ietf.org/html/rfc5880`, Jun. 2010.

[43] James Kempf, Elisa Bellagamba, András Kern, David Jocha, Attila Takács, and Pontus Sköldström. Scalable fault management for openflow. In *Communications (ICC), 2012 IEEE international conference on*, pages 6606–6610. IEEE, 2012.

[44] Hyojoon Kim, Mike Schlansker, Jose Renato Santos, Jean Tourrilhes, Yoshio Turner, and Nick Feamster. Coronet: Fault tolerance for software defined networks. In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–2. IEEE, 2012.

[45] Maciej Kuźniar, Peter Perešíni, Nedeljko Vasić, Marco Canini, and Dejan Kostić. Automatic failure recovery for software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 159–160. ACM, 2013.

[46] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010.

[47] Hui Lu, Nipun Arora, Hui Zhang, Cristian Lumezanu, Junghwan Rhee, and Guofei Jiang. Hybnet: Network manager for a hybrid network infrastructure.

In *Proceedings of the Industrial Track of the 13th ACM/IFIP/USENIX International Middleware Conference*, page 6. ACM, 2013.

[48] Michael Markovitch and Stefan Schmid. Shear: A highly available and flexible network architecture marrying distributed and logically centralized control planes. In *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, pages 78–89. IEEE, 2015.

[49] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[50] Péter Megyesi, Alessio Botta, Giuseppe Aceto, Antonio Pescapé, and Sándor Molnár. Challenges and solution for measuring available bandwidth in software defined networks. *Computer Communications*, 99:48–61, 2017.

[51] John Moy. Ospf version 2. Technical report, 1997.

[52] Leonardo Ochoa Aday, Cristina Cervelló Pastor, and Adriana Fernández Fernández. Current trends of topology discovery in openflow-based software defined networks. 2015.

[53] Mohammed Osman and Josep Mangues-Bafalluy. Hybrid sdn performance: Switching between centralized and distributed modes under unreliable control communication channels. *Journal of Sensor and Actuator Networks*, 10(3), 2021. ISSN 2224-2708. doi: 10.3390/jsan10030057. URL `https://www.mdpi.com/2224-2708/10/3/57`.

[54] Mohammed Osman, José Núñez-Martínez, and Josep Mangues-Bafalluy. Hybrid sdn: Evaluation of the impact of an unreliable control channel. In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 242–246, 2017. doi: 10.1109/NFV-SDN.2017.8169866.

[55] Farzaneh Pakzad, Marius Portmann, Wee Lum Tan, and Jadwiga Indulska. Efficient topology discovery in software defined networks. In *2014 8th international conference on signal processing and communication systems (ICSPCS)*, pages 1–8. IEEE, 2014.

[56] Zhouyue Pi and Farooq Khan. An introduction to millimeter-wave mobile broadband systems. *IEEE communications magazine*, 49(6):101–107, 2011.

[57] Mark Reitblatt, Marco Canini, Arjun Guha, and Nate Foster. Fattire: Declarative fault tolerance for software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 109–114. ACM, 2013.

[58] Federica Rinaldi, Helka-Liina Maattanen, Johan Torsner, Sara Pizzi, Sergey Andreev, Antonio Iera, Yevgeni Koucheryavy, and Giuseppe Araniti. Non-terrestrial networks in 5g & beyond: A survey. *IEEE Access*, 8:165178–165200, 2020.

[59] Stefano Salsano, Pier Luigi Ventre, Luca Prete, Giuseppe Siracusano, Matteo Gerola, and Elio Salvadori. Oshi-open source hybrid ip/sdn networking (and its emulation on mininet and on distributed sdn testbeds). In *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, pages 13–18. IEEE, 2014.

[60] Stefano Salsano, Pier Luigi Ventre, Francesco Lombardo, Giuseppe Siracusano, Matteo Gerola, Elio Salvadori, Michele Santuari, Mauro Campanella, and Luca Prete. Hybrid ip/sdn networking: open implementation and experiment management tools. *IEEE Transactions on Network and Service Management*, 13 (1):138–153, 2016.

[61] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. Enabling fast failure recovery in openflow networks. In *2011 8th International Workshop on the Design of Reliable Communication Networks (DRCN 2011)*, pages 164–171. IEEE, 2011.

[62] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. Openflow: Meeting carrier-grade recovery requirements. *Computer Communications*, 36(6):656–665, 2013.

[63] Amin Tootoonchian and Yashar Ganjali. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pages 3–3, 2010.

[64] Jean-Philippe Vasseur, Mario Pickavet, and Piet Demeester. *Network recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*. Elsevier, 2004.

[65] Stefano Vissicchio, Laurent Vanbever, and Olivier Bonaventure. Opportunities and research challenges of hybrid software defined networks. *ACM SIGCOMM Computer Communication Review*, 44(2):70–75, 2014.

[66] Feng Wang, Heyu Wang, Baohua Lei, and Wenting Ma. A research on high-performance sdn controller. In *2014 International Conference on Cloud Computing and Big Data*, pages 168–174. IEEE, 2014.

[67] Shie-Yuan Wang. Comparison of sdn openflow network simulator and emulators: Estinet vs. mininet. In *2014 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2014. doi: 10.1109/ISCC.2014.6912609.

[68] Shie Yuan Wang and HT Kung. A simple methodology for constructing extensible and high-fidelity tcp/ip network simulators. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 3, pages 1134–1143. IEEE, 1999.

[69] Shie-Yuan Wang, Chih-Liang Chou, and Chun-Ming Yang. Estinet openflow network simulator and emulator. *IEEE Communications Magazine*, 51(9):110–117, 2013. doi: 10.1109/MCOM.2013.6588659.

[70] Takuma Watanabe, Takuya Omizo, Toyokazu Akiyama, and Katsuyoshi Iida. Resilientflow: Deployments of distributed control channel maintenance modules to recover sdn from unexpected failures. In *Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the*, pages 211–218. IEEE, 2015.

[71] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1):27–51, 2015.

[72] Liehuang Zhu, Md Monjurul Karim, Kashif Sharif, Fan Li, Xiaojiang Du, and Mohsen Guizani. Sdn controllers: Benchmarking & performance evaluation. *arXiv preprint arXiv:1902.04491*, 2019.