Universitat Politécnica de Catalunya

Network Engineering Department

PhD Thesis

Scalability and Robustness of the control plane in Software-Defined Networking (SDN)

Author: **Yury Andrea Jiménez Agudelo**

Advisor: **Cristina Cervelló i Pastor**

Barcelona, May 2016

Universitat Politécnica de Catalunya
Network Engineering Department

Contents

List of Figures

List of Tables

Part I

Network Management Fundamentals and State of the Art

Chapter 1

Introduction

First computer networks began in a laboratory in the early 1960s with the idea of sending data between two computers where data was divided and encapsulated into packets. This network grew over the next few years and began to support applications such as file transfers and electronic mails. A few years after, in 1972, the idea of connecting independent networks each based on a different technology resulted in the appearance of Transmission Control Protocol (TCP) and Internet Protocol (IP). The former provides reliable communication across different networks and the latter deals with the delivery of packets. The goal of the Internet was simply to carry the packets from source to destination, where simple routing algorithms, installed on each network element, computed the routing tables to achieve that goal. In this model, the control plane merely needed to manage an end-to-end communication, where all traffic was treated in exactly the same way: the best effort service.

The growth of computer networks. The simplicity of Internet design has led to enormous growth and innovation. In recent decades several network technologies, services and applications have appeared, which demand specific network requirements for their correct operation. The growth of computer networks together with the use of increasingly sophisticated user's terminals, has led to the networks to evolve from:

- a static academic network managed by a centralized entity evolving into a network operated by numerous providers where routing protocols operate in a distributed way and each operator manages its network based on a set of policies, and
- a network system designed to provide basic end-to-end communication between two simple machines (in a topology known in advance) evolving into a heterogenous networking system (wire and wireless) that

delivers a range of specific service requirements far more sophisticated than best-effort packet delivery (in a unknown network topology).

The complexity of heterogeneous network configuration. In traditional networks, operators are responsible for providing a network configuration sufficiently robust to deal with a wide range of network events and applications. To achieve this is incredibly difficult because: i) the state of the networks can change continuously and today's networks do not provide a mechanism to automatically respond to the wide range of events that may occur and ii) the static nature of current network devices does not permit detailed control-plane configuration, given that the hardware and software are provided by the manufacturer and can not be customized. This network rigidity has led to:

- the development of new specific protocols and mechanisms on top of TCP/IP to transport the enormous amounts of data in an efficient, robust and flexible manner,
- the manual installation of policies to manage anomalies during network changes,
- the installation of several network components called middleboxes that provide specific functions to alleviate the lack of in-path functionalities within the network.

This is the basis of the current, present-day Internet and its architecture, that has grown in an evolutionary fashion from experimental beginnings, rather than from a deliberate strategy.

The unpredictable network growth in terms of size and heterogeneity, has exposed a number of fundamental complexities in the current architecture, such as:

- networking devices usually support a handful of commands and configurations based on a specific embedded operating system (OS) [1], and as a result, new software can not be installed on forwarding devices because of incompatible hardware or because the currently available software is incapable of managing all the hardware capabilities,
- manual configuration of control functions on network devices that may lead to misconfigurations [2]. For instance, more than 1000 configuration errors have been observed in BGP routers [2]. Undesired network

behaviour may result from a simple misconfigured device that may compromise the network operation for hours [3], and

- the vertical integration of middleboxes makes it difficult for operators to specify high-level network-wide policies using current technologies.

Network management requires more intelligent and efficient management systems to coordinate thousands of network elements and applications, the high demand on network performance and growing configuration complexity [4], [5]. In recent decades, several approaches have been introduced in order to improve the network management, such as: MPLS, virtualization and programmable networks. These latter networks have been proposed as a way of facilitating network evolution. In particular, Software Defined Networking (SDN), a networking paradigm focused on allowing software developers to rely on network resources in an easy manner, unifying the state network distribution and a general-purpose technique to manage any type of network in an transparent manner.

In SDN, network intelligence is logically centralized in software-based controllers (the control plane), and network devices become simple packet forwarding devices (the data plane) that can be programmed via an open interface. By decoupling the control and data planes, network devices can be easily programmed and reconfigured, allowing the behaviour of different types of network devices to be unified.

Several SDN architectures have been proposed to handle current and future network services [6], [7], [8], [9]. However, there are still important research challenges to be addressed in SDN. Some of these current challenges are related to: i) SDN scalability as control is centralized, ii) control plane robustness as any failure can lead to switches to be disconnected from the controller, iii) consistency of network information as wrong decisions can be made affecting network performance and iv) security as controllers can be attacked. The purpose of this manuscript is to address the first three of the aforementioned problems.

1.1 Goals of the thesis

The main objective of this thesis is to design network mechanisms to manage the scalability and robustness of the control plane in SDNs. Three specific goals are defined:

G.1 Select the controller placements in SDN networks.

- **G.1.1** Classify the existing approaches to find the controller placement in SDNs.
- **G.1.2** Design an algorithm to find both the number of controllers and their location on the network.
- **G.1.3** Implement and evaluate the solution proposed, contrasted with other existing solutions.

G.2 Define a mechanism that allows the robustness of control plane in SDNs to be determined.

- **G.2.1** Classify the existing approaches to define the control plane robustness.
- **G.2.2** Design and implement a metric to define the robustness of a control plane.

G.3 Design a protocol to discover the network topology in SDNs.

- **G.3.1** Classify the existing approaches to discover the network topology in SDNs and maintain it updated.
- **G.3.2** Design a protocol to discover the network topology in SDNs and a set of mechanisms to maintain the network topology updated.
- **G.3.3** Implement and evaluation of the discovery network protocol .

1.2 Structure and Overview

This manuscript is divided in two Parts. The main concepts used throughout this manuscript as well as the most relevant approaches proposed in each of the issues covered here are presented in Part I as follows. Chapter 2 presents a brief overview of the notion of SDNs and OpenFlow. Chapter 3 introduces the main solutions for each one of the following aspects: i) the controller placement problem, ii) control plane robustness and iii) the discovery of the network topology by controllers.

Part II presents the contributions of this thesis. Chapter 4 describes the implications of controller placement in the SDN network performance and proposes an algorithm to find the number and placement of controllers to satisfy a specific requirement. Chapter 5 presents a robustness metric to measure the resilience of the control plane in SDNs. Chapter 6 presents a protocol to discover the network topology by controllers and also introduces a set of network mechanisms to maintain the network topology consistency on the control plane. Finally, Chapter 7 presents the main conclusions of this thesis.

Chapter 2

Network Management Fundamentals

As a starting point for this manuscript, this chapter presents the networking concepts and technologies that will be used in the following chapters. Firstly, SDN architecture and the most relevant forwarding network protocols for SDNs are introduced. Secondly, existing path recovery mechanisms and traditional protocols used in current networks to discover network failures are presented.

2.1 Outline

This chapter presents the main network concepts of traditional and SDN networks. Section 2.2 presents the traditional network architecture: data plane, control plane and management plane. Section 2.3 introduces the concepts related to SDNs such as: its definition, architecture, planes and interfaces. Section 2.4 outlines the two most relevant communication protocols in SDNs, which are: OpenFlow and ForCES. Section 2.5 addresses the recovery path strategies in traditional networks. Section 2.6 presents two protocols implemented in traditional networks, these are BDF and LLDP. BDF detects network changes and LLDP updates the network topology information. Section 2.7 concludes the chapter.

2.2 Traditional networks

Computer networks are typically built from a large number of interconnected network devices such as routers, switches and numerous types of middleboxes. Traditional IP networks consist of three integrated planes on each network device. These are:

- **The data plane**: also called the forwarding plane. It corresponds to the set of physical and virtual network devices in the underlying network infrastructure that may include any forwarding devices such as routers, switches, virtual switches, wireless access points, among others. These network devices are responsible for forwarding data based on the information in their forwarding tables that are programmed by distributed routing protocols, such as BGP and OSPF.

- **The control plane**: this plane is defined by the set of protocols and algorithms that are used to compute and populate the routes programmed in the forwarding tables of network devices included in the data plane. These network mechanisms (protocols and algorithms) are implemented on the network devices, defining the network behaviour, that is, what and how data is forwarded on the network.
- **The management plane**: this plane includes the software services to remotely monitor and configure the control functionalities of the network devices in the data plane. Network managers are responsible for configuring manually the network policies in the control plane. They transform the high level-policies into low-level configuration commands, in response to a wide range of network events and application requirements.

In traditional networks, network policies are defined in the management plane, the control plane enforces these policies and the data plane forwards data based on these policies.

2.3 Sofware Defined-Networking (SDN)

2.3.1 Definition

According to Open Network Fundation (ONF), Software-Defined Networking (SDN) is an emerging architecture where the forwarding state in the data plane is managed by a remote control plane decoupled from the former. This network architecture has the following characteristics:

- ▷ the network control and forwarding functions are decoupled,
- ▷ the network control is executed by an centralized external entity,
- ▷ the network is programmable,
- ▷ forwarding decisions are flow-based,
- ▷ SDN has the capacity to the initialize, control, change, and manage network behavior dynamically via open interfaces.

▷ the underlying infrastructure is abstracted by applications and network services.

In SDN, data is forwarded through flows, which are defined as a sequence of packets between two end-points. All packets of a flow receive identical service policy treatment at the forwarding devices. The flow abstraction allows the behaviour of different types of network devices to be unified.

Thanks to all the aforementioned characteristics, SDN is claimed to be dynamic, manageable, cost-effective and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications.

2.3.2 Network elements

Basically two network elements are considered in SDN, these are controllers and forwarding elements.

The controller

Controllers in SDN are considered as the "brain" of the network, since they have a whole view of the network status and the forwarding logic required to forward data flows properly. This is possible through an open interface to the network, devices southbound from the controller. Through this interface the controller can: i) communicate and program the network devices, ii) execute basic functions, such as monitoring network devices and even gathering network statistics, among other functions.

The forwarding devices

This refers to all the entities, physical and virtual, that receives packets on its ports and performs one or more network functions on them. In SDN, network devices are often represented as basic forwarding hardware accessible via an open interface. However, different physical network equipments with an interface to the controller can be considered as forwarding devices in SDN.

2.3.3 SDN Architecture

An SDN architecture consists of three different planes and a set of interfaces that allows the communication between them. These planes are: control plane, data plane and application plane, as illustrated in Fig. 2.1.

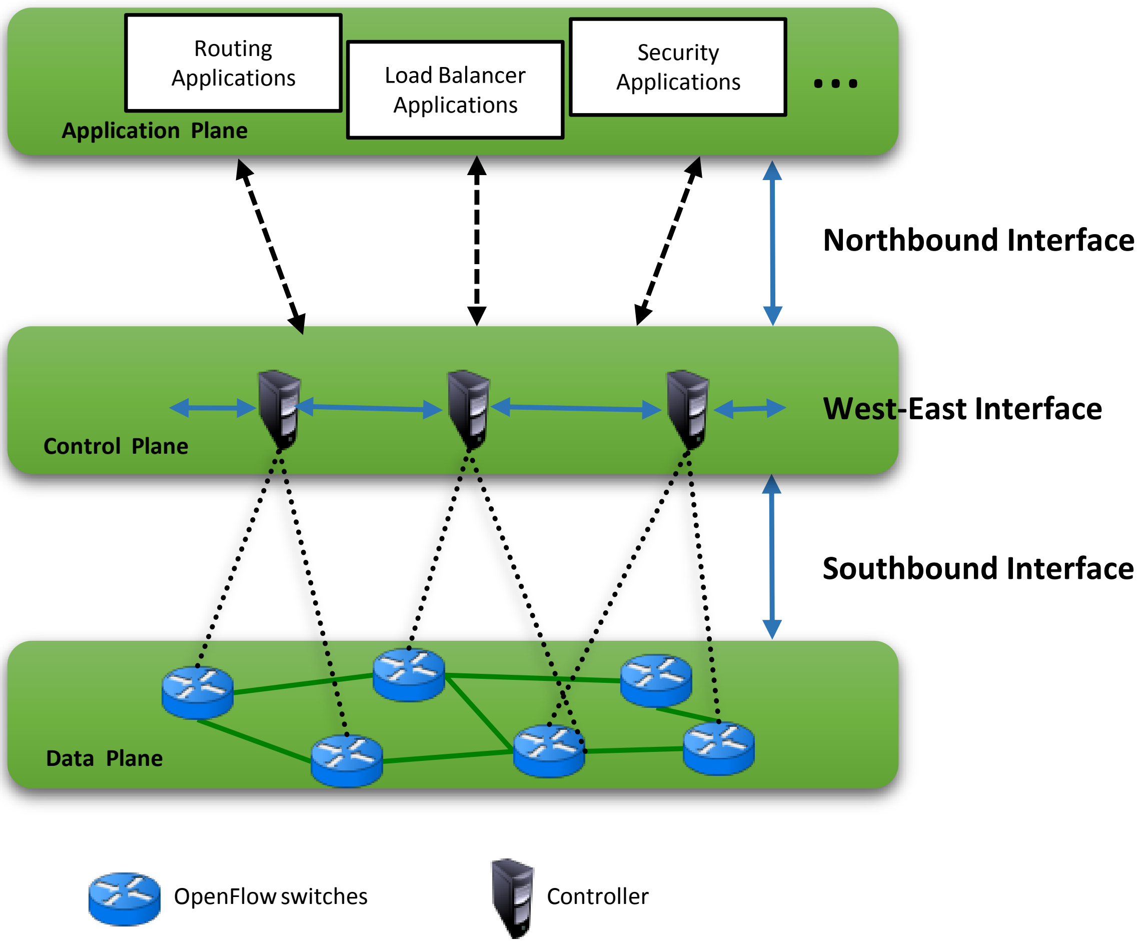


Figure 2.1: SDN network architecture.

Control Plane

Responsible for making decisions on how packets should be forwarded by one or more network devices and pushing such decisions down to the network devices for execution [10]. This plane consists of a centralized set of software-based SDN controllers that has an abstract view of the whole network infrastructure, enabling the network manager to apply customized policies across the network devices (through the southbound interface), based on the network topology or external service requests. The SDN controller is at the heart of the architecture. It is the intelligent entity that controls resources to deliver services [11]. Control-plane functionalities usually include:

- Topology discovery and maintenance
- Packet route selection and instantiation
- Path failover mechanisms
- install the forwarding rules on the forwarding tables based on the requested performance from the applications and the network security policy, and
- collect status information about the forwarding plane.

Data Plane

The data plane consists of physical and virtual forwarding devices that are accessible via the southbound interface through which controller and forwarding devices can communicate with each other. Forwarding devices can support basic functions like forwarding, but also other types of functions, such as: caching, transcoding and monitoring, among others.

Application Plane

The plane where applications and services that define network behavior reside. This plane contains network applications that interact with the controller to achieve a specific network function to fulfill the network operator needs, such as: quality of service, security, virtualization and traffic engineering functions. This plane allows the behaviour of desired control requirements to be specified on the abstract network view. For this purpose, the control plane provides an abstract view from the network to the application plane, while this is shared via a general interface called Northbound. This abstract view does not contain detailed connectivity information, but enough information for the applications to request and maintain connectivity.

In SDN, three interfaces are defined:

- **Southbound interface**: this is the interface between the control plane and the forwarding plane, through which controller and switches can communicate with each other.

- **Northbound interface**: it represents the software interface between the software modules of a controller and the network applications.
- **East-West interface**: through this interface controllers can communicate with each other.

SDN relies on three main abstractions [1], that allow the administrators to run the network as a whole, by way of an unified interface:

- Abstraction on the distributed state: network controllers get a global view of the network state, so there is no longer a needed for distributed algorithms on devices with partial network information.
- Abstraction on the forwarding model: forwarding hardware (in network switches) is decoupled from the network control plane (handled by network controllers).
- Abstraction on the specification of network operation: this allows a network application to express the desirable network behaviour without being responsible for implementing it.

2.4 Communication protocols

Different communication protocols have been defined in SDN to address the forwarding plane of the network elements on an SDN. Two of the most common are: OpenFlow [12] and the Forwarding and Control Element Separation ForCES [13].

ForCES and OpenFlow are similar in the following aspects:

- Both protocols consider the separation of the control plane from the data plane,
- Both protocols standardize information exchange between the control and data planes.

2.4.1 OpenFlow

OpenFlow is a communication protocol that enables the controller-to-switch communication in SDNs through the southbound interface [12], [14]. This protocol also defines a set of basic forwarding and management functions. The forwarding functions let programmers address the network operation by routing packets. The set of management functions can be used to control network features. For instance, switches can inform the controller when links go down or when receiving a packet for which there is no forwarding instruction.

This controller-to-switch protocol runs over either Transport Layer Security (TLS) or an unprotected TCP connection. OpenFlow provides software-based access to the forwarding tables that instruct forwarding devices how packets are forwarded. OpenFlow defines the following network components for its operation: OpenFlow switches, OpenFlow controllers and control channels.

OpenFlow switch: OpenFlow switches or forwarding devices have one or more flow tables and a group table. Forwarding devices perform packet lookups and take forwarding decisions based on flow tables and group tables. These *Flow tables* contain a list of flow entries, each of which determine how packets belonging to a flow will be forwarded. Flow entries consist of: a match field, a counter and actions.

- **Match field:** this is used to match incoming packets, entries in the match field contain either a specific value against which the corresponding parameter in the incoming packet is compared or a value indicating that the entry is not included in this flow's parameter set.

- **Counters:** these are used to collect statistics for a particular flow, such as the number of received packets and the duration of the flow.

- **Actions:** these are a set of instructions to be applied upon a match, which define how to handle matching packets. These actions describe packet forwarding, packet modification, and group table processing operations. For instance, actions can specify that the packet will be forwarded through a specified port.

Upon receiving a packet, a switch extracts the packet header field of the flow table entries. If a match is found, the switch applies the set of actions associated with the matched flow entry. If there is not a match, the action taken by the switches depends on the instructions defined in the table-miss flow entry. This table specifies a set of actions to be performed when a match is not found for a packet, such as: dropping the packet or re-forwarding the packet to the controller.

OpenFlow Controller: this is defined as an entity or server software that interacts with the OpenFlow switch using the OpenFlow protocol. This protocol connects controller software to network devices so that controllers can configure network devices and inform where to forward packets.

The OpenFlow channel: the control channel is the interface that connects each OpenFlow switch to a controller. Through this interface, the controllers configure and manage the switches, receive events from the switches, and send packets out the switches (to add, update, and delete flow entries in flow tables). The OpenFlow channel is usually instantiated as a single network connection. This may be encrypted using TLS, but may be run directly over TCP.

OpenFlow-Resilience

Openflow provides some strategies and mechanisms against controller, control channel and datapath failures.

Controller. In OpenFlow, three controller roles are defined; equal, slave and master. The default role of a controller is equal, in this role the controllers have access to the switches and can modify their state. A controller can request a change in its role from slave to master. In a slave role, the controllers only have access to read the switches and can only process port status messages. In the master role, the controller has complete access to the switches; switches ensure that a maximum of one controller can be in the master state.

Switches may be simultaneously connected to multiple controllers in equal state, multiple controllers in slave state, and at most one controller in master state, for reliability purposes. Following a master controller failure, a slave controller becomes master controller. This role change mechanism supports multiple controllers for failover, allowing the switches to be able to continue operating if their master controller fails.

Control channel. The OpenFlow channel may be composed of multiple network connections. In addition, OpenFlow specifies two simpler modes to deal with the loss of connectivity with the controller. In fail secure mode, the switches continue operating in OpenFlow mode until it reconnects to a controller. In fail standalone mode, the switches revert to using normal processing. In addition, in OpenFlow is also considered that a switch can have multiple control paths to the same or different controllers.

Data path protection. Group tables allow the network to configure fast failover mechanisms. The action buckets in the group tables provide the ability to define multiple forwarding actions. For instance, when the type of a group table is fast failover (FF), it means that the flows can be rerouted if a failure occurs in the datapath. Therefore, the controller must anticipate every possible failure and compute backup paths in a proactive manner. This group type enables the switches to change forwarding port without requiring a round trip to the controller. If no buckets are live, packets are dropped.

2.4.2 ForCEs

Forwarding and Control Element Separation (ForCES) is an open, programmable distributed network architecture, standardized by the IETF. This defines a new environment to build network devices that split them into units. ForCES also defines associated protocols to standardize information exchange between the control plane and the forwarding plane.

The ForCES network architecture defines two logical entities called the Forwarding Element (FE) and the Control Element (CE), and an API through which both network elements can communicate with each other by using the ForCES protocol.

- **Forwarding Element**: this is a logical entity that implements the ForCES Protocol. FEs use the underlying hardware to provide per-packet processing.
- **Control Element**: this element is a logical entity that implements the ForCES Protocol and uses it to instruct FEs on how to handle packets.

The protocol works based on a master-slave model, where the FEs are slaves and CEs are masters. Unlike OpenFlow, ForCES defines Logical Function Blocks (LFBs) that reside on the forwarding elements and are managed by the controllers. The LFBs enable the CEs to configure and manage the FEs.

The resources in FEs are instantiated in LFBs, each of which has a specific function in the packet forwarding chain. Multiple LFBs are interconnected via data paths to form an LFB topology in FE, so that the FE can carry out a complex process on packet forwarding. The management of LFBs includes the configuration and inquiry of LFB attributes, capabilities and events. ForCES protocol makes it possible for CE to dynamically manage the LFBs, such as to add, remove and modify some of them. Manageability of FEs by CEs provided in this model is at a much higher level than the manageability in present commercial routers aiming to provide the ability to configure new services in an open and simple way.

2.5 Recovery

Different kinds of disjoint control paths, partially or completely disjointed with respect to a defined control path between a switch-to-controller (called the primary control path), can be used to recover a switch-to-controller communication after it is broken. Different backup paths are described below. For this purpose, consider $P1$ as the primary control path between a switch s and a controller C and $P2$ as the backup control path.

Switch disjoint control path: a switch disjoint path $P2$ is the one that has only the source switch and controller in common to the primary path, $P1 \bigcap P2 = \{s, C\}$.

Link disjoint control path: a path between switch s and controller C is called the link disjoint ($P2$) if it has no common links with the primary path $P1$. The primary control path and its backup disjoint link path may contain the same intermediate switch(es), but can not share link resources.

Partial disjoint control path: this backup path may contain disjoint switch sub-paths to the controller, while the remaining intermediate switches and also some links can be the same as in the primary control path.

Disjoint control path to another controller: this control path has the same source switch but different controller destination as the primary path. This backup path may share switches and links with the primary path or may be completely disjointed from it ($P1 \bigcap P2 = \{s\}$).

Fig. 2.2(a) illustrates a physical network topology and each one of the remaining figures illustrates the primary control path and one of the different backup control paths described previously between the source switch (switch 1) and the controller (placed in node 6).

Fig. 2.2(b) shows a switch disjoint backup control path that can be configured in the case of any link/switch failure in the primary control path. Fig. 2.2(c) illustrates a link disjoint backup control path that provides protection for any link failures in the primary control path. Finally, Fig. 2.2(d) represents a backup partial disjoint path that has link (1, 2) and switch 2 in common to the primary control path.

Note that different backup control paths exist than those shown in Fig. 2.2. These can also be considered in order to protect the network against failures, improving network resilience. Ideally, the controller placements should be selected to provide the highest level of disjointness in paths of a control plane as it reduces the number of switches affected in the presence of failures, leading to a reduction in data loss.

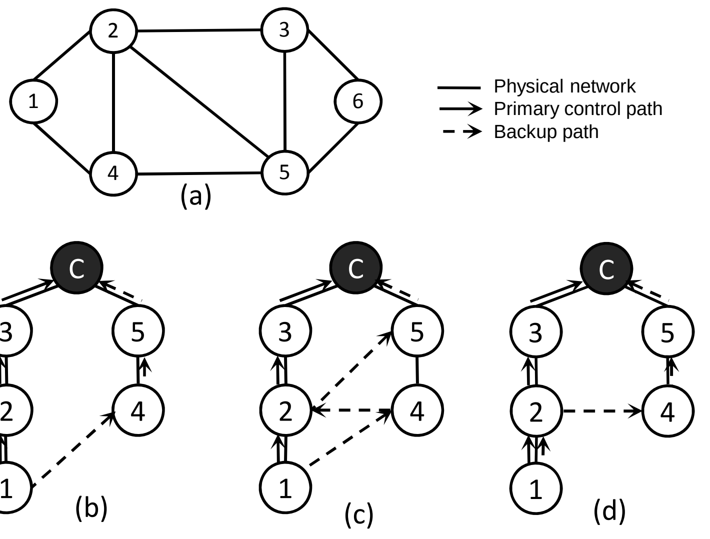


Figure 2.2: Representation of backup control paths.

2.6 Network Protocols

Two protocols used in traditional networks to detect network changes and discover the network topology are briefly described below.

2.6.1 Link Layer Discovery Protocol (LLDP)

Link Layer Discovery Protocol (LLDP) was standardized by the IEEE. This protocol runs over any data link layer network that allows the network devices (switches and routers): i) identify capabilities and their adjacent neighbours, and ii) listen to their neighbour devices. LLDP information can only be sent to and received from devices that are directly connected to each other through the same link. That is, announced information is not forwarded to other devices on the network. Network devices discover the status of their neighbours as they continually broadcast and listen to LLDP messages on each connection, discovering when a new device is added or removed. In this way, each network device can maintain its local network topology information updated.

Network devices identify neighbours according to the information (such as the MAC address) they receive in LLDP messages. This information is sent in packets called LLDP Data Units (LLDPDUs). The data sent and received via LLDPDUs is useful as network devices discover: i) the neighbour nodes of each device and ii) through which ports they connect to each other.

LLDP defines the following aspects:

- A set of common advertisement messages (Type Length Values),
- A protocol for transmitting and receiving advertisements,
- A method for storing the information that is contained within received advertisements.

LLDP is unidirectional, operating only in an advertising mode. Therefore, LLDP does not solicit information or monitor state changes between LLDP network devices.

2.6.2 The Bidirectional Forwarding Detection (BFD)

The Bidirectional Forwarding Detection (BFD) protocol detects failures in forwarding paths, operating on top of any data protocol (network plane, link plane, tunnels, etc.).

To detect failures BFD implements a control and echo message mechanism. This is a simple Hello protocol that transmits BFD packets periodically over a path between two pre-configured end-points to detect its status. If an end-point stops receiving BFD packets for long enough, it assumes that a network device in the monitored path has failed. Each network device transmits control messages with the current state of the monitored link or path. That is, a node receiving a control message, replies with an echo message containing its respective session status. A session is built up with a three-way handshake, after which frequent control messages confirm absence of a failure between the session end-points. Under some conditions, systems may negotiate not to send periodic BFD packets in order to reduce overhead.

2.7 Conclusions

The Internet has been very successful in providing us with the capacity to exchange information in the modern world. The Internet architecture has shown its robustness by supporting the continued growth of applications, services, users, and a large variety of network technologies over which it currently runs. However, nowadays, the strong coupling between control and data planes in traditional IP networks has made it difficult to add new network functionalities, since it implies the modification of the control plane on all network devices.

In contrast, SDN provides flexibility and scalability to the networks by decoupling the control and data planes. The logical centralization of the control plane (in SDN) offers several benefits: i) it simplifies the network management, reducing the possibility of errors when modifying the network policies, ii) SDNs can react to network changes, leading to more consistent and effective policy decisions and iii) it simplifies the development of more sophisticated networking functions, services and applications.

In short, SDN refers to the ability of software applications to program individual network devices dynamically and therefore control the behavior of the network as a whole [10]. OpenFlow-based SDN technologies enable the network to address the high-bandwidth, dynamic nature of today's applications, adapting the network resources, and significantly reducing operations and management complexity.

Chapter 3

Review of the state of the art

Aiming at confirming that the objectives of this thesis are not yet covered, in this chapter, a set of approaches that improves scalability, robustness and information consistency on the control plane are introduced. To manage an SDN network, controllers need to discover the network topology of the switches they are responsible for. This information is used by the controllers to plan, configure and monitor the end-to-end data paths. However, initially controllers do not have information about the network and the switches do not have a control path until they are set up. To solve this dilemma, current solutions assume that controllers discover the network topology through the traditional link discovery protocol LLDP. In addition, network topology information must be consistent, such that controllers can manage the network in an efficient way. Approaches that handle these topics are introduced in this chapter.

3.1 Outline

First section is devoted to describe the scalability, robustness and information consistency issues in SDN networks. Section 3.3 reviews the controller placement problem (CPP) approaches. In Section 3.4, different aspects of the controller placements approaches are described, while that in Section 3.6, different metrics used to select the controller placements are classified and described. Section 3.7, presents a classification of the controller placement approaches found in the literature, based on the aspects described in previous sections. Sections 3.8 and 3.9 introduce some approaches to discover the network topology and detect network failures, respectively. Finally, in Section 3.10, the conclusions of the chapter are presented.

3.2 Challenges in SDNs

Although the decoupling of control functions provides network flexibility, it also imposes several challenges in the control plane, related to the control plane scalability, resilience and data consistency among others issues [1], [2], [15], [16], [17], [18], [19], [20], [21], [22].

3.2.1 Scalability

When the SDN network scales up in the number of switches or traffic, it can increment drastically the load on the controllers that can become a bottleneck [1], [17], [18], [19], [23], [24] and [25]. This is because:

- whenever a large quantity of control messages arrive at a controller, the bandwidth, memory and processor of controllers are all potential bottlenecks,
- if the network has a large diameter, no matter where the controller is placed, some switches will encounter long flow setup latencies [16], [26],
- finally, since the system is bounded by the processing power of the controller, flow setup times can grow significantly as demand grows with the size of the network [17]. According to [27] and [28] the load of processing events is generally regarded as the most significant part of the total load on the controllers.

One of the most important reasons to distribute the network control is based on the fact that one controller alone may not have enough capacity to manage the whole network, and therefore it could become a bottleneck in terms of processing power, memory, or input/output bandwidth. As explained in [17], in a centralized and reactive SDN network, scalability problems can be caused by flow initiation overhead or resiliency to failures. In large networks with a distributed control plane, these scalability problems may also arise, since controllers not only have to process requests coming from switches it is responsible for, but also requests sent from other controllers. As in a centralized SDN network, in a distributed SDN network, controllers have limited capacity of memory and CPU that can be saturated if the size of a network grows or if the switch load is not distributed homogeneously between the controllers, [29], [30]. In addition, increasing network traffic lead to a reduction of the available bandwidth in the links used by the control channels, limiting the switch-to-controller communication. This situation is critical in a reactive approach, given that the controller can not do anything about the control link capacity as it can not treat messages faster than it receives them [31], [32].

Several approaches have been proposed to distribute the control plane across multiple controllers to improve the scalability of SDN, Kandoo [33], HyperFlow [34], and Onix [35], however, in these approaches the controller placement is not defined. Each one of those approaches distributes controller states differently. Kandoo distributes controller states by placing the controllers in two levels, a root controller and multiple local controllers. Local controllers respond to the events that do not depend on global network state, while the root controller takes actions that require a global network view. HyperFlow handles state distribution of the controllers through a publish/subscribe system based on the WheelFS distributed file system. Finally, controller state distribution in Onix is managed through a distributed hash table.

In general, controller placement approaches are not concerned with the controller scalability, because they assume that commercial controllers are scalable in terms of capacity (quantity of flows processed per second). However, it has been demonstrated that, controller overload and long propagation delays among controllers and controllers-switches can lead to a long response time of the controllers, affecting their ability to respond to network events in a very short time and reducing the reliability of communication [2], [16], [34].

3.2.2 Control plane resilience

In a centralized controller architecture as SDN, the network management is compromised in case of network failures (e.g., switch and controller failures). This is because, when a switch fails in the physical network, it not only affects the switch-to-controller communication, but also all the switches that include the failed switch in their control path.

In general, different stages can be implemented to ensure the resilience of networks [36], [37]. For instance, that includes redundant network paths. While such redundancy is desirable in both data and control paths, their effectiveness is dependant upon the ability of individual network devices to quickly detect failures and reroute traffic to an alternate path [38]. The detection times in existing protocols are typically greater than one second or much longer, for some applications, this is too long to be useful.

After detecting a failure or network degradation, the switch-to-controller communication should be re-established, if it is possible. The OpenFlow standard defines that a switch can have a connection to different controllers in order to improve the network resilience. In the case of a controller failure, all the switches managed by it are disconnected from the control plane, and therefore, those switches can not forward flows destined to switches that are not configured in their flow table yet. The OpenFlow standard defines that a controller can also have backup controllers, called slave controllers.

In this context, the control plane resilience can be handled from the point of view of the controller placement. This is because, the controller location influences the control plane topology as described in [36], [39], [40] and [41]. Therefore, depending on the network topology and the controller placements, the switches disconnected from the controller can re-establish the connection to the control plane, if it is possible [42],[43], [44].

Given the criticality of the data today, networks are typically constructed with a high degree of redundancy. Therefore, in SDNs, not only the data plane (data paths) has to be protected against failures, but also the control plane (control paths).

3.2.3 Consistency of the network information over controllers

The failure detection time not only affects the response time of the network to re-establish the affected flows, but also the time switches can update the network status information on the control plane [16]. Keeping network information consistent is fundamental to make right routing decisions. That is because in SDNs, network decisions (e.g., backup paths, data paths, load balancing) are made based on the knowledge of the underlying network topology and resource utilization [45], [46]. In [47] and [48], the impact of the consistent global network view on network control was studied. Authors concluded that inconsistent information may significantly degrade the network performance.

3.3 Controller placement

The scalability and resilience of the control plane have been tackled from the point of view of the controller placement. In terms of scalability, the controller placements in a SDN network can be selected to guarantee that the control paths as well as the controllers have enough capacity to handle the traffic coming from the switches they manage, avoiding they become a bottleneck as in [49]. In terms of resilience, the controller placements can be selected to maximize the number of control paths protected [39], [40], or minimize the data loss [44], [50], [51].

The network status consistency is also affected by the controller placements. Regardless of the strategies used to manage the network state consistency, the connectivity among controllers determines the maximum time required to update information among them. It is called the window of inconsistency in [16]. This is a factor of delay bounded by the farthest controllers in the network and the load on controllers. In [47], the authors describe the impact of the physical distribution of the control plane for the performance and coordination of a control application logic.

Related to the location of the controllers in the network, several approaches have been proposed from the controller placement problem in SDNs formulated in 2012 [52]. Followed by Heller et al.[52], during the last years, several research works have been proposed on the controller placement problem [26], [27], [39], [40], [41], [42], [44], [50], [51], [53], [54], among other approaches. Approaches focused to virtualize the SDN resources have also been proposed, [55].

In general, these aforementioned approaches can be differentiated by the metric(s) to optimize, the strategy used for finding the controller placement, if approaches consider the network resources (controller, switch and link capacities) to the selection of the controller placement, and if controllers and/or switch can be reallocated dynamically. In addition, the controller placement approaches can also be classified according to the considerations or assumptions made to select the placement of the controllers in the network.

3.4 A controller placement taxonomy

The controller placement problem can be described by three further considerations that depend on the network requirements. First consideration is related to the reallocation of switches-to-controllers and controllers, which can be static (where switch-to-controller assignation do not change) or dynamic (where switches can dynamically change of controller). Second consideration to select the controller placement is related to the robustness, this is considered as a relevant characteristic that control planes should have [56]. Finally, approaches are also classified based on the considered assumptions in the selection of the controller placements.

3.4.1 Static vs. Dynamics

In SDNs, the controller placements are selected offline, that is, during the network design stage. However, the number of controllers as well as their placement can be dynamically adapted during the network operation based on the network changes.

Static

This approach does not contemplate the possibility of reallocating switches to controllers or activate a new controller in case of network changes. Several situations can lead to a need for reallocation switch-to-controllers:

- Controller failure: in this case, switches without an active connection to a controller can be managed by another controller,
- Change of network conditions: increment of network traffic can saturate a controller and it can become a bottleneck, affecting the network performance as all requests can take more time to be attended,
- Save energy: some controllers can be sub-utilized, these are controllers that manage a low traffic compared with other controllers in the network. To save energy, switches can be reallocated in order to turn off the controller with low load.

Dynamic

In the aforementioned situations, both the controller placements and switches-to-controllers can be reallocated dynamically to improve the network performance, for instance, minimize the flow setup time or minimize the controllers overhead as in [26] and [53].

In general, dynamic approaches try to reconfigure the distribution of switches-to-controllers in order to balance the load on the controllers while optimizing the utilization of the network resources (e.g., controller utilization). A static controller placement assignation achieves good results with regard to optimality, since that it does not consider dynamic network changes that can compromise the controller scalability and/or robustness. While a dynamic approach is better suited to deal with high network dynamicity, it tends to come at the cost of less optimal solutions.

In [26], authors propose a framework for dynamically adjusts the number of active controllers and assigns to each controller a set of switches according to network state while ensuring minimal flow setup time and communication overhead. In [57], authors propose a non-zero-sum game based on a dynamic controller placement technique. The controllers can be dynamically activated and disabled based on the traffic demand. The optimal number of controllers can be found by adding and rejecting controllers, ensuring the maximum utilization of controllers from a set of controllers in the network.

However, selecting a new controller and migrating the switches to that new controller, is non trivial. It is even more complicated when considering physical controllers, as frequent switching on/off consumes more time and power [57].

In [58] and [59], an architecture that provides the ability to dynamically adapt the controller resources is proposed. Switches are assigned to controllers based on the traffic conditions.

3.4.2 Robustness vs. Unprotected

Failures on the control plane are due to underlying physical causes that can be identified and are statistically independent [60]. The failure of a single switch (or link included in a control path) affects the switch-to-controller communication of all switches that share the failed switch/link. Therefore, in environments where fault-sensitive applications are deployed inside the networks, it must react efficiently to re-establish the affected data and control communications.

Robustness

In the context of a control plane, when a failure occurs, the switches that have lost communication to their controller should be capable of setting-up a backup control path to their controller. The standard communication protocol, OpenFlow, defines that a switch can have multiple disjoint paths to its controller or another one. To do that, the controller placements can be selected in order to maximize the control path redundance when node and/or link failures occur as in [39], [40], [51]. If there is no redundancy, the control plane is referred to be unprotected.

Unprotected

In this case, switches have only one control path to their controller. Therefore, if a failure occurs, the switch-to-controller communication is lost until its controller detects and configures an available control path to it. The time a communication is recovered depends on the network connectivity and the time it takes to the controller to discover the event and configure the control path over the affected switches, if it exists. This means that in case a switch fails, there is no guarantee that the switch-to-controller path can recover after a failure.

3.4.3 Assumptions vs. Real network conditions

The controller placements are selected based on a limited set of requirements, which are defined based on the network/service requirements. When several network requirements and network constraints are defined, the formulation becomes too complex, increasing the time resolution.

Assumptions

To simplify the complexity of the problem and therefore its resolution, some assumptions are considered in the selection of the controller placements. First assumption is related to the location of the controllers in the network. In this case, the controller placements are selected from a limited set of nodes instead of considering the entire network space. Second consideration is related to the number of controllers, which is assumed to be defined in advance.

Real network conditions

Approaches that do not make any assumption about those mentioned aspects, consider the real network conditions to select both, the controller placements and the number of controllers required to satisfy a specific objective.

3.5 Computing the optimal controller placements

This section presents the different objectives, strategies and the metrics that can be defined for selecting the controller placements in SDNs.

3.5.1 Main controller placement objectives

The controller placement problem consists of discovering both the optimal number of controllers and their placement in a network to satisfy a specific objective that can be formulated in terms of:

Metrics to provide QoS

The controller placements can be selected according to a set of quality of service requirements defined by the service provider. These QoS requirements are related to the switch-to-controller delay [42], [52], inter-controller delay [26], controller utility [27], among others. The location of the controllers in the network, can be also selected based on specific QoS metrics that guarantee the flow setup time and the response time of a request [26]. For instance, critical services as video or voice over IP have to satisfy specific QoS requirements to not be seriously affected.

Economical profit of the controller placement

From the point of view of network providers, a natural objective to select the controller placement would be to minimize the capital and operational expenditure. This objective is directly proportional to reduce the number of controllers in the network and use the controller resources in an efficient way. The number of controllers implemented in a network, directly impacts the capital and operational expenditures (CAPEX and OPEX). Therefore, it becomes necessary to find the optimal number of controllers in a network and the switch-to-controller assignation. In order to reach this goal, controller placement approaches should try to assign the maximum number of switches to each controller while satisfying a set of network requirements [43].

For instance, in [49] the objective is to minimize the cost of assigning a switch-to-controller, and in [54] the objective function is to minimize a given cost function, that seeks to maximize the resilience.

Resilience of the control plane

In terms of the control plane, resilience is related to find the controller placement that: i) minimize the data loss, by reducing the number of control paths affected by a network failure as in [44], [51] or ii) maximize the number of backup paths per each control path or at least, find the number of backup paths that satisfy defined resilience requirements [39], [40],[41] and [61].

3.5.2 Optimization strategies

The controller placement problem is NP-hard. Therefore, for large networks the time to find the optimal solution becomes very high. In this case, different strategies have been used to solve the controller placement problem in an efficient way.

Exact solutions

To find an optimal control placement solution, the problem has to be formulated by means of Integer Linear Programming (ILP). When using this strategy, network resources can be limited (e.g., controller, switches and links resources). In addition, other specific requirements such as resilience, switch-to-controller delay and inter-controller delay can be defined. However, the disadvantage of this strategy is the time it takes to find a solution, finding the controller placements in large networks can take a long time, as shown in [43] and [62].

Heuristic solutions

This strategy allows the controller placement problem to be solved in a more efficient time in comparison with ILP. Heuristic strategy does not find an optimal solution, but it finds a solution near the optimal, compromising optimality for short execution time. In general, the controller placement approaches are formulated through a heuristic algorithm.

Clustering solutions

Clustering is used in many disciplines and applications, it is an important tool that seeks to identify homogeneous groups of objects based on the values of their attributes. In the controller placement problem, clustering can be used with heuristic and exact solutions. In this case, the clustering approach is responsible for reducing the search space where a controller placement is sought, defining the set of feasible controller placements. Then, a heuristic or exact strategy can be used to select the better controller placement.

Strategies for hierarchical clustering generally fall into two types, agglomerative and partitive approaches [63]. In the case of an agglomerative approach, each node starts being its own cluster, this is the case of *k-center* and *k-Median*, and pairs of clusters are merged until a certain condition is satisfied. In a partitive approach, all the nodes are part of one cluster, while is recursively split until a terminating condition is met.

3.6 Controller placement metrics

Different metrics have been considered to select the controller placement. These can be classified according to the main network objective to select the controller placements, as described in Section 3.5.

3.6.1 Metric related to the network performance

These metrics evaluate the performance of the resulting control planes, which are related to: the switch-to-controller path and the switch allocation between controllers. These metrics can also be used to compare different controller placement approaches.

Stretch

This metric compares the path length of each switch to the controller placement found (control path) with the shortest path from the switch to the controller. A high stress might result in some control paths with an additional delay because the resources of the switches/links can be shared by multiple control paths. The more control paths use the same switch or link, the higher the impact regarding possible side effects. For instance, if multiples control paths use the same switch/link, when it fails all the control paths including this switch are disconnected from the controller.

Path length

The path length metric measures the number of hops between the controller and each switch that is managed by it. The longer a control path, the more resources are used and the node dependence increases (number of downstream nodes). This means that the path length has a direct impact on the cost of the resulting control plane. Depending on the links traffic, switches that have longer control paths to their controller can take a long communication delay, affecting the control plane performance. In long paths, when a network failure happens, all the downstream nodes to the failure node are disconnected from the controller.

Control path delay

This is the time it takes to a switch to communicate with its controller and vice versa. Therefore, this delay influences: i) the time switches announce events to their controller and ii) the time it takes to the controllers to configure their switches. Given the importance of this metric in the network performance, most of the approaches take into account this metric in the selection of the controller placement.

Inter-controller delay

This is the time it takes to each pair of controllers communicate with each other in a control plane. In a distributed control plane, controllers need to communicate among themselves to exchange control information and update their routing tables. These are essential operations to the network operation. To make right forwarding decisions, controllers must have the network topology information updated. In [16] and [47], the authors describe the impact of the physical distribution of the controllers for the performance and coordination of a control application logic. Regardless of the strategies used to manage the network state consistency, the connectivity among controllers determines the time required to update information among them. It is called the window of inconsistency in [27]. This is a factor of delay bounded by the farthest controllers in the network and the load on controllers. A high inter-controller delay can affect the network operation, due to the controllers can make wrong decisions if routing tables or control information is not updated in a short time. Nonetheless, multiple controllers may be used to reduce latency or increase fault tolerance [64]. In [65] an inter-controller communication mechanism was proposed.

Flow setup time

One of the key functions of an SDN controller is to establish flows. As such, some of the performance metrics associated with the controller scalability are the flow setup time and the number of flows per second that the controller can setup. In a reactive flow management approach, when a switch receives a flow that it does not know how to forward, the switches has to forward the flow to the controller for processing. The time associated with the flow setup time is the sum of the time it takes to send the packet from the switch to the controller, the processing time in the controller and the time it takes to send the configuration message back to the switch. Assuming that the controller is not a bottleneck and that it has information about the service required, the total setup time can only be affected by the distance between switch-controller. In [27], it has been demonstrated that a long flow setup time limits the network convergence time, and affects the controller ability to respond to network events in a minimal time that can degrade the application and service performance. In [26], authors propose a dynamic controller placement to minimize the flow setup time by dynamically changing the number of controllers and their locations.

3.6.2 Metric related to control plane scalability

One of the most important reasons to distribute the network control is based on the fact that one controller alone may not have enough capacity to manage the whole network, and therefore it could become a bottleneck in terms of processing power, memory, or input/output bandwidth [64]. As explained in [17], in a centralized and reactive SDN network, scalability problems can be caused by flow initiation overhead or resiliency to failures. In large networks with a distributed control plane, these scalability problems may also arise, since controllers not only have to process requests coming from switches it is responsible for, but also requests sent from other controllers. As in a centralized SDN network, in a distributed SDN network, controllers have limited capacity of memory and CPU that can be saturated if the size of a network grows or if the switch load is not distributed homogeneously between the controllers. In addition, increasing network traffic lead to a reduction of the available bandwidth in the links used by the control channels, limiting the communication with the control plane. This situation is critical in a reactive network, given that the controller can not do anything about the control link capacity as it can not treat messages faster than it receives them [31].

Therefore, in large scale WANs, the control plane topology can limit the availability, response time and convergence time on the network [27]. The reason is that, control applications require the ability to reprogram data plane state at very fine time-scales to satisfy network objectives in SDN. Therefore, selecting the controller placements to keep the flow setup time as low as possible is fundamental for an efficient network operation.

Link capacity

In SDNs, data and control paths share network resources. Therefore, the controller placements should be selected so that: i) all switches are managed, ii) the switch-to-controller control paths have enough capacity and, iii) the controllers have enough available resources. A congested link that is used by one or more control paths can add a switch-to-controller communication delay that affects the response time of a request and consequently the flow setup time [31].

Controller utilization

The controller utilization is measured by the number of flows managed by each controller. Each controller has an upper limit on the number of control messages and flows it can handle at any time. Therefore, there is a limited number of switches that can be managed by a controller. In [52], it was shown that one controller alone is capable of managing a complete network, in terms of controller capacity. However, the switch-to-controller delay can be very high, affecting the application performance. In large SDN networks, data center or enterprise networks that can manage a high traffic load, only one controller is not enough.

The placement of controllers should try to minimize the propagation latency, while the load of each controller should not exceed its capacity. The capacitated controller placement problem (CCPP) mentioned in [49] considered the problem of load controller capacity.

Number of controllers

The number of controllers in a control plane not only depends on the traffic managed by the network, but also on the other requirements that must be satisfied (e.g., control path delay, inter-controller delay). It has also been shown that the number of required controllers is more dependent on the topology than on network size [40].

Increasing the number of controllers in the network can significatively improve the resilience, as shown in [52], [43] and [54]. In [42] and [43], authors shown that there is a specific number of controllers that improve the network resilience, which depends on the network topology. It is also shown that, using more controllers than the necessary does not improve the network performance, but increases the solution cost and can difficult the controller communication . Authors in [66], propose a mathematical model to define the optimal number and locations of controllers. They also consider heterogeneity of controllers and their interconnections.

3.6.3 Metric related to control plane robustness

In traditional networks, the network robustness mainly depends on the node connectivity in the network [67], [68],[69]. In SDN, in addition to the network connectivity, the controller placements is also a decisive factor in the resulting control plane robustness [56]. The controller placements influence both i) the number of switches that can be disconnected or unprotected when a network failure occurs and ii) the number of switches that can reestablish the communication to the control plane through a backup path (or protected switches). Therefore, the controller placements can be selected to reduce the control plane resilience.

Number of backup paths

When a switch-to-controller communication is broken due to a link or switch failure, those switches that are disconnected from their controller can used a backup path to its controller or another one, as specified in the OpenFlow standard [70]. In the case of a controller failure, the switches managed by it can be managed by a backup controller [71].

Solutions based on connecting switches to several controllers in a cost-effective, have been proposed in order to maximize network robustness and therefore the network performance [72]. In [39] and [40], each switch must meet a reliability constraint so that operative paths towards any of the controllers it connects to, exists with at least a given probability. As a result the set of controllers that minimizes the associate cost are found. In [41] and [61], the authors select the controller placements that maximize the number of switches protected in the presence of a failure in the upstream switch. To achieve high availability, in [47] robustness is handled in terms of control plane connectivity.

Data Loss

In [44] and [51], a reliability metric called the expected percentage of control path loss is proposed. This metric is defined as the number of broken control paths due to network failures. In [50], a reliable metric that measures the expected percentage of valid control paths when network failures happen is defined. Therefore, these approaches do not consider backup paths, they find the controller placements that minimize the data loss due to network failures.

3.7 A classification of the controller placement approaches

Thus, all controller placement approaches proposed in the literature can be categorized according to: a) whether they are Static (S) or Dynamic (D), and b) whether node protection (Resilience (R)) or nodes are Unprotected (U) is considered. In addition, solutions can be categorized according to: i) the number of controllers (K), ii) the controller placements (CP) or iii) both the number of controllers and their location ($K-CP$). If an approach considers Real network Conditions, it is represented as (RC). In Table 3.1, the controller placement approaches are classified according to the aforementioned aspects by using the following syntax:

[S—D]/[R—U]/[K/CP/K-CP/RC].

The first character denotes, whether the approach is is Static or Dynamic. Likewise, the second character denotes whether the controller placement approach considers Resilience to the control plane or if the control plane is Unprotected. Finally, the third character denotes whether an approach assumes any of the following aspects to select the controller placements: i) the number of controllers defined by **K**, ii) the controller placements **CP** or iii) if an approach assumes both the number of controllers and their location, $k - CP$. Otherwise, if an approach does not assume any of the aforementioned aspects, it is denoted as **RC**. So, an approach denoted as S/R/RC will be a static and resilience approach that does not make any assumption about the number of controllers neither their placement in the network.

3.8 Network discovery

Different communication protocols have been defined in SDN to address the communication between switch-to-controller, such as OpenFlow [12] and the Forwarding and Control Element Separation ForCES [13]. These protocols do not define neither the allocation of switches to controllers nor the communication path between switch to controller. In general, to establish a switch-to-controller communication, the aforementioned communication protocols considers that:

- each switch has programmed the IP address of its controller and,
- they also have information about the TCP port number through with can communicate to their controller.

Therefore, controllers and switches can contact each other on the corresponding IP address and TCP port programmed in advance, through a Transport Layer Security (TLS) section. Consequently, the allocation of switches to controllers and the network topology information on control plane dependent on/require of human intervention.

To maintain the network information on the control plane updated, some protocols used in traditional networks with this purpose have been adopted for SDN networks. This is the case with the link discovery protocol (LLDP), a protocol used by the network devices to announce their identity , capacities and neighbours. LLDP has been adopted for SDN networks to maintain

Table 3.1: Classification of controller placement approaches.

Classification	Reference	Strategy	Contribution
S/U/K	[52]	Heuristic	Analyzes the impact of the controller locations on the control path delay.
	[73]	Clustering/ Heuristic	Finds the CPs that balance load on the controllers.
S/U/CP	[49]	ILP	Focuses on the capacitated CPP.
	[66]	ILP	Defines the type of controllers and connections between network elements.
	[74]	ILP	Proposes a multi-criteria optimization method applicable to the CPP.
S/R/K	[54]	Clustering/ Heuristic	Divides the CPP in two sub-problems, partitions network and assigns CPs.
	[41]	Heuristic	Finds the CPs that maximizes the control plane resilience.
	[62]	Heuristic	Analyzes the trade-off between control path delay Vs other metrics.
	[43]	ILP	Analyzes the trade-off between control path delay Vs other metrics.
	[44]	Heuristic	Introduces a reliability metric that measures the data loss.
S/R/CP	[40], [39]	ILP/ Heuristic	Finds the CPs that guarantee a reliability constraint.
	[61]	Heuristic	Improves network resiliency and finds CPs to maximize fail-over probability.
	[50]	Heuristic	Defines a resilience metric that minimizes the data loss.
S/R/K-CP	[51]	ILP	Characterizes the control plane reliability.
D/U/CP	[26]	ILP	Proposes a dynamic version of the CPP.
	[57]	Heuristic	Finds the optimal CPs by dynamic addition or deletion of controllers.
D/U/RC	[53]	Clustering/ Heuristic	Combines CP with a dynamic flow management algorithm.

the network topology information and the network resources information updated on the control plane. This mechanism is referred to as OpenFlow

Discovery Protocol (OFDP) [75].

Other approaches to discover the underlying network topology are based on LLDP-OpenFlow or LLDP-ForCES.

3.8.1 Network discovery based on OpenFLow protocol

In [46], authors propose a topology discovery mechanism based on OFDP protocol. The contribution of this approach, in comparison with the LLDP protocol is the reduction of the number of LLDP $Packet - Out$ messages a controller has to forward in order to discover the links between switches. For this purpose, the authors consider that:

- controllers have information about the switches in the network, but they do not have information about the links between them and,
- controllers have a control channel established with each switch in the network.

In this approach, the topology discovery is reduced to discover the links in the network. Given that an OpenFlow switch can not by itself send, receive and process LLDP messages, these messages are encapsulated in the control messages defined in OpenFlow. As part of the initial protocol handshake, controllers can request information to the switches through OpenFlow messages via their control path. This information includes the active ports and MAC addresses of the switches. While LLDP messages have information about, the port ID and time-to-live [46].

The information provided by all of these messages is used by the controllers to discover the connectivity between switches. When a controller receives a $Packet - In$, it discovers the ID of the switch and the ingress port via which the packet was received. From this information, and from information contained in the LDDP packets (e.g., Port ID, TLV, etc.), the controllers can infer that existence of links between two switches. The controllers have to execute this procedure periodically to maintain the network topology information updated.

A disadvantage of this approach is the high load that a controller has to manage, which depends on the number of LLDP $Packet - Out$ messages the controller needs to send (that is proportional to the number of ports in the network) and the number of LLDP $Packet - In$ messages it receives (it depends on the network topology). Besides, this approach only informs about the link status.

In order to reduce the number of LLDP *Packet − Out* messages that a controller has to send, authors propose that SDN switches are capable of re-writing the LLDP port ID according to the port the packet is being sent out in. Consequently, a controller is not required to forward as many messages as there are ports per switch, as each switch can re-forward the message.

3.8.2 Network discovery based on ForCES protocol

An approach to discover the network topology based on ForCES is proposed in [45]. Authors propose a generic model for extracting the topology information directly from the network devices, defining the switch-to-controller paths by configuring the Logical Function Blocks (LFBs). In ForCES, switches have Logical Function Blocks or LFBs, which are configured by the controllers, such that switches are capable of receiving, transmitting and modifying packets. Each switch runs the traditional LLDP protocol periodically to maintain the local neighbour table updated. In the occurrence of an event (e.g,. a new switch or a neighbour switch is broken), the switch announces it to the controller, updating the network topology information.

3.9 Network topology consistency

In an operative network, topological changes can happen anywhere and anytime, which must be detected and announced to the respective controllers to maintain the network topology information updated on the control plane. Regarding the importance of failure detection in SDN networks, there is not a detailed mechanism of failure detection on any of the communication protocols defined for SDN networks (e.g., OpenFlow specifications); this is still an open issue. In this section, a description of some proposed detection failure mechanisms in SDN networks are presented.

The speed at which failures are detected directly affects the performance of both the control and data planes. This is because, in the context of the control plane, it is desirable to maintain the consistency in network information at all times to avoid making routing decisions that involve disabled network resources. At the same time, in the context of the data plane, the time a failure is detected affects the communication recovery time.

3.9.1 Fault detection in SDN networks

Whereas traditional network protocols rely on their distributed algorithm running on the network switches to detect and react in the presence of failures, in SDN networks, as a consequence of the separation between the control and data planes, these functions are no longer executed by the data plane. Instead, in SDN networks, the operator through the controllers has to explicitly define the switch behaviour after a failure occurs in the network. Despite this fact, failure detection mechanisms can be considered, in general, to be executed by controllers. However, due to the resulting high overloading on the controller, it also has been considered that SDN switches can execute this function.

Detection failure by controllers

Different approaches have been considered to discovery the network topology, but few of them address the detection of failures. For instance, in [46], authors propose a mechanism to discover the links between switches by encapsuling the OpenFlow packets in LLDP packets. However, this approach can also be used to discover network failures through OpenFlow messages. While that in [76], authors propose the creation of a ring topology that includes all the switches and links of the network through which probe packet are forwarded to monitor the network status.

The basic idea behind failure detection approaches executed by controllers consists of forwarding probe packets through the end-to-end paths being monitored, which require a response from the switches; an unanswered message implies that a failure has happened. In this model, when a failure occurs, it is hard to detect its location with precision. To detect the location of a failure, controllers can forward probe packets to each one of the switches in the monitored path, such that each message sent has to be answered before forwarding the message to the next switch in the path.

When multiple and simultaneous failures happen, the detection of a specific failure becomes complex as the network may be partitioned. This simple detection mechanism is neither time efficient nor scalable as the controller has to send an increasing number of messages directly proportional to the number of switches monitored to check if the network elements are operational.

In addition, the speed of this failure detection is slow, as the failure detection depends on:

- the path length and,
- the rate at which messages are sent.

This detection model imposes a high load on the controllers that can affect the performance of the original tasks it was designed for. Hence, the controller must be able to handle and react to millions of monitoring packets per second just to monitor the state of the network. This not only leads to a high load on the controller, but also on the control paths.

The approach proposed in [76] is designed to verify topology connectivity and locate link failures. For this purpose, a single closed path that includes each edge in the network is created. This defines a logical ring topology where all switches in the network are included. In order to define the cycle, each link can be used twice, if necessary. After the ring topology is built, each controller in its network domain proceeds to install flows in each switch, configuring the monitoring cycle.

To verify the network topology connectivity, the controllers inject probe packets that are forwarded along the cycle, unless there is another rule specified. If there is no failure in the ring or segment inspected, the probe packets must be forwarded back to the controller. A bidirectional logical ring topology is created to locate an arbitrary link failure.

When a probe packet fails to return to a controller, it defines different segments of the ring to be inspected (by sending probe packets) and detects the location of a failure. Depending on which messages are received back, a controller can detect the segment where the failed link is located. Each segment of the logical ring must be inspected in order to locate the failure. For this purpose, controllers have to configure the forwarding tables of switches involved in each segment.

This approach can not detect and locate more than a single link at the same time. The reason for this is that, in the case of multiple failures, the monitored path/cycle is broken in different places impeding the forwarding of control messages to the controller. Besides, this mechanism can not be implemented for any kind of network, since there are network topologies for which it is not possible to built a ring/cycle.

Some disadvantage of the aforementioned approaches are:

- the performance of controller-based detection approaches is poor as it can take a long time to discover a failure. In [76] this time is proportional to twice the delay on each monitored path,
- overload the controller,
- controllers are unable to differentiate between switch and link failures.

Alternative approaches to detect failures in SDN networks consider that the fault detection function should remain directly implemented by the switches.

Detection failure by switches

Different switch-based fault detection approaches have been proposed to improve the detection time without overloading the controller, such as [38] and [77]. The objective of these approaches is to detect as soon as possible a network failure, such that data communication can be recovered as fast as possible. To ensure this, both approaches consider that all switches have backup paths to the controller, which can be activated as soon as a failure is detected.

In [77], authors propose a set of functions to provide reliability to the network, based on the idea of re-establishing a communication when a path failure occurs. Within these functions, authors propose the implementation of a monitoring function on the OpenFlow switches to monitor a data path without involving the controller. The end-source of a monitored flow forwards periodically and interleaves probe packets into the data flow running along the data path. The switches along the path are able to receive and forward these packets, as they have static flow table entries that allow the packets to be processed accordingly. However, the switch responsible for detecting a failure in the data path is the destination switch that extracts the probe packets to check the path status. The absence of probe packets during the data flow transmission, indicates that there is a failure in the path. In this case, the destination switch changes to a backup control path to re-establish the communication. The detection fault time, in this approach, depends on the data path length and the interval between probe packets.

Authors in [38] use the Bidirectional Forwarding Detection (BFD) protocol to detect the path state between two pre-configured end-points. BFD is a simple Hello protocol that transmits BFD packets periodically between the end points. If an end point stops receiving the BFD packets, it assumes that the path has failed. Actions are defined in the Action Buckets, which activate the protected path. In order to reduce the fault detection time, authors in [38] considered that switches can be configured to monitor their adjacent neighbour switches in an active way, instead of monitoring only the communication between end-points. For this purpose, a BFD session is configured between each pair of adjacent switches in a data path. If periodical messaging over the session fails, a switch assumes the link to its adjacent switch is lost, updating the Action Bucket status and sending this information to the controller, which configures a new path. The overhead on the network itself remains low as it does not involve the controller until the failure is localized.

In general, authors in the aforementioned switch-based approaches consider that the Group Tables extend OpenFlow configuration rules allowing advanced forwarding and monitoring functions at switch level [38]. For instance, the Fast Failover Group Table can be configured to monitor the status of ports and interfaces, independent of the controller.

From the approaches introduced, it is clear that fault detection approaches executed by switches reduce the number of messages used to monitor the data paths, reducing the time a failure is detected without posing a processing load on the controller compared to when failures are detected by the controllers. Note that, the detection failure mechanisms are designed to monitor only data paths.

3.10 Conclusions

In this chapter, controller placement approaches and some mechanisms to discover the network topology and keep it updated are introduced. The controller placement problem is NP-hard, therefore, each approach finds the controller placements based on a set of limited constraints. Several network metrics and algorithms approaching this problem have been discussed in the literature, so far.

In this chapter, a set of general aspects found in the controller placement solutions was described along three distinct aspects: i) static vs. dynamic, ii) protected vs. unprotected and iii) real network conditions vs. assumptions. In addition, the different metrics used to select the controller placements were described. This information was used to create a classification of the controller placement approaches found in the literature. Finally, solutions that discover the network topology and detect network changes and failures were described.

Part II

Contributions to the SDN management

Chapter 4

Discovering controller placement in SDN networks

The software-defined networking (SDN) advocates a centralized network control global view. Large SDN networks may consist of multiple controllers in one or different controller domains, distributing the network management between them. Each controller has a logically centralized but physically distributed vision of the network. In this context, a key challenge faced by providers is to define a scalable control plane that exploits the benefits of SDN, given that design aspects in the control plane such as load distribution between controllers, propagation delays among controllers and delay between controller-to-switches can lead to a long response time of the controllers, affecting their ability to respond rapidly to network events and reducing the reliability of communication. In this chapter, a new solution to the controller placement problem called $k - critical$ is proposed, that determines where to place controllers and how many controllers need to be installed in a network in order to create a robust control plane while satisfying a controller-to-switch delay. $K - critical$ was published in [78] and [42].

4.1 Outline

Section 4.2 describes the implications of the controller placement on the SDN network performance. In Section 4.3, controller placement approaches found in the literature are classified based on the criteria used to select the controller placements. Section 4.4 describes $k - critical$ in detail, the concepts defined and the process used to select the controller placements for any kind of network, and in Section 4.5 a scalable version for $k - critical$ is defined to find controller placement in big networks in an efficient time. Section 4.6 presents a comparison of $k - critical$ with other controller placement approaches based on clustering. Finally, Section 4.7 concludes the chapter.

4.2 Controller placements and its implications

The control plane in SDNs may take any topology, including that of a star, where a single controller manages the network; a hierarchical architecture where controllers are connected creating a mesh network; or even a ring composed of a set of controllers that are managed using a distributed hash table. Each one of these control plane topologies has implicit limitations. However, regardless of the topology, there are some general aspects, such as the delay both among controller-switches and delay among controllers (as described in Section 3.6), the controller scalability, that affect the ability of the controllers to respond to network events [50].

The controller placement influences every aspect of a decoupled control plane, from state distribution options to fault tolerance, to performance metrics and network reliability. The importance of controller placement is well established [17], [52], [51], [79]. In [52] the authors show the implications between the number of controllers and the delay communication between switches-to-controller and between controllers, and in [51] the authors develop several placement algorithms to make placement decisions to maximize the reliability of SDN. In [79] the authors describe the implications of the local network view of the controllers and discuss the design of a distributed control plane.

4.3 Shortcomings of existing controller placement approaches

In general, the controller placement in an SDN network depends on three aspects: the application requirements, the physical network topology and the performance of the method used for this purpose. As with any distributed management system, a control plane in SDN should be flexible, scalable and robust. In SDN, the application plane provides flexibility to the network, as it gives the switches the ability to adapt their operation in the event of network changes. Meanwhile, the scalability and robustness of a distributed control plane are properties influenced largely by controller placement, as mentioned in the last section. Therefore, both the controller placement and the number of controllers not only have to be selected to satisfy the application requirements, but also to provide scalability and robustness to the control plane.

These criteria, scalability and robustness, are applied to classify the different approaches proposed by the research community to solve the controller placement problem. This classification is shown in Fig. 4.1. Here, approaches are classified according to the requirements taken into account to select the controller placements. That is, if an approach takes into account the available resources in the network to select the controller placement (e.g., links and controller capacities), it is said that the approach selects the controller placements that create a scalable control planes. On the another hand, if during the controller placement selection, an approach considers any metric that allows the network to minimize control data loss in the event of network failures (switches and/or links failures), it is considered that the approach selects the controller placements that create a robust control plane.

The switch-to-controller delay is regarded as one of the main restrictions for any application running on the control plane. Therefore, this metric is also used to classify the approaches in Fig. 4.1.

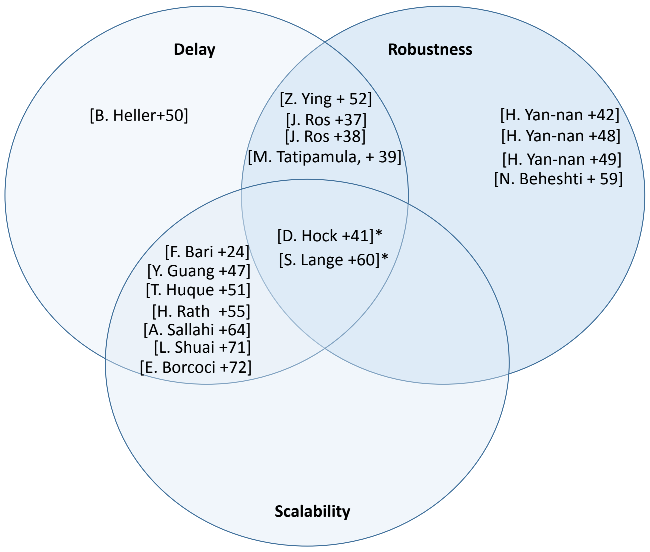


Figure 4.1: Classification of controller placement approaches according to their objective.

Switch-to-controller delay

As can be seen in this diagram, most of the solutions consider the switch-to-controller delay in the selection of controller placement. In [52], [26], [53], and [57] this delay is limited to a specific value that is defined by the application, while in the remaining approaches grouped in this category, the objective is to find the best trade-off between this delay and other requirements. In this case, the selection of the controllers may not satisfy the delay required for the correct operation of an application, as the controller selection depends on other requirements and restrictions. For instance, in [40] the objective is to maximize the controller utilization while minimizing the switch-to-controller delay.

From the pool of approaches considering the switch-to-controller delay in the controller selection, only [26] and [66] consider the flow setup time in their formulation.

Assumptions

To simplify the complexity of the problem and therefore its resolution, in [26], [50], [57], [66], [49], and [40] the authors assume the facilities where a controller can be installed are limited and known. As shown in [53], limiting the controller location in the network may lead to an inefficient solution compared to considering the entire solution space. As this restriction may increment the number of controllers required to find a specific delay, or if the objective is to minimize the switch-to-controller delay, the minimum delay value found may not satisfy the delay required for an application.

Another assumption to simplify the controller placement problem is to define the number of controllers k in advance. This assumption is considered in approaches [50], [51] and [54]. In [44] both assumptions are adopted, the number and a possible topological placement of the controllers.

Defining the number of controllers k in advance clearly simplify the problem resolution, as the problem is reduced to find the best k placement for the controllers. But in practice, it is hard to know in advance both the exact number of controllers required to handle a network and their possible location in the network. Given that, it has been demonstrated in [43] that the number of controllers to be used and their placement depend on the network topology on the one side, and the trade-off between the availability and the performance metrics fixed by the network operator on the other side.

Despite a good enough trade-off between metrics can be find for k controllers, two situations can happen:

- the number of controllers k can satisfy the required network/application requirements with great difference compared to the minimum requirements, that leads to the network resources to be sub-utilized or,
- the number of controllers can not satisfy the required network requirements. Therefore, considering these assumptions in the controller

placement problem have important effects on the network performance, as has been shown in [43] and [62].

Approaches that appear in more than one category are marked with an asterisk (*). This is the case for approaches [43] and [62] that evaluate the trade-off between switch-to-controller delay and different metrics. To do that, these approaches evaluate all possible controller placement on the network while varying the number of controllers.

From the set of solutions presented in Fig. 4.1 there is only one approach that does not made any assumption about the number of controllers neither their placement in the network, this is [53]. However, the resulting control plane built from controller placement selected using this approach does not consider in its formulation any metric to minimize the data loss in case of network failures.

To summarize, the most important disadvantage found from the controller placement approaches shown in in Fig. 4.1 are related with:

- real network requirements are not taken into account,
- relevant aspects in the control plane design are assumed,
- the delay between switch-to-controller and inter-controller delay are not restricted and,
- few controller placements approaches provide robustness to the control plane, through greedy strategies.

Complexity of approaches

Below, the complexity of each one of the strategies used for discovering the controller placements are presented.

Exact solution: In terms of the complexity of the solutions, approaches that solve the controller placement based on Integer linear programming such as [26] and [66], find an optimal solution. However, it becomes impractical with real-sized networks due to the number of constraints they handle, even if these approaches only take into consideration the application requirements in terms of delay and the availability of resources on the links.

Heuristic solution: This strategy reduces the solution complexity compared when using an exact strategy. This is an alternative to find a feasible solution in a scalable way with a good approximation to the optimal solution. Most of the approaches named in Fig. 4.1 formulate the CPP by using heuristic.

Clustering solution: Traditional clustering approaches such as *k-Center* or *k-Median* are also considered when selecting controller placement in an SDN network, given the similarities between the objective of these problems and the controller placement problem. Clustering approaches reduce the complexity of the solution compared to an Integer Linear Programming (ILP) formulation where each switch is evaluated as a possible controller. This is because, in a clustering approach, a cluster is created from a reference switch, which is selected as controller placement, if any switch in the cluster can improve a given restriction. If exists a switch in the cluster that improves the delay to all switches in the cluster, it is declared as controller placement. Complexity of clustering approaches is defined by $O(nk)$, where n is the number of switches in the network and k the number of controllers. However, a critical aspect in clustering approaches is the selection of the reference switch to initiate the creation of the clusters. This switch is usually selected randomly. Even though it simplifies the problem, it affects the density or distribution of elements in the clusters. In a SDN network, this situation may leave the controller with a high number of switches carrying a high load, whilst other controllers may be underutilized.

Using clustering approaches to solve the controller placement problem have other additional issues. First, the number of controllers to be located in the network has to be known in advance. Second, the optimized metric value may not satisfy the specific application requirement. Finally, a significant shortcoming of these simple formulations (*k-Center* or *k-Median*) is that a small number of distant switches, called outliers, can exert a disproportionately strong influence over the final solution.

This is clearly true for min-max problems like *k-Center*, where a single switch residing far from the other switches may force a center to be placed in its vicinity. With min-sum formulations this effect is reduced, but it is still possible if the switches are sufficiently far away. These switches have the undesirable effect of increasing the cost of the solution, without improving the level of service to the majority of switches.

As shown in Fig. 4.1, solutions that consider robustness, application requirements and resource constraints are limited.

4.4 K-Critical

$\mathcal{K}-\mathcal{Critical}$ is a clustering approach designed in this thesis to find both the number of controllers and the controller placement that satisfy the application requirements and reduce data loss in an SDN network. $\mathcal{K}-\mathcal{Critical}$ provides an efficient way of finding controller placement in SDN networks. This is because, it restricts network area where the optimal location of each controller is sought, instead of evaluating every switch as a possible controller placement.

Given an SDN network topology denoted by the tuple $G = (V, E)$, where $V = \{1, \ldots, \mathcal{N}\}$ is the set of vertices and E is the set of edges, and a switch-to-controller delay restriction called $\mathcal{D}_{req}$, $\mathcal{K}-\mathcal{Critical}$ finds the set of controllers $C = \{C_1, \ldots, C_k\}$, where $C \subseteq V$.

$\mathcal{K}-\mathcal{Critical}$ divides the controller placement problem into two subproblems. First, the *candidate switch selection* problem to find the set of candidate switches (defined as $C_{switches}$) that can become a controller placement, and second, the *controller placement selection* problem to find the optimal controller placement (denoted as (C_j)) from the set of candidate switches. For illustrative purposes consider the network topology G shown in Fig. 4.2.

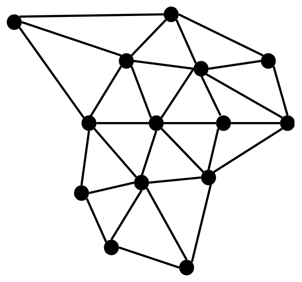


Figure 4.2: Network topology.

Fig. 4.3 shows the graph G in a geometrical space in order to illustrate clearly the sequence of processes executed by $\mathcal{K}-\mathcal{Critical}$ to select the controller placements. As illustrated in Fig. 4.3, $\mathcal{K}-\mathcal{Critical}$ finds the controller placements required for a network topology G in k phases (each row represents a phase), selecting a controller placement C_j in each phase as explained below.

Through the *candidate switch selection* problem, $\mathcal{K}-\mathcal{Critical}$:

1) selects a reference switch called the *critical node* (Cr_n) that is used as reference to define the set of candidate switches $(C_{switches})$,

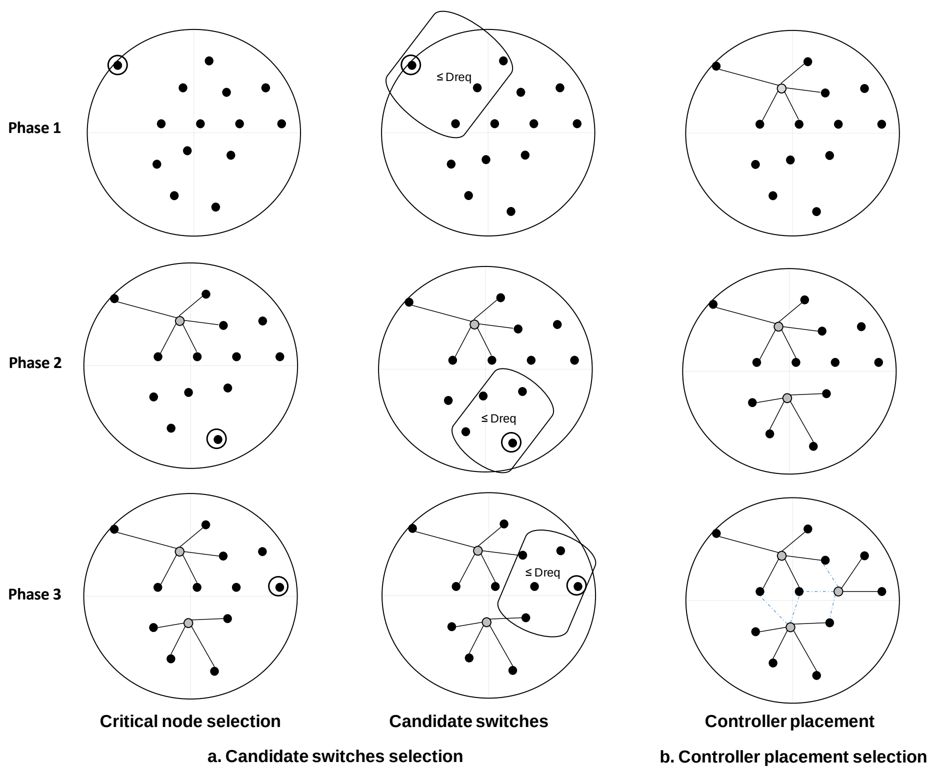


Figure 4.3: k-Critical process.

2) discovers the controller placement from the set of $(C_{switches})$ (These processes are represented by the first and second columns in Fig. 4.3, respectively),

3) creates from each candidate switch a subgraph G^*, this consists of all the switches $v \in V$ that may be managed by a candidate switch, in the event that such a candidate switch is selected as controller placement, denoted as V_v^*, and the links between these switches defined by E_e^*.

4) gets as result, the subgraph created from each candidate switch, defined by $G^* = (V_v^*, E_e^*)$.

On the another hand, the *controller placement selection* problem:

1) evaluates the connectivity characteristics of each switch $v_i \in C_{switches}$ in relation to the set of switches in its subgraph G^*. For this purpose, a tree rooted from each candidate switch v_i that includes the switches in its subgraph G^*,

2) measures the expected data loss of the resulting tree,

3) selects as controller placement the switch that minimizes the data loss, creating the control plane topology.

In each phase, these two processes are executed, as a result the candidate switch $v_i \in C_{switches}$ that minimizes the expected data loss is selected as a controller placement, C_j. These are the switches shown in grey in each phase in the last column in Fig. 4.3. For each controller placement selected, a cluster that includes the switches it can manage and the links between these switches is created.

Fig. 4.4 shows how these two problems are related.

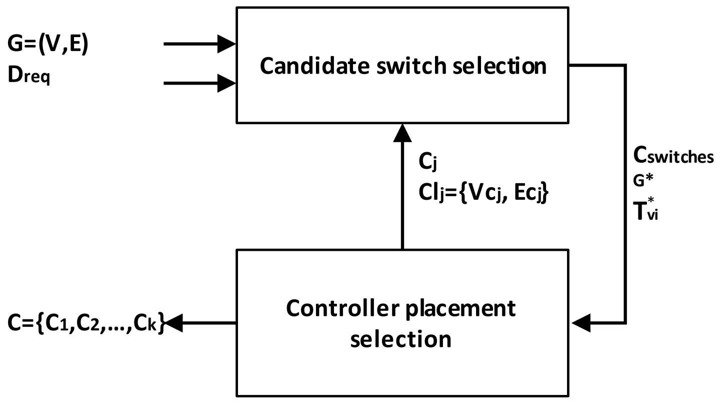


Figure 4.4: $\mathcal{K}-\mathcal{C}ritical$ processes.

4.4.1 Definitions

$\mathcal{K}-\mathcal{C}ritical$ partitions a graph in k clusters, each one managed by a controller. A cluster can be formally defined as:

Definition 3.1 ($Cluster_j$, Cl_j). A cluster consists of the set of switches that are in a range less or equal to $\mathcal{D}_{req}$ from a controller placement $C_j \in C$, and the set of links that connect these switches to the controller. The set of switches in a cluster is defined as:

$$V_{C_j} = \{v_{j'} : d(v_{j'}, C_j) \leq \mathcal{D}_{req}, \quad j, j' \in \mathbb{N}, 1 \leq j' < \left|V'\right|, 1 \leq j < |C|\}, \quad (4.1)$$

where V' is the set of switches that has not been included in a cluster.

The control channel between each switch $v_{j'} \in V_{C_j}$ to controller placement $C_j \in C$ is represented as a list of traversed links, as follows:

$$E_{Cj} = p(v_{j'}, C_j) = \{e_1 = \{v_{j'}, v_1\}, e_2 = \{v_1, v_2\}, ..., e_k = \{v_{k-1}, C_j\}\}, \quad (4.2)$$

where $e_1, e_3, ..., e_k \in E$. Therefore, a cluster created from a controller C_j can be represented as $Cl_j = (V_{C_j}, E_{C_j})$, and the set of all the clusters in phase $j < k$ in a network can be denoted as $G_{Cl} = (Cl_1, Cl_2, ..., Cl_k) \rightarrow G_{Cl} = (V_{Cl}, E_{Cl})$. Note that after finding the controller placements required, that is after k phases, $V_{Cl} = V$.

The set of switches that have not been included in a cluster V' is defined in the first phase as $V' = V$, and in each phase, when a controller placement is selected, this set of switches is updated as follows $V' = (V' - V_{C_j})$.

$\mathcal{K} - \mathcal{C}ritical$ defines the set of switches from where a controller placement is selected taken as a criterion, the communication delay time between switches in the network. To select these switches, $\mathcal{K} - \mathcal{C}ritical$ defines a reference switch called *critical node* that is formally defined as follows:

Definition 3.2 (Critical node, Cr_n). A critical node is a switch that is not included in any cluster Cl_j, and that has the furthest distance to the set of switches that has already been assigned to a cluster, defined by V_{Cl}. The shortest delay between a vertex $v \in V_{Cl}$ to a vertex in $v' \in V'$ is defined as $d(v, v')$. Then, in each phase, the critical node in V' can be formally defined as the node v' that:

$$maximize \sum_{v' \in V'} d(v, v') \quad \forall v \in V_{Cl}, \quad (4.3)$$

where $v' = Cr_n$ and, therefore, $(Cr_n \subset V')$.

The first critical node selected in the network is the switch with the highest delay to the rest of the switches, given the absence of any cluster, $G_{Cl} = \emptyset$.

$\mathcal{K} - \mathcal{C}ritical$ leverages link delays to determine the set of candidate switches $C_{switches}$: only switches for which the maximum delay to the critical node Cr_n is less or equal to $\mathcal{D}_{req}$ are selected as candidate switches to become a controller. These switches are illustrated in Fig. 4.3 in Phase 1b. The set of candidate switches to manage a critical node Cr_n is formally defined as:

Definition 3.3 (Candidate switches, $C_{switches}$). This is the set of switches $v_i \in V'$ that, due to their location, are at a delay less or equal to $\mathcal{D}_{req}$ from the *critical node*, Cr_n. This set of switches can be defined as follows:

$$C_{switches} = \{v_i : d(v_i, Cr_n) \le \mathcal{D}_{req}, \quad i \in \mathbb{N}, 1 \le i < |V'|\}, \qquad (4.4)$$

where $|V'|$ defines the number of switches in the set V'.

4.4.2 Problem formulation

In physical SDN network topologies, where switches forward traffic between them and their traffic rate is not known, $\mathcal{K} - \mathcal{C}ritical$ provides a first controller placement approximation to guarantee:

- all switches in the network topology G are managed by a controller and,
- the controller placements guarantee a minimum delay communication $\mathcal{D}_{req}$ to each switch, while reducing, as much as possible, the flow communication time between any pair of end switches in the network.

To find the number of controllers and their placement in the network, the following sets and parameters have been defined in $\mathcal{K} - \mathcal{C}ritical$:

Topology:

G	Network topology, $G = (V, E)$.
V	Set of switches in the network topology G.
E	Set of links in the network topology G.
$N(v)$	Neighbour switches of switch v.
$d(i,j)$	Delay in link $(i,j) \in E$.
$D[i,j]$	Matrix of delays for links in E.

Requirements:

D_{req}	Required switch-to-controller delay.

Variables:

Cr_n	Critical node.
$C_{switches}$	Set of candidate switches.
G^*	Subgraph created from each candidate switch.
V_v^*	Set of switches in G^*.
E_e^*	Set of links in G^*.

$T_{(V_v^*,v_i)}$	Shortest tree rooted at candidate switches v_i.
$T_{v_i}^*$	Set of shortest trees rooted at switches $v_i \in C_{switches}$.
C_j	Controller placement selected.
Cl_j	Cluster created from controller placement C_j .
Vc_j	Set of switches in cluster Cl_j.
Ec_j	Set of links in control channels from each switch in Vc_j to C_j.
C	Set of controller placement selected in network topology G.
G_{Cl_j}	Set of clusters created from each controller placement.

4.4.3 Candidate switch selection

The candidate switch selection problem can be formally stated as follows:

Given: a network topology represented by a graph $G(V, E)$ and the application requirement in terms of switch-to-controller delay, $\mathcal{D}_{req}$.

Output: a shortest tree $T_{(V_v^*,v_i)}$ rooted at each candidate switch, where $v_i \in C_{switches}$.

Objective: find the set of candidate switches in each phase, such that in k phases it i) guarantees the coverage of all switches in the network by at least one controller and ii) ensures an specific switch-to-controller delay for all switches in the network.

Algorithm 1 specifies the selection of the candidate switches in an SDN network. Essentially, this process is divided into three processes:
First, the critical node Cr_n is selected from the switches that have not been assigned to any cluster (that is set V'), after that, the possible candidate switches $C_{switches}$ that can manage Cr_n are found (lines 1-3).
Second, from each candidate switch $v_i \in C_{switches}$ a subgraph $G^* = (V_v^*, E_e^*)$ is created, which contains all the switches that a candidate switch v_i can manage in the event of being selected as a controller placement as well as the links between these switches. This subgraph is used to find the shortest paths from each switch $v \in V_v^*$ to the candidate switch v_i creating a tree defined as $T_{(V_v^*,v_i)}$. The set of shortest trees created from each candidate switch are stored in the matrix defined as $T_{v_i}^*$ (lines 4-16).
Third, the *controller placement selection problem* evaluates the data loss from each shortest trees rooted at candidate switches, selecting the switch

Algorithm 1 Candidate switch selection algorithm.

```
Input: G=(V,E), $D[i,j] \leftarrow$ SP Delay Matrix. $\mathcal{D}_{req} \leftarrow$ Req. Delay, $V' = V$, $N(v)$,
    [C, $C_{switches}$, $G_{Cl}$, $Cl_j$, $E_e^*$, $V_v^*$, $T_{v_i*}$] $\leftarrow \emptyset$
Output: set of trees rooted at the candidate switches .
1: while $V' \neq \emptyset$ do
2:   Find critical node $Cr_n$ from $V'$ through Eq.(4.3)
3:   Find candidate switches $C_{switches}$ that can manage $Cr_n$ through Eq.(4.4)
4:   for each switch $v_i \in C_{switches}$ do
5:     $V_v^* \leftarrow$ set of switches defined by Eq.(4.1), where $v_i$=$C_j$
6:     for each switch $v \in V_v^*$ do
7:       for $v_n \in N(v)$ do
8:         if $v_n \in V_v^*$ then
9:           $E_e^* = E_e * \cup e(v, v_n)$
10:        end if
11:      end for
12:      $T_{(V_v^*, v_i)} \leftarrow$ shortest path $(v, v_i)$
13:    end for
14:    Define subgraph $G^* = (V_v^*, E_e^*)$ for $v_i$
15:    $T_{v_i}^* \leftarrow T_{(V_v^*, v_i)}$
16:  end for
17:  Select Controller placement $C_j$ from $C_{switches}$ (Go Algorithm 2)
18:  Create cluster from controller placement selected, $Cl_j = (V_{Cj}, E_{Cj})$ through
     Eq.(4.1) and Eq.(4.2)
19:  $V' = V' - V_{C_j}$
20:  $G_{Cl} = G_{Cl} \cup Cl_j$
21:  $\mathcal{C} = \mathcal{C} \cup C_j$, C$_j \leftarrow \emptyset$, $\mathcal{C}_{switches} \leftarrow \emptyset$, $G^* \leftarrow \emptyset$
22: end while
```

v_i that minimizes the data loss in the SDN network. After selecting a controller placement C_j, the set of switches as well as the set links involved in the control channels from each switch to the controller create a cluster, $Cl_j = (V_{C_j}, E_{C_j})$. Consequently, the variables are updated (lines 17-21) to find the next controller placement. This process finishes when all switches are included in a cluster, this is when $V' = \emptyset$.

4.4.4 Controller placement selection

Intuitively, a network is robust when communication between two switches is not severely affected by –or can efficiently recover from– network failures, thus minimizing (ideally, avoiding) data loss in these cases. In $\mathcal{K} - \mathcal{C}ritical$, it is considered that:

- control channels are the shortest paths from a controller to a switch,
- each controller manages its nearest switches,
- several control channels can use the same switches/links.

In terms of a control plane, this notion of robustness implies that the data loss due to a switch failure along control channels of a given SDN network is low, meaning that the number of involved switches in a control path, and therefore its hop length, is small. In addition, the number of control channels that share a link/switch also should be low. That is because, in case of a switch failure, all the control channels using this switch/link are broken.

In this context, the robustness of a tree $T_{(V_v^*,v_i)}$ rooted at $v_i \in C_{switches}$, can be estimated from the expected data loss of each switch in the tree. Note that for tree networks, there is no loss of generality in considering only switch failures because, as far as the amount of data loss is concerned, the failure of a switch is equivalent to the failure of the link to the parent. This applies to every switch except the candidate switch (root switch), which is assumed not to fail. This is because, if the root fails, data forwarded from each switch in the tree to the controller is lost.

In the context of the *controller selection problem*, the robustness for a tree $T_{(V_v^*,v_i)}$ can be estimated as the sum of the expected data loss when considering that each switch fails in different times. That can be defined as follows: the expected data loss d_x for a switch x depends on the number of switches n_x that are rooted at it, and on the probability that the switch x fails. If it is considered that every switch in the tree topology $T_{(V_v^*,v_i)}$ has the same loss probability, the expected data loss d_x depends on the set of downstream switches in the subtree rooted at failed switch x, T_x, as these are disconnected from the controller. This can be formalized as follows.

Let $W(x)$ be a variable that indicates if a switch is affected by failure of switch x, that can be defined as follows:

$$W(x_i) = \begin{cases} 1 & \text{If} \quad i \in T_x \\ 0 & \text{Otherwise} \end{cases} \tag{4.5}$$

where T_x represents the subtree rooted at switch x. Therefore, the number of switches disconnected from the control plane due to a failure in switch x, are the downstream switches (D_n) of x (including switch x itself), that can be computed as:

$$D_n(x) = \sum_{i \in T_{(V_v^*, v_i)}} W(x_i) + 1. \tag{4.6}$$

The expected data loss of a tree $T_{(V_v^*, v_i)}$ can be defined as:

$$d(T_{(V_v^*, v_i)}) = \frac{1}{|T_{(V_v^*, v_i)}| - 1} \sum D_n(x) \quad \forall x \in T_{(V_v^*, v_i)}, \tag{4.7}$$

where $|T_{(V_v^*, v_i)}|$ is the number of switches in the tree $T_{(V_v^*, v_i)}$.

The candidate switch v_i selected as a controller placement is the switch that minimizes the data loss:

$$\text{Minimize} \quad d(T_{(V_v^*, v_i)}). \tag{4.8}$$

Therefore, the *controller placement selection problem* can be formally stated as follows:

Given: a set of shortest trees rooted at switches included in the set of candidate switches, defined as $T_{v_i}^*$.

Output: controller placement C_j.

Objective: Discover the controller placement that maximizes the control plane robustness.

Algorithm 2 specifies the selection of controller placements in an SDN network. It starts computing the data loss for each tree rooted at a candidate switch $T_{(V_{v*}, v_i)} \in T_{v_i}^*$, by considering each switch x in the tree fails in different times (line 1-7). Computing the data loss of a tree as the sum of the nodes disconnected when each switch $x \in T_{(V_{v*}, v_i)}$ fails (lines 3-7). This result obtained for the set of trees rooted at the candidate switches is stored in a vector defined as $d(T_{(V_v^*, v_i)})$ (line 8). At the end, the candidate switch that minimizes the data loss is selected as a controller (line 10).

$\mathcal{K} - \mathcal{C}ritical$ tackles failures in controllers by selecting controllers of backup. These controllers are selected at the same time a controller placement is found, these are the candidate switches that maximize the control plane robustness.

Algorithm 2 Controller selection algorithm.

```
Input: $T^*_{v_i} \leftarrow$ set of trees rooted at candidate switches
Output: Controller placement $C_j$.
 1: for each tree $T_{(V^*_v, v_i)} \in T^*_{v_i}$ do
 2:     for each switch $x \in T_{(V^*_v, v_i)}$ do
 3:         for $x \in T_x$ do
 4:             compute $W(x_i) \leftarrow$ through Eq.(4.5)
 5:         end for
 6:         $d(x) \leftarrow$ saves $W(x_i)$ for each switch $x \in T_{(V^*_v, v_i)}$
 7:     end for
 8:     $d(T_{(V^*_v, v_i)}) \leftarrow$ saves $\sum d(x)$ for each tree $T_{(V^*_v, v_i)} \in T^*_{v_i}$
 9: end for
10: return   $\arg\min_{v_i \in C_{switches}} d(T_{(V^*_v, v_i)})$.
```

4.4.5 Complexity analysis

The complexity of controller placement approaches is defined by the number of operations required to select a controller within a network graph G. In the case of $\mathcal{k} - \mathcal{Critical}$, the number of operations depends on the size of the network and of the set of candidate nodes in each phase. This is because, $\mathcal{k} - \mathcal{Critical}$ builds a delay-based shortest path tree from each candidate switch in each phase, and then executes a heuristic algorithm on the computed trees to quantify the data loss. In the worst case, that is when all switches (N) in the network are candidates ($N = k$) and therefore all the switches are included in the tree, the complexity of $\mathcal{k} - \mathcal{Critical}$ is defined by $O(n) * (n-1)$, that is $O(n^2)$. Note that this is the case when only one controller manages the whole network.

Although $\mathcal{k} - \mathcal{Critical}$ reduces the number of switches evaluated to select a controller placement when $k > 1$ compared to other approaches, the set of operations executed to compute the data loss per each candidate switch can take a long time, since that it depends on the network topology size and the network connectivity. Thus, aiming to provide a near-optimal solution within reasonable computational effort, the next section presents a function to measure the tree robustness to select the controller placement in $\mathcal{k} - \mathcal{Critical}$ in an efficient time.

4.5 Heuristic Algorithm for k-Critical

From the robustness metric presented in the previous section, it can be deduced that one of the main characteristics that a controller placement must have to reduce the expected data loss in a network is a high number of interfaces and that it is located at a short delay from the other switches in the network. That is, the diameter of the control plane topology should be as small as possible, as longer paths imply larger data loss as many switches depend on the switches near the controller, longer transmission times and higher traffic load on links. With respect to controller connectivity, the intuition is that a high connectivity (a high number of switch interfaces) should present a homogenous switch distribution on the branches to reduce the data loss, by reducing the branch length and the number of downstream switches at each switch of the tree. In consequence, the control plane built from selected controller placement with these characteristics tends to be wide (*i.e.*, switches have many children) near the controllers and narrow (*i.e.*, with decreasing number of children) for switches closer to leaf switches.

Based on these concepts, a function called *theta* is defined to evaluate the robustness property of the tree topology created from each candidate switch. This function evaluates the connectivity characteristics of each candidate switch $v_i \in C_{switches}$, specifically the tree diameter and the switch degree characteristics, as follows:

Definition 3.4 (Theta Function). This function is defined by two metrics that are constructed by considering a weighted combination of the switch connectivity (switch degree) and the path weight (delay and network diameter). In general, θ function is defined as follows:

$$\theta = \gamma \times (\text{switch connectivity}) + (1-\gamma) \times (\text{path weight}) \tag{4.9}$$

In function θ, the coefficient γ weighs the switch connectivity and the path weight for each switch $v_i \in C_{nodes}$. This coefficient is defined by:

$$\gamma = \frac{\mathcal{L}_h}{\mathcal{L}_{Max}}, \tag{4.10}$$

being $0 \leq \gamma \leq 1$, and where $\mathcal{L}_h$ is the maximum path length measured in the number of hops from the switch evaluated v_i to a leaf switch, and $\mathcal{L}_{Max}$ is the maximum path length found among the candidate controllers. Note that γ can be conveniently redefined to allow optimization of any other variable that needs to be taken into consideration in a particular case.

Note that γ is really a function of the depth of a switch in the tree, as it measures the depth of the deepest leaf in the tree created from each candidate switch. Coefficient γ takes the lowest value for the candidate switch with smallest diameter, and 1 for the candidate switches with the largest diameter. This factor depends on the characteristic of the underlying graph; it is not a property of the control plane.

Function θ provides a weight of the distribution of switches in a tree that is defined as:

$$\theta(n) = \gamma(v_i)\frac{deg(v_i)}{N - (deg(v_i) + h_i(v_i))} + \big(1 - \gamma(v_i)\big)\frac{\delta_{max}(v_i)}{\mathcal{D}_{req}}. \tag{4.11}$$

The first term in Eq.(4.11) weighs the connectivity of a switch that is defined as the relation between switch degree $deg(v_i)$ and the distribution of the N switches to h_i hops in the resulting shortest tree topology, $T_{(V_{v*},v_i)}$. $h_i(v_i)$ is the number of switches that a candidate switch has at i hops in its associated tree, and $\delta_{max}(v_i)$ is the maximum root-to-leaf delay within the control tree rooted at v_i. The second term of this function weighs the path weight in terms of delay, this is defined by the maximum delay value that the candidate switch reaches when it is the root, with respect to the required switch-to-controller delay, $\mathcal{D}_{req}$.

The candidate switch with best relation between the delay and the switch distribution is selected as the controller placement. Considering the physical network diameter (γ) allows the selection of the candidate switch with the best relation between the delay and the switch distribution in its branches.

4.5.1 Fault-Tolerant SDN controllers

In SDN networks, controller failures are tackled by considering backup controllers called slave controllers in the Openflow standard, which change their operational mode when their master controller fails.

An important limitation when considering backup controllers is the fact that these controllers do not have information about the network state after they change their operation mode from slave to master controllers.

In an SDN model where each controller has information about the whole network, but only manages a specific set of switches based on its local network state information, the controllers announce critical local events to all other controllers in the control plane to keep the network information consistent. In this context, a long delay between controllers has a large impact on the network performance, as it affects the time it takes for controllers to update their network state information. As a consequence, controllers can make wrong decisions if network state information is inconsistent.

To handle this issue, in [71] authors propose that the controllers replicate the network and application state (the Network Information Base or NIB) in a shared data store implemented on servers. To ensure a smooth transition from slave to master controller, operational master controllers always update the NIB before modifying the state of the network. When a master controller fails, its backup controller takes over its role and its first action is to read the current state from the data store. As the network state in the data store is always up-to-date, backup controllers have a consistent view of the network from the outset.

Authors in [80] propose a hierarchical SDN control plane, where the information about network state is originated at the bottom of the hierarchy and is passed upwards from children to parent. On the other hand, configuration information that defines the switch behaviour is forwarded downwards from servers to the switches, where each controller/server computes the configuration for their associated switches. At the highest level, there is a logical server that has information about the whole network. Architecturally, logical servers appear as a single machine (that is, they are logically centralized), but in practice they can be replicated for fault tolerance.

In general, the performance of hierarchical control planes depends on the convergence time, which is a function of the depth of the hierarchy [80]. This convergence time depends mainly on two aspects: i) the processing capacity of the switches and controllers and ii) the delay between switch-to-controller and between controller-to-server.

In this section an extension of $\mathcal{k} - \mathcal{Critical}$ is introduced to select the server placements to build hierarchical SDN network architectures. Selection of the server placements is based on i) the placement of the controllers and ii) the convergence time ($\mathcal{D}_{cvg}$) required in the SDN network. The convergence time ($\mathcal{D}_{cvg}$) may be the maximum response time permissible for an application without affecting the quality of service. This delay has the constraint that $\mathcal{D}_{req} + \mathcal{D}_{c,s} \leq \mathcal{D}_{cvg}$, where $\mathcal{D}_{c,s}$ is the controller-to-server delay.

$\mathcal{K} - \mathcal{Critical}$ applies the same criteria to select the server placements as in the controller placement selection. That is, from the set of controller placements found, $\mathcal{k} - \mathcal{Critical}$ selects the critical controller. This controller is taken as a reference to find the set of candidate switches where a server can be installed. Note that as in the controller placement selection (defined in Section 4.4.4), considering the critical switch/controller allows the network to discover the controller that limits the server placement in the network.

After evaluating each candidate switch through the theta function Eq.(4.11), the switch selected as server placement is the one that has the best relation between i) the switch distribution (highest theta function value) and ii) the number of controllers that can reach the candidate switch in a delay less or equal to $\mathcal{D}_{c,s}$.

Below, the concept definitions are introduced to find the server placements in an SDN network.

Definition 3.5 (Critical controller placement, Cr_c). A critical controller placement is the controller that is included in set C, and has the furthest distance from the rest of controllers. If the delay between two controllers in set C is defined as $d(C_i, C_j)$, then the critical controller Cr_c is the controller C_k that:

$$\text{maximize} \sum_{C_k \in C} d(C_k, C_j) \quad \forall C_j \in C, \quad C_k \neq C_j. \tag{4.12}$$

Definition 3.6 (Candidate server placements, C_{sp}). This set consists of all the switches v_i that are not included in C and have a delay to the critical controller Cr_c less or equal to $\mathcal{D}_{c,s}$. The set of switches that are not included in C is defined as $V_c' = (G - C)$. Therefore, the set of candidate switches selected to become a server placement is defined as follows:

$$C_{sp} = \{v_i : d(v_i, Cr_c) \leq \mathcal{D}_{c,s}, \quad \forall v_i \in V_c', \quad i \in \mathbb{N}, 1 \leq i < \left|V_c'\right|\}, \tag{4.13}$$

where $\left|V_c'\right|$ defines the number of switches in the set V_c'.

Definition 3.7 (Candidate controller covered, $C(v_i)_{cover}$). This is the number of controllers in set C that can reduce their delay communication time to $\mathcal{D}_{c,s}$ in the event of candidate switch v_i being selected as a server placement.

$$X_{(C_j, v_i)} = \begin{cases} 1 & \text{If} \quad d(v_i, C_j) < D_{c,s} \\ 0 & \text{Otherwise} \end{cases} \tag{4.14}$$

$$C(v_i)_{cover} = \sum_{v_i \in C_{sp}} X_{(C_j, v_i)} \quad \forall C_j \in C, \tag{4.15}$$

where, $X_{(C_j, v_i)}$ is a binary variable, which becomes 1 if the delay from candidate switch v_i to a controller $C_j \in C$ is less or equal to $\mathcal{D}_{c,s}$. Otherwise, it is 0.

Therefore, the server placement problem can be formally defined as follows:

Given: a set of controller placement C in network G, *Output*: set of server placements S,

Objective: To find the set of server placements that ensures a communication controller-to-server delay not higher than $\mathcal{D}_{c,s}$.

Algorithm 3 defines the process to select the server placements. In order to restrict the controller-to-server delay requirement, an initial condition is defined in Algorithm 3. This condition limits $\mathcal{D}_{c,s}$ in the control plane to $\frac{D_{cvg}}{k+1}$, where $\mathcal{D}_{cvg}$ is the maximum convergence time delay for an application running on network G (lines 1-3). This condition, together with the critical controller criterion, guarantees that the minimum number of servers is added.

Initially the delay from each controller to a server is defined as infinite, given that no server placement has been selected in the network. Algorithm 3 is executed while there are controllers to a delay higher than $\mathcal{D}_{c,s}$ to a server.

This process is executed as follows. First, the critical controller $Cr_c \subset C'$ is discovered, along with the set of candidate switches C_{sp} that are selected to become a server placement for Cr_c (line 5-6). After that, each candidate switch $v_i \in C_{sp}$ is evaluated by means of function theta, and the number of controllers that v_i can reach to satisfy the required $\mathcal{D}_{c,s}$ is also computed. This is defined as $C(v_i)_{cover}$ (lines 7-13). In order to add the lowest number of server placements in the network, it is considered that any new server placement should reduce the delay to the maximum number of controllers in the network. For this purpose, the candidate switch (v_i) selected is the one that has the best relation between theta function value and the number of controller placements to which the candidate switch v_i can reduce the delay communication to $\mathcal{D}_{c,s}$. This relation is called θ', that is defined as: $\theta' = \frac{\theta}{C(v_i)_{cover}}$, selecting the candidate switch with the highest θ' value (lines 14-16). If the previous condition can not be satisfied, the candidate switch selected is the one that minimizes the delay to the rest of the controllers (lines 18-19). This process continues until $\mathcal{D}_{c,s}$ is satisfied for all connections between controller-to-server in the network.

4.5.2 An example

Consider the network topology in Fig. 4.5, for which the set of controller placements that satisfy a switch-to-controller delay requirement of $\mathcal{D}_{req}$ equal to 45 μsec must be found. In Fig. 4.5, the number on the edges represents the delay on the link. In $\mathcal{K} - \mathit{Critical}$ basically three processes are executed:

Algorithm 3 Controller placement selection.

```
Input: $(\mathcal{N} \times \mathcal{N})$ SP Delay Matrix. G=(V,E), $\mathcal{D}_{c,s}$, $\mathcal{D}_{cvg}$, $S = \emptyset$ , $d(C_i, S) = \infty \forall \ C_i \in C$, C'=C, $k = |C|$.
Output: Controller placement $C_j$.
 1: if $\mathcal{D}_{c,s} > \frac{D_{cvg}}{k+1}$ then
 2:     $\mathcal{D}_{c,s} = \frac{D_{cvg}}{k+1}$
 3: end if
 4: while exists a $C_i \in C' \rightarrow d(C_i, S_j) \geq \mathcal{D}_{c,s}$ do
 5:     $Cr_c \leftarrow$ Find the critical controller placement in C' through Eq.(4.12)
 6:     $V_c' = G - C' - S$
 7:     $\mathcal{C}_{sp} \leftarrow$ Find the candidate switches in $V_c'$ through Eq.(4.13)
 8:     for each candidate switch $v_i \in \mathcal{C}_{sp}$ do
 9:         Evaluate $\theta$, Eq.(4.11)
10:         Compute $C_{covered}$ through Eq.(4.15)
11:         $C(v_i)_{covered} = C_{covered}$
12:         $\theta'_{v_i} = \theta / C(v_i)_{covered}$
13:     end for
14:     if $C(v_i)_{covered} \in C_{covered} \neq 0$ exists then
15:         $S = S \cup v_i$, where $v_i$ is the candidate switch with the highest value in $\theta'_{v_i}$
16:         $C_c \rightarrow$ set of controllers $C_j \in C'$ for which $X_{(C_j, vi)} > 0$
17:     else
18:         Select $v_i \in \mathcal{C}_{sp}$ that $\rightarrow$ Minimize $\sum_{v_i \in \mathcal{C}_{sp}} d(v_i, C_j) \quad \forall C_j \in C$
19:         $S = S \cup v_i$
20:     end if
21:     $\mathcal{C}_{sp} \leftarrow \emptyset$, $C' = C' - Cr_c - C_c$
22: end while
```

1) Critical node selection, 2) Candidate switch selection and 3) Controller selection.

First, $k - \mathcal{Critical}$ selects switch 13 as a critical node in the network. The set of candidates switches that can manage the critical node consists of the following switches, $C_{switches} = \{8, 10, 11, 12\}$. By evaluating these switches through function theta Eq.(4.11), where N=13 and $h_i = 2$, the following values are obtained: $\theta_8 = 0.28$, $\theta_{10} = 0.67$, $\theta_{11} = 0.5$ and $\theta_{12} = 0.43$.

Note that even though the switches 10, 11 and 12 have the same degree (Fig. 4.6), the switch distribution in the tree created at switch 10 is better (in terms of switch distribution on the branches, reducing the data loss in case of switch failures) and therefore the highest function value is obtained.

From the controller placement selected (switch 10) a cluster (Cl_{10}) is created that includes all the switches it can manage, $Cl_{10} = \{4, 6, 7, 9, 10, 11,$

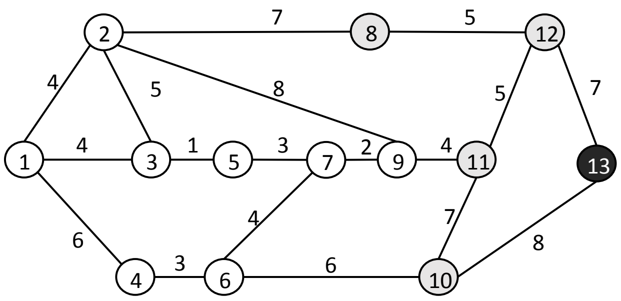


Figure 4.5: Network topology.

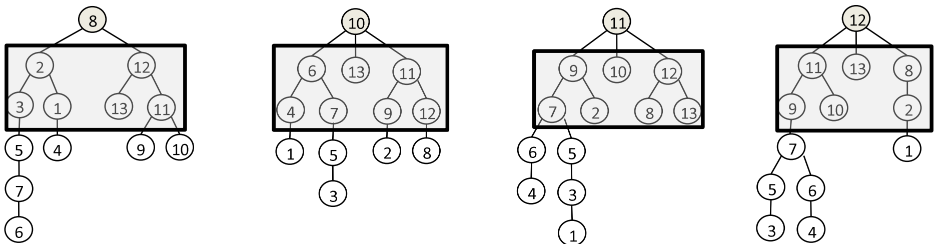


Figure 4.6: Tree topologies from candidate switches, $C_{switches} = \{8, 10, 11, 12\}$.

$12, 13\}$. As not all switches can be managed by the controller selected Cl_{10}, $k - Critical$ continues the process by finding another controller placement.

To do that, a new critical node is found from the set of switches that has not been included in a cluster. This set is defined as $V' = \{1, 2, 3, 5, 8\}$. From this set, the switch selected as the critical node is switch 8. The set of candidate switches that can manage the critical node is defined as $C_{switches} = \{1, 2, 3, 5, 8, 9\}$. Fig. 4.7 shows the set of shortest tree topologies created from each one of the candidate switches. By evaluating these trees through Eq. 4.7, trees rooted from switch 2 and switch 1 are found to have the lowest data loss. However, switch 2 is selected by function θ as it has more interfaces and can lead to a better distribution of the load on the branches than switch 1, reducing the data loss. Note that, there are switches that can be managed by different controllers, in the example, switch 9 can be managed by both controllers C_2 and C_10. In this case, a switch is managed by the controller that reduces the switch-to-controller delay.

As a result, the control plane consists of two controllers (C_2 and C_10), each one managing a specific set of switches as shown in Fig. 4.8.

The resulting control plane topology created from the controllers selected is robust, because the loss of management data due to switch or link failures

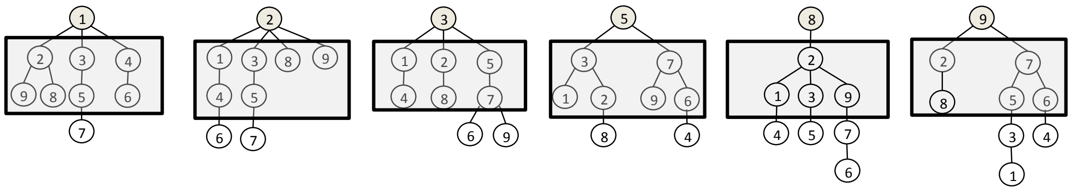


Figure 4.7: Tree topologies from candidate switches, $C_{switches} = \{1, 2, 3, 5, 8, 9\}$.

is minimized. Moreover, it is homogeneous because the number of switches along the different branches is balanced.

Note that the set of controller placements found by $\mathcal{k} - \mathcal{Critical}$ guarantees that every switch is managed by at least one controller and that the maximum delay between them is no higher than $\mathcal{D}_{req}$. However, the delay among controllers in the control plane is not limited. Therefore, controllers can be placed with a long delay between them, affecting the network response time. As introduced in the last section, a solution to reduce the impact on the performance of a distributed SDN network following network failures, consists of having a distributed data store (that can be considered as a controller) that can be consulted by any controller in order to update the network state on their local caches.

To find the server placements on the network, $\mathcal{k} - \mathcal{Critical}$ follows the same procedure as followed when finding the controller placements. That is, first the critical controller in C is controller C_10, along with the set of candidate switches to become a server placement, defined as $C_{sp} = \{4, 6, 7, 9, 11, 13\}$. By evaluating each one of these switches using function θ', the switch with the highest function value is switch 9. Consequently, this switch is selected as a server placement, which allows the time topology information is updated on the control plane to be reduced.

In Fig 4.8, the logical switch-to-controller and controller-to-server paths that define the control plane topology are illustrated. As can see from this figure, control paths of different switches can share resources.

4.6 Comparison of Controller placement approaches

$\mathcal{k} - \mathcal{Critical}$ is a controller placement approach inspired by clustering approaches. The main differences between $\mathcal{k} - \mathcal{Critical}$ and the traditional clustering approaches such as *k-center* and *k-Median*, solutions considered in [52], is that $\mathcal{k} - \mathcal{Critical}$:

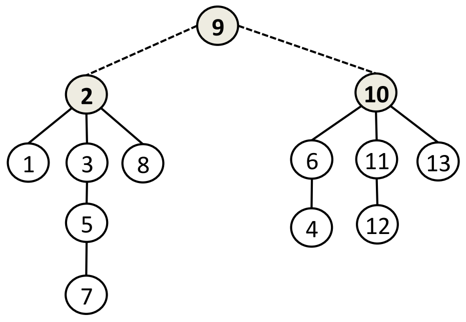


Figure 4.8: Control plane created from candidate switches selected.

- finds the controller placement to satisfy a specific switch-to-controller delay,
- finds the number of controllers required,
- selects the controller placement taking into account the switches outside the average delay in the network.

In order to compare $\mathcal{k}-\mathcal{Critical}$ performance with *k-center* and *k-Median* approaches, they have been adapted to discover the set of controllers that satisfies a delay constraint, defined as $\mathcal{D}_{req}$. Just like in $\mathcal{k}-\mathcal{Critical}$, in these algorithms a controller can be placed in any switch in the network.

4.6.1 k-Median problem

This clustering approach finds the set of controller placement C that minimizes the average propagation latencies between switch-to-controller, defined as $d(v, C_k)$, using Eq. (4.16).

$$L_{avg} = \frac{1}{\mathcal{N}} \sum_{v \in V} d(v, C_k). \tag{4.16}$$

Where $\mathcal{N}$ is the number of switches in a network.

Algorithm 4 describes this process. It starts identifying the shortest link in the network, selecting as candidate controller the switch i in the shortest link that minimizes the average delay (L_{avg}) to the rest of the switches in the network (lines 1-2). All switches v for which $d(v_i, v) \leq \mathcal{D}_{req}$ are joined to the selected switch v_i, creating a cluster (line 4). Then, each switch in the cluster is evaluated through Eq. (4.16), checking if there exists a switch in the cluster with better average delay than the switch v_i, that satisfies the

Algorithm 4 Controller selection algorithm using k-Median.

```
Require: ($\mathcal{N} \times \mathcal{N}$) Delay Matrix, $G = (V, E)$, $\mathcal{D}_{req} \leftarrow$ Req. Delay,Cluster $\leftarrow \emptyset$
1: find the shortest link $(i, j)$ in network G
2: select the switch $v_i$ that minimizes the delay to the rest of the switches in G
3: while there are switches not belonging to a cluster do
4:     Cluster$_k \leftarrow$ switches $v \in V$ that $d(v_i, v) \leq \mathcal{D}_{req}$, where $v \notin$ Cluster
5:     for each switch $v \in$ Cluster$_k$ do
6:         Evaluate $L_{avg}$
7:     end for
8:     Select the switch $j \in$ Cluster$_k$ that minimizes $L_{avg}$ as controller
9:     Cluster $\leftarrow$ Cluster $\cup$ Cluster$_k$, Find the nearest switch $i$ to the Cluster, where
       $k \notin$ Cluster, Cluster$_k \leftarrow \emptyset$
10: end while
```

$\mathcal{D}_{req}$ for all switches in the cluster (lines 5-7). If a switch $n \neq v_i$ exists that minimizes the L_{avg}, it is selected as new controller placement and the cluster is updated (lines 8-9). Otherwise the switch v_i is selected as controller. To find additional controllers, the algorithm finds the nearest switch v_i to the cluster or set of clusters created, building a new cluster, this process is repeated until all switches are included in a cluster.

4.6.2 k-Center problem

This approach finds the controllers so as to minimize the maximum distance of the switches v to their closest controller C_k (see Eq. (4.17)), as described in Algorithm 5.

$$L_{k-center} = \max_{v \in V} \min_{C_k \in C} d(v, C_k). \tag{4.17}$$

At the beginning, *k-Center* selects randomly a switch $v_i \in V$ in the network, creating a cluster with all switches for which $d(v_i, v) \leq \mathcal{D}_{req}$ (lines 1-3). If there exists a switch n in the cluster that minimizes the delay to the switches, it is selected as controller and the cluster is updated (lines 4-7). Otherwise the switch v_i is selected as controller. After that, the algorithm finds the furthest switch v_i to the cluster or set of clusters included in *Cluster*, repeating the process until all switches are assigned to a cluster.

4.6.3 Evaluation and results

This section performs an evaluation of the implications of the controller selection in the control topology performance in terms of delay, data loss,

Algorithm 5 Controller selection algorithm using k-Center.

```
Require: $(\mathcal{N} \times \mathcal{N})$ Delay Matrix, $G = (V, E)$, $\mathcal{D}_{req} \leftarrow$ Req. Delay,Cluster $\leftarrow \emptyset$
1: $v_i \leftarrow$ Select randomly a switch $v \in V$
2: while there are switches not belonging to the cluster do
3:     Cluster$_k \leftarrow$ Find from $(\mathcal{N} \times \mathcal{N})$ the switches $v$ that satisfy $d(v_i, v) \leq \mathcal{D}_{req}$,
       where $v \notin$ Cluster
4:     for each switch $v \in$ Cluster$_k$ do
5:         Evaluate $L_{k-center}$
6:     end for
7:     Select as controller the switch $j$ that minimizes the $d(v, j)$
8:     Find the furthest switch $k$ from Cluster
9: end while
```

the allocation of switches to controllers and average tree-depth.

For this purpose, three different network categories have been randomly generated, which are defined based on their network connectivity as: sparse, medium and dense networks. These networks were generated using the software described in [81]. Switches in sparse networks have between 1 and 10 neighbors; in medium networks between 20 and 30 neighbors, and in dense networks between 40 and 50 neighbors. For each category 100 networks were generated which consist of 100 switches with the edge distance uniformly distributed between 1 and 10 km. The controllers were found for different switch-to-controller delay values in the range of microseconds. The results presented are the average results obtained for each network category evaluated. MATLAB was used to evaluate the algorithms and their performance.

The performance and robustness of the trees built at the controllers selected for each one of the presented solutions and for each one of the network categories are evaluated. In order to compare the effect that the controllers selected have over control topology, a quantitative analysis is applied to determine the number of optimal controllers for each network category. Based on the maximum delay time reached for each network category, it was found that the optimal number of controllers is the one that has the best cost/benefit relation. In other words, it is the maximum number of controllers for which adding another controller results in negligible delay reduction. Fig. 4.9, 4.10 and 4.11 illustrate the number of controllers found for each defined $\mathcal{D}_{req}$ in each network category.

For sparse networks (Fig. 4.9), a significant delay range –from 23 μs to 60 μs– is covered when using 5 controllers, note that this is the same number of controllers found for all methods. The benefit of using 5 controllers is a considerable delay reduction in comparison with using fewer controllers, and reduces the maximum delay (60 μs) by more than a third when using only one controller. This is an appreciable delay reduction, given that this network category has a low connectivity. Consequently, 5 controllers are considered as optimal.

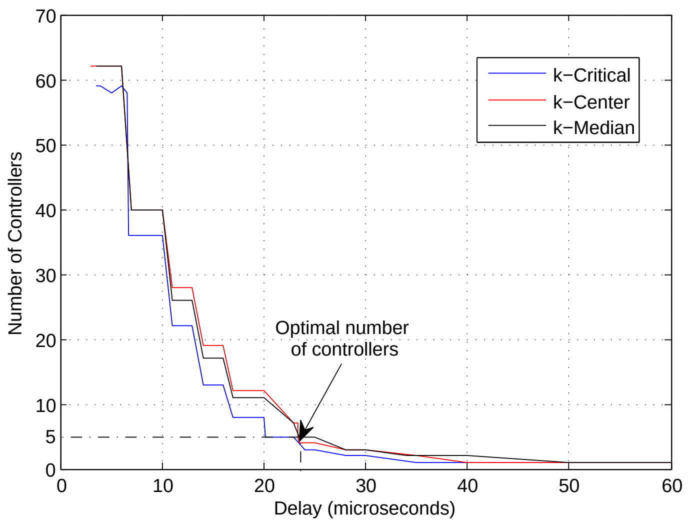


Figure 4.9: Number of controllers for all possible delay ranges in generated networks with sparse connectivity.

Fig. 4.10 shows the number of controllers found for a specific delay range in medium connectivity networks. As can be seen, the maximum delay reached (35 μs) is significantly lower than for sparse networks (approximately a half). This is a logical result, because networks with high connectivity have a small diameter. For this network category, it was found that the delay decreases when from 1 to 3 controllers are considered. But just considering 3 controllers halved the maximum delay time (17 μs) for all methods. Therefore, it can be considered that the optimal number of controllers to manage this network category is 3.

As was to be expected, for dense networks the maximum delay in a tree is reduced (27 μs) (Fig. 4.11) in comparison with the maximum delay for the medium network category.

Fig. 4.11 shows that there are two points for which the delay reduction is significant, i.e., 1 and 5 controllers. As can be seen (Fig. 4.11) when only one controller manages the network, trees with a $\mathcal{D}_{req}$ between 20 μs

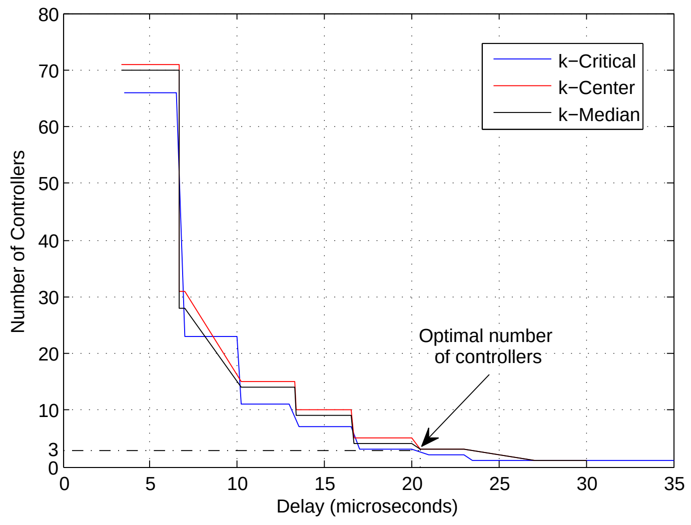


Figure 4.10: Number of controllers for all possible delay ranges in generated networks with medium connectivity.

and 26 μs can be built. Note that the delay reduction is negligible when 2, 3 or 4 controllers are considered compared to using just 1. A significant delay reduction is reached when 5 controllers manage the network, which reduces the maximum delay by half. In order to select the optimal number of controllers for dense networks, it was taken as a reference the delay reached for the optimal number of controllers selected for medium and sparse network categories, 17 μs and 23 μs, respectively. If this delay interval is considered as optimal (17 μs to 23 μs), one controller is a good selection in terms of delay for dense networks, since trees with 20 μs can be built. Using just one controller is not considered a problem, even though it implies that it has to handle the information from all the network, which may be inefficient. Some mechanisms may be considered for processing information efficiently (e.g., parallel processing) when just 1 controller has to manage the whole network.

The presented results indicate that using more controllers than the optimal number or using a poor controller location, reduces slightly the delay compared to the cost of adding more controllers.

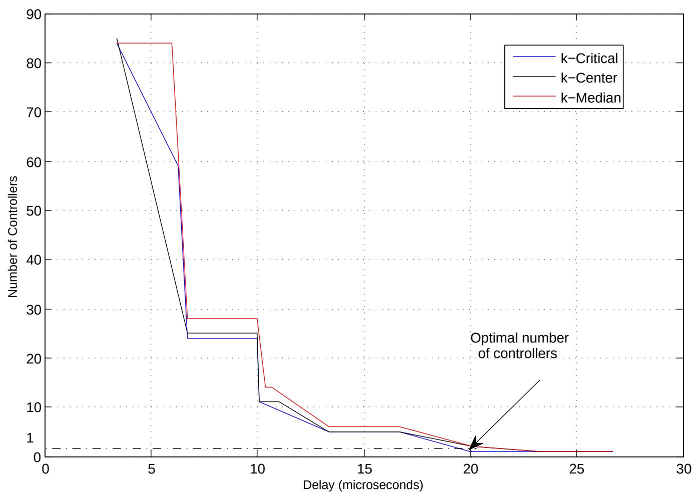


Figure 4.11: Number of controllers for all possible delay ranges in generated networks with high connectivity.

4.6.4 Analysis

In this section, the clustering methods are compared from the point of view of the characteristics of the tree topologies built from controllers selected for each one of the presented methods. This analysis is done for the optimal number of controllers found for each network category.

Fig. 4.12 shows the maximum number of switches managed by the controllers selected from each method. The maximum number of switches managed by controllers are averaged over the 100 random graphs in the respective categories. Fig. 4.12 shows that for sparse and medium connectivity networks the controllers selected by $k - Critical$ have better switches distribution than the other solutions. The results obtained for dense networks is not shown, because just 1 controller manages the whole network for all cases.

Fig. 4.13 shows the average tree-depth obtained from controllers selected for each method.

From Fig. 4.12 and Fig. 4.13 it can be observed that the allocation of switches to controllers affects the tree-depth. For instance, the trees created from the controllers found by *k-Median* have the worst switch distribution, and as a consequence have the longest paths. For the dense network category, when only one controller is required, $k - Critical$ has the best switch

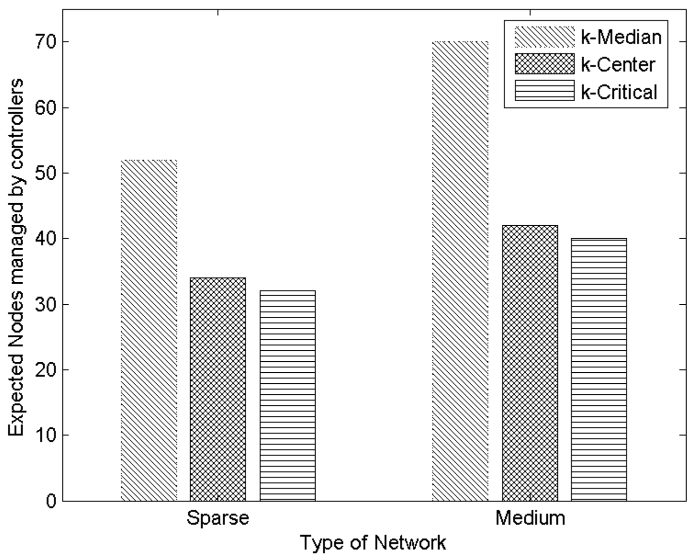


Figure 4.12: Average switches managed by controller.

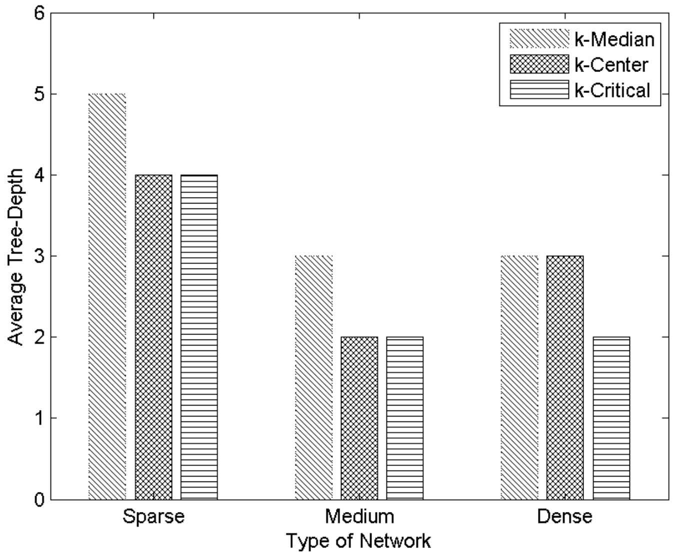


Figure 4.13: Average depth on control topology.

distribution among branches, and consequently the resulting trees are the shortest ones.

Fig. 4.14 shows the expected data loss on trees built from the selected controllers. For each network category, the data loss parameter is computed using Eq. (4.7). From the obtained results, it can be seen clearly the relation among switch distribution, tree-depth and data loss. The shortest trees have the lowest data loss, which is the case for trees created from the controllers selected by $\mathcal{k} - \mathcal{Critical}$. On the other hand, the longest trees have the highest data loss, as shown in the case of *k-Median* in Fig. 4.14.

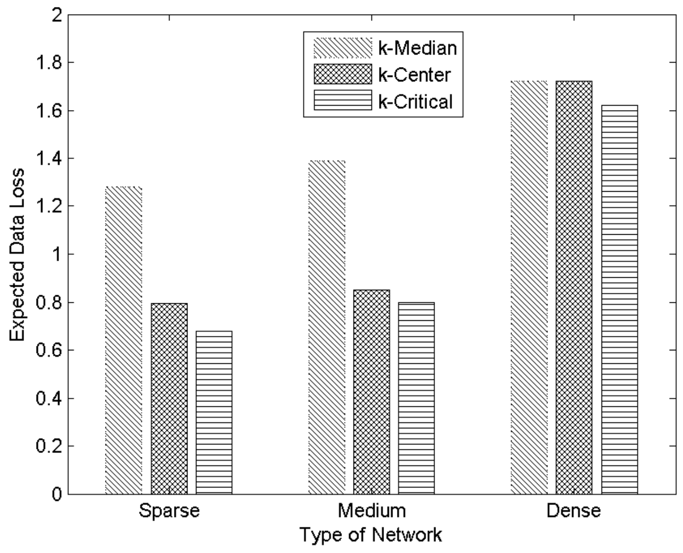


Figure 4.14: Expected data loss on randomly generated networks.

Fig. 4.15 shows the maximum delay of the trees obtained for each network category. As shown, $\mathcal{k} - \mathcal{Critical}$ improves the robustness for networks with high connectivity but, as a consequence, the delay of the branch cannot be improved in comparison with other methods. However, it is $\leq \mathcal{D}_{req}$. Thus, $\mathcal{k} - \mathcal{Critical}$ improves the tree robustness at the expense of delay.

Even though the trees created from controller placement by *k-Center* and $\mathcal{k} - \mathcal{Critical}$ have similar characteristics in terms of tree-depth and switches managed by controllers, the resulting trees built at controllers selected by $\mathcal{k} - \mathcal{Critical}$ reduces considerably the data loss for all the network categories, as it can be seen in Fig. 4.14. *k-Center* has good results on average, but not as good as $\mathcal{k} - \mathcal{Critical}$.

As shown, $\mathcal{k} - \mathcal{Critical}$ (heuristic version) makes the best controller selection in general. The good results obtained using $\mathcal{k} - \mathcal{Critical}$ are due to the criterion to select the candidate switches to be a controller placement, the

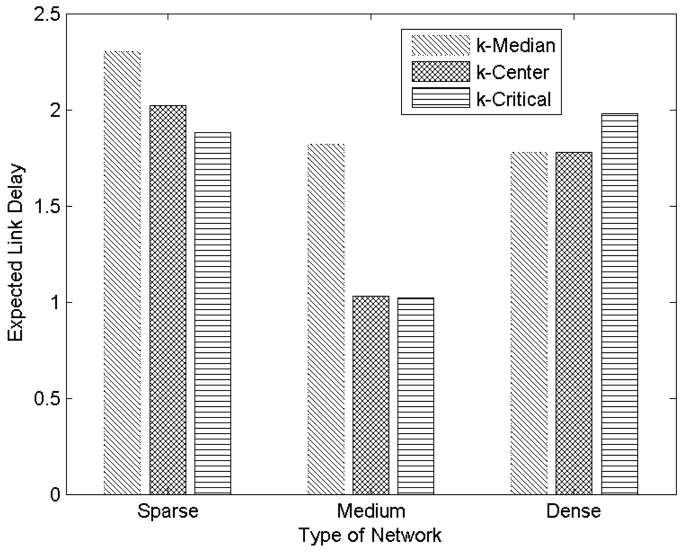


Figure 4.15: Expected link delay on control topology.

critical node criterion.

This is because, *k-Center* and *k-Median* do not obtain a good performance due to the initial criteria to select the controller is not efficient.

The computational complexity for each one of the controller placement solutions evaluated is $O(nk)$. In general, finding a controller requires checking the shortest distances of the n switches in the network. Therefore, the computational complexity for finding one controller is defined by $O(n)$. This process has to be repeated until k controllers are discovered, for all the approaches.

4.7 Conclusions

In this chapter a novel controller placement algorithm, called $k - \mathit{Critical}$, was proposed. It has been designed to find the number of controllers and their placement in any kind of network. Unlike other proposals, $k - \mathit{Critical}$ not only solves the controller placement problem, but also finds the required number of controllers. Controller placements selected by $k - \mathit{Critical}$ reduce the data loss in the case of failures, and also balance the load among them in terms of number of switches. This is because $k - \mathit{Critical}$ takes into account the network topology and selects the controller placements based on the critical nodes. These are the switches that limit the switch-to-controller assignation based on the control plane requirements.

The controller placements selected by $\mathcal{K}-\mathcal{C}ritical$ permit the construction of robust control planes in response to network disturbances. This is because the physical characteristics of the network are taken into account when selecting the controllers. This solution can also be used to find the best controller location when a network needs to be extended or the load needs to be reallocated.

$\mathcal{K}-\mathcal{C}ritical$ performance was compared to existing clustering approaches, *k-Center* and *k-Median*, through intensive numerical simulations. To do that, three different network categories were considered. The results clearly showed the benefits of the proposed algorithm when compared to these solutions. From the results obtained, it can be seen that the number of controllers, as well as their location, have a high influence on the network performance. Therefore, a poor controller selection can affect considerably the control network robustness, which affects the network operation (e.g., long recovery time after failures). On the other hand, using more than the optimal number of controllers can be inefficient and costly, because the delay improvement is negligible.

Based on the comparison of clustering approaches, it was found that the criterion used to select the first controller is crucial in defining the control plane topology and therefore its performance. A good controller selection allows the network to balance load and respond to events in the shortest time possible.

$\mathcal{K}-\mathcal{C}ritical$ provides a first approach to find the controller placements to design control planes for SDNs, where the controller placements are selected based on the switch-to-controller delay and the control plane robustness. In general, it was found that the optimal number of controllers depends on the physical network characteristics and the application requirements.

Part III

Contributions to build a robust control plane in SDN

Chapter 5

Evaluation of control plane robustness

Separation of control and data planes in SDN networks introduces new challenges in terms of the reliability of the communication between planes that needs to be addressed, as it is no longer directly linked. In SDN, if a switch-to-controller connection is interrupted, the affected switch will not know how to re-connect to the controller (i.e., a backup path to the controller does not exist) and will lose its forwarding functionality. The level of protection of a control plane depends mainly on the underlying network connectivity and the controller placements, since controllers can re-establish communication with the data plane, if backup paths exist. This chapter presents a new robustness metric that measures the level of protection in a control plane. This metric is defined as the number of switches in the network that are capable of reestablishing the communication with the corresponding controller after any link/switch failures.

5.1 Outline

Section 5.2 presents some characteristics of the recovery process when different backup paths are used to protect the control plane following switch/link failures. Section 5.3 introduces some approaches that select the controller placement based on the resilience it provides to the control plane. Section 5.4 describes in detail an approach called *Fast Failover* that evaluates each switch in the network to select the switch that maximizes the network resilience as a controller placement. Section 5.5 introduces a proposal of robustness metric to evaluate the resilience of a control plane in the presence of any link/switch failures. In Section 6.7.2 the characteristics defined to evaluate both approaches *k-Critical* and *Fast Failover* are defined, and Section 5.7 presents the analysis of the characteristics of the control planes built by each approach, including their resilience that is evaluated using the metric proposed in 5.5. Finally, in Section 5.8, the conclusions of the chapter are presented.

5.2 Network Resilience in SDN

To protect SDN networks in the presence of network failures, in OpenFlow [12], the *de facto* standard protocol for communications between the control and data planes in SDN, backup control paths are considered as an alternative to improve the control plane network resilience. When a network failure that affects the control plane occurs, switches disconnected can restore the communication with the control plane, according to the OpenFlow, (1) by allowing the switch to use an alternative path to the same controller through a predefined backup path, or (2) by allowing the affected switch to re-establish the communication with a different controller through a predefined control path.

5.2.1 Backup Control Paths

In OpenFlow, a switch may establish communication with a single controller, or may establish communication with multiple controllers. Having multiple controllers improves reliability, as a switch can continue operating if one controller or the controller connection fails. Following this idea, in [41], authors consider that switches can have a pre-configured backup path, so that if the primary control path does not work properly, the backup path (secondary) could be used. In [40], each switch must be guaranteed that an operational path towards any of the controllers it connects to, exists with at least a given probability.

Below, the recovery process executed by switches when considering different backup control paths is described.

Switch disjoint control path: When intermediate switches receive a notification about a failure in a control path it is part of, those switches do not have information about how to contact their controller, as they do not have forwarding information in their forwarding tables. In this case, the only switch that can initiate the control path re-establishment is the source switch. Therefore, it is necessary to announce the event to the source switch, which re-forwards the control messages through the backup control path to the controller. As a consequence, the recovery time mainly depends on the control path length, the primary and the backup path. In general, disjoint paths offer the maximum level of survivability, as these protect the primary control path from any link and switch failures. Fig. 5.1(a) shows the case when link $(3, C)$ fails and it is detected by switch 3. The fault announcement is then forwarded to source switch 1 that after receiving this message starts the backup path configuration process.

Link disjoint control path: If a link fails in the primary control path, the decision to re-route the control traffic is made locally at the switch that detects the failure. Consequently, this backup path offers only survivability on link failures in the primary control path. The recovery time depends on the backup path length, if it exists. Consider the network topology presented in Fig. 2.2, and Fig. 5.1(b) illustrates the link disjoint path used by switch 3 to re-establish the communication to the controller when link $(3, C)$ fails in the primary control path between switch 1 to controller C.

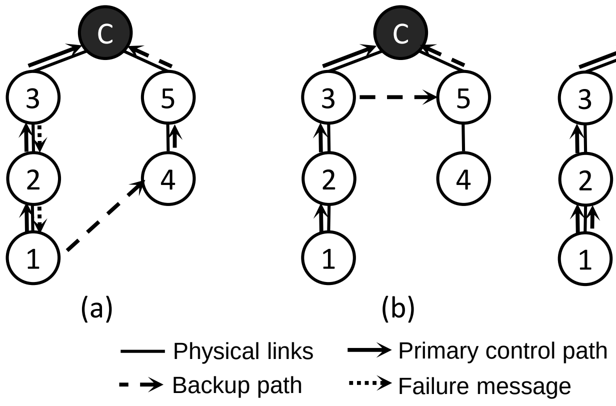


Figure 5.1: Representation of backup control paths.

Partial disjoint control path: when a switch detects a failure in its control path, it can re-establish the communication if the partial backup path is not affected by the failure. Ideally, each switch should have as backup path to its controller a switch disjoint sub-path, in order to protect a primary control path in the event of any link/switch failures. Such that, any switch in the primary control path can re-establish communication with the control plane. The level of survivability provided by this backup disjoint path improves by minimizing the number of switches and links shared with the primary control path. The highest level of survivability is obtained when each switch in the primary path has a switch disjoint sub-path to the controller. In this case, the recovery time a switch-to-controller communication is re-established depends on the backup path length. Fig. 5.1(c) illustrates the case when switch 3 in the primary control path fails. In this scenario, switch 2 detects this failure and starts forwarding the control messages to the controller through the backup control path $(2 - 5 - C)$.

Disjoint control path to another controller: a switch can communicate with another controller through a control path when it detects a failure in the communication with its controller. This backup control protects the network against controller failures and against switches without an alternative backup control path to their controller. In this last case, it is necessary to have extra communication between controllers to coordinate and exchange information about switch management, given that a switch can only be managed by a single controller at any time. This is the only backup control path that offers survivability in case of a controller failure. As a switch disjoint path, the recovery time depends on the length of both control paths, the primary and the backup.

The protection paths that reduce the use of the memory of the switches are: the switch disjoint paths and the disjoint control path to another controller, as the backup path is only configured/stored in the source switches. However, these are the protection paths that take more time in recover a switch-to-controller communication. The remaining protection paths have to be configured on each switch in the primary path.

5.3 Control plane resilience metrics

As described in Section 3.5, different resilience metrics have been defined to select the controller placements that permit the data loss to be minimized in occurrence of failures. These metrics can be classified based on whether or not they consider a protection path to recover a communication after it is interrupted.

For instance, in [44], [51] and [54] authors do not consider a protection path and therefore, the communication re-establishment to the control plane is not contemplated. In these approaches, the network resilience is defined in terms of the number of control paths affected due to a link/switch failure.

In [54], the switch selected as controller placement is the one that minimizes the average disconnection, where the average is computed by taking into account all pairs of switches that communicate with each other in a control path. The objective in [51] and [44] is to find the controller placement that maximizes the expected percentage of valid control paths in the presence of network failures.

The solutions that consider a protection paths such as, [40] and [41], define the network resilience in terms of the number of switches that can re-establish the communication to the control plane after a failure. In [40], the control plane resilience is defined in terms of the probability a source switch can be re-connected to the control plane after a link/switch failure. For this purpose, the controller placement selected is the one that satisfies a resilience metric value that is defined in terms of the number of switch disjoint control paths to the controller from each source switch. In order to satisfy the constrained value of the resilience metric for each source switch, authors also consider switch disjoint control paths from a switch to different controllers in the control plane.

In [41], the controller placements are selected such that the connection between the control plane and the forwarding plane is less likely to be interrupted when the adjacent switch/link of a switch in the control plane fails. It is considered that a good selection of the controller placement must result in a high number of reliable paths from the switches to the controller, in the sense that a large number of switches must have backup paths to the controller (all kinds of backup paths are considered in this approach). In this approach, the controller placement selected is the one that maximizes, in general, the number of backup disjoint control paths from the switches to their controller.

From the aforementioned approaches, the backup switch *disjoint control paths*, considered in [40], offers the best recovery approach. This is because, this approach provides the highest level of survivability to the control paths in the case of any link/switch failures in the network in comparison with the partial disjoint control paths considered in [41]. However, when a switch-to-controller communication is broken, it requires the failure to be announced to the source switch, which may take a long time. This time depends on the control path length.

Protection provided by *partial disjoint control paths* considered in [41] is limited. It only protects switches against failures in their adjacent switch or link in the control plane tree. For each intermediate switch between the source switch and the controller, a backup switch disjoint sub-path to the controller must be computed to avoid the backup path be affected by the failure in the primary path. Therefore, an inherent drawback of this protection backup path is the number of backup control paths that must be computed to protect a primary control path. The number of control paths that must be computed is defined by the length of the primary control path.

Note that, if only partial disjoint backup paths from each switch to controller are considered, the level of protection can be compared with a switch disjoint path. Considering disjoint backup paths also may reduce the number of switches that have to change their path to the control plane in the event of network damage, compared to a complete switch disjoint path.

Let consider the control plane represented by Figs. 5.1 when a failure in link (3,C) happens. Fig. 5.1(a). illustrates the process executed by the control plane to recover the communication of the switch 1 to the controller when considering a switch disjoint path, while that Fig. 5.1(b) and Fig. 5.1(c) show two different alternatives switch disjoint sub-paths to recover the communication to the controller. From this simple example can be seen that, when considering partial switch disjoint paths to the controller, the configuration time can be reduced (as the number of switches to be re-configured is lower than in the case of the backup switch disjoint path) as well as the traffic on the network.

In general, network resilience metrics proposed by authors consider protection paths. In some cases limiting the protection to a specific back control path (e.g., switch disjoint path as in [40]) and the protection to specific failures (e.g., failures in the adjacent switches/links in the control plane as in [41]). These considerations limit network protection, and therefore, the network resilience. Note that, one of the main conditions that must be found to provide a high protection level to the control plane is to choose the controller placements that maximize the level of disjointness of: 1) primary paths in a control plane and 2) switch disjoint backup control paths for each switch to the controller. Condition (1) reduces the number of switches affected by a link/switch failure and condition (2) provides the highest survivability level to the control plane. These conditions reduce the recovery time of a communication after a failure and also reduce the traffic generated on the network during the recovery process.

5.4 Fast Failover

This section describes in detail the approach proposed in [41]. This approach is taken as a reference point to propose a new robustness metric for the control plane in SDN networks. This is because, the approach has some desirable characteristics for the design of a resilient control plane, such as:

- it considers all protection backup paths to select the controller placement,
- the number of switches that must be configured to re-establish the switch-to-controller communication is reduced compared with a switch

disjoint control path, and therefore,

- it reduces the recovery time and number of fault messages in case a network failure.

In this approach, resilience is defined in terms of the number of switches protected against a disconnection to their parent switch in the control plane tree. A switch is considered protected against a failure if it has a backup path in the direction of the controller that is not affected by the same failure. The idea behind this solution is to locate the controller in a place so that the connection between the control and data plane is less likely to be interrupted, minimizing the disconnections when a failure occurs. In this approach, it is considered that the network is split and that just one controller is selected in each network partition.

In order to select the controller placement, a tree (T_n) rooted at each switch $n \in V$ is created to evaluate the resulting tree resilience. Based on the protection level of each switch in the tree and its number of downstream switches (d_s), a weight is assigned to each switch n, defined as $w(n)$. The switch ($n \in V$) with the highest protection metric value is selected as a controller placement for the split network. Considering the downstream switches in the weight computation allows a higher weight to be assigned to the switches closer to the controller that has protection. This is important given that switches with high protection near the controller can minimize the switches disconnected in case the connection to the controller is interrupted.

To compute the resilience provided by a controller placement n or the protection level of T_n, it is considered that the parent of a switch $s \in T_n$ is its immediate upstream switch p, and the set of the children of a switch are its downstream switches, d_s, except for the leaf switches. In order to compute the protection level of T_n, the switch protection for each switch is evaluated when considering the failure of its upstream link (s, p), as well as its upstream switch p.

The switch protection for each switch $s \in T_n$ is computed as follows:

1) Switch s is protected against a failure in its upstream link (s, p) if at least another upstream link $(s, p1) \in G$ exists, where $p1$ is not a downstream switch of switch $s \in T_n$, $p1 \notin d_s$. If switch s has link protection a weight $w1$ is assigned to it.

2) Besides, switch s is protected against a failure of its upstream switch (p) by $p1$, if $p1$ is not a downstream switch of switch p, that is $p1 \notin T_p(s)$. In this case a weight $w_1 + w_2$ is assigned to the switch.

If a switch n is neither protected against link nor switch failures, the weight assigned is zero, $w(n) = 0$. Algorithm 6 shows this process. After evaluating all of the switches $s \in T_n$ in the split network, switch n that maximizes the sum of weights is selected as controller. With this approach, when a switch detects a failure in its ongoing link or its upstream switch, it only has to change the path to the controller by going through another upstream switch without any instruction from the controller. This requires just one local change in the outgoing interface of the affected switch. This approach can be formally defined as follows:

$$P(s) = \begin{cases} 0 & \text{if } \exists\ s' \in V, s \notin T_p(s) : (s, s') \in E \\ 1 & \text{otherwise.} \end{cases} \tag{5.1}$$

According to this expression, a switch s is considered *protected* if $P(s) = 0$, and unprotected otherwise. Note that this protection metric is only effective, in practice, against failures in the parent switch; but the fact that a switch is *protected* according to this definition does not imply that it is resilient with respect to further failures in the corresponding tree, *e.g.* if the failure occurs at the 2-hop parent. Therefore, this protection metric is not sufficient to determine the network robustness with respect to any link/switch failure in the network. This metric provides an idea about how many switches have a certain local (1-hop) protection, but is not informative about the effective level of protection. Hence, can not be used to directly infer the number of switches that can re-establish communication with the controller in case of any link/switch failure in the network.

Algorithm 6 Fast Failover approach

```
Require: $G = (V, E, \delta)$, $T_n$ (trees rooted at $n$, $n \in V$)
  for each $n \in V$ do
    $T_n = (V_{T_n}, E_{T_n}) \leftarrow$ shortest path tree (w.r.t. $\delta$) rooted at $n$ $w(n) \leftarrow 0$
    for each $s \neq n$, $s \in V$ do
      $w(n) = |T_s| - 1$
      if $\exists\ r \in V : (s, r) \in E, r \notin d(s)$ then
        $w(n) \leftarrow w(n) + w_1$
        if $\exists\ q \in V : (s, q) \in E, r \notin d(p(s))$ then
          $w(n) \leftarrow w(n) + (w_1 + w_2)$
        end if
      end if
    end for
  end for
  return arg $\max_{n \in V} w(n)$
```

5.5 Towards a network robustness metric

This section proposes and formalizes a complete measure for the robustness of SDN control planes. This metric quantifies the number of switches potentially affected by any network failure, and their possibility to recover from them with a 1-hop reconnection by using a backup path to the controller. The resilience metric proposed in [41] is a valid first-order estimation of robustness. It implicitly assumes that all possible failures are equivalent, as it does not take into consideration the different impact of failures in the network. Given that when there is a failure in an SDN network, the number of affected switches does not only depend on their local protection at 1-hop (with respect to their immediate parent), but on their local protection with respect to the failing switch and also on the protection provided by upper switches affected by the failure. Consider the situation in Figure 5.2, where A is assumed to be controller. For instance, note that G is a local 1-hop protected switch, according to expression (5.1): if E fails, it can still get reconnected to the control plane tree via D, which is a neighbor of G in the network. However, G is not protected against a failure in B, given that its possible path through D also involves B. Note also that G would, in practice, be "protected" against a B failure if there was a link in the network between E and F (not shown in Figure 5.2) and its parent E could hence reconnect to the control plane tree via $\{F, C, A\}$.

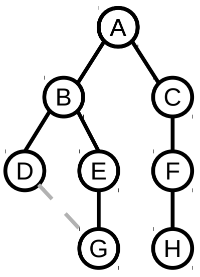


Figure 5.2: Average switch distribution for each network category.

5.5.1 Robustness as generalized switch protection

In order to define an accurate metric for control plane robustness, a generalized local switch protection parameter is introduced here, defined in Eq. (5.2). For this purpose, consider that the control plane or management tree of a Software Defined Network (SDN) is induced by the selection of a controller $n \in V$ and defined as a subgraph $T_n \subset G$, which set of vertices and edges are denoted by V_{T_n} and E_{T_n}, respectively.

Let T_x be the subtree rooted at switch x that consists of the set of branches or paths $p(x, y)$, where y is the leaf switch of each branch at T_x. A switch $s \in T_x$ has *generalized local protection* against a failure in x if it has a connection with a switch s' in the direction of the controller that is not in T_x.

$$P'(s) = \begin{cases} 0 & \text{if } \exists\ s' \in V, s \notin T_x(s) : (s, s') \in E \\ 1 & \text{otherwise.} \end{cases} \tag{5.2}$$

In this context, given a control plane tree T_n (rooted in controller n), the switches primarily affected by a failure in switch x are the downstream switches of x, that is, $|T_x|$. The fraction of these switches that are locally protected against this failure is then:

$$\frac{1}{|T_x|} \sum_{s \in T_x} P'(s). \tag{5.3}$$

Given a controller n, switches y depending (downstream) on a locally-protected switch s^* against a failure in x are also protected, as the reconnection of s^* allows them to communicate to the controller by way of the path $p(y, s^*) \cup p(s^*, n)$. The binary function

$$F(z) = \prod_{s \in p(z,x)} P'(s) = \{0, 1\} \tag{5.4}$$

indicates whether a switch $z \in T_x$ is protected against the failure of x (value 0) or not (value 1). Note that, for a switch z in T_x to be protected, it is only necessary that one of the ascendant switches of z in the path towards x is locally protected, or z is locally protected itself. In other words, a locally protected switch induces protection for any downstream switch. Taking all these cases into account, the fraction of protected switches (locally protected or indirectly protected by upper switches) against the failure of x in T_x can be computed as follows:

$$R_{x,n} = \frac{1}{|T_x|} \sum_{z \in T_x} F(z) = \frac{1}{|T_x|} \sum_{z \in T_x} \prod_{s \in p(z,x)} P'(s). \tag{5.5}$$

In order to discover the control plane robustness in a control plane tree $T_n \subset G$, rooted at controller n, for each possible switch failure in the network, the number of affected switches and the number of affected unprotected switches are computed, that is, those switches which can not re-establish communication with the controller n. Algorithm 7 describes the computation of the robustness of a control plane. Given a controller n and the control plane tree T_n (the shortest tree rooted at n), each switch $x \in V_{T_n}$ is considered to fail (line 1). To compute the number of switches affected by this failure, a sub-tree rooted at switch x is built and for each switch s in the subtree a backup path to the controller is sought (lines 3-11). As a result, the switches that can not re-establish the communication to the controller through an adjacent switch are obtained. Finally, the number of switches protected against switch/link failures is computed (line 12).

Algorithm 7 Evaluating Control Plane Robustness

```
Require: $G = (V, E), n \in V, T_n \subset G$
 1: for each switch-to-fail $x \neq n, x \in V_{T_n}$ do
 2:     $T_x \longleftarrow$ subtree rooted at $x$
 3:     for each path $p(x, y) \subset T_x$, where $y$ is a leaf switch do
 4:         for each switch $s \in p(x, y), s \neq x$ do
 5:             $P(s) \longleftarrow$ Eq. (5.1)
 6:             for each switch $z \in p(x, s)$, where $z \neq x$ do
 7:                 $P'(z) \longleftarrow$ Eq. (5.2)
 8:             end for
 9:             $F(s) \longleftarrow \prod_{z \in p(x,s)} P'(z)$
10:         end for
11:     end for
12:     $R_{x,n} \longleftarrow \frac{1}{|T_x|} \sum_{s \in T_x} F(s)$
13: end for
14: return $\mathbb{E}_n\{R_{x,n}\}$
```

5.6 Evaluation

This section evaluates two approaches, *Fast Failover* and *k-Critical*, in terms of the characteristics of the resulting control planes created from the controllers selected by them.

In addition, the robustness of these control planes is evaluated by using the robustness metric proposed in the last section.

5.6.1 k-Critical and the restriction to k=1

In its most general form, *k-Critical* selects, for a given SDN topology and a maximum controller-to-switch delay, k switches to be controllers and generates their corresponding control plane trees, in a way such that the maximum delay between a controller and a leaf switch belonging to its tree is smaller than the maximum requested delay.

According to the results presented in the last chapter, the generated control plane rooted at each controller selected by *k-Critical* is homogeneous (*i.e.*, the number of switches per branch is balanced between different branches) and robust (*i.e.*, the loss of control data due to switch/link failures is low).

Note that unlike the *Fast Failover* algorithm, *k-Critical* determines the set C of controller candidates based on their network connectivity. Switches selected as candidate switches to be a controller are those that fulfill the $\mathcal{D}_{req}$ requirement: this is the maximum switch-to-controller delay within the shortest paths tree (computed with respect to delay metric). The candidate switch with the best tradeoff between delay and load distribution is then selected as a controller.

Unlike other approaches, *k-Critical* does not define a resilience metric to select the controller placement, it only combines the advantages of high switch connectivity and small diameter networks to get a network topology that reduces the data loss in case of failures. As it selects controller placement that reduces the diameter of the control topology as much as possible, it also reduces the switch-to-controller transmission times along with the traffic load on links.

With respect to switch connectivity, *k-Critical* aims at distributing homogenously the switches on the branches to reduce the data loss, by reducing branch length and the number of downstream switches at each switch of a control plane or tree. In consequence, the control plane built from any selected controller tends to be wide (*i.e.*, switches have many children) near the controllers and narrow (*i.e.*, with decreasing number of children) for switches closer to leaf switches.

In order to compare the resilience of the control planes created from the controller selected by *Fast Failover* and *k-Critical*, this latter approach must be restricted to $k = 1$, in which only 1 controller is selected by the algorithm. This is due to fairness reasons, as *Fast Failover* relies on the selection of a single controller (see Section 5.4).

5.6.2 A Note on Complexity

Algorithm complexity is defined by the number of operations required to select a controller within a network graph G. Both algorithms build a delay-based shortest path tree for some switches in the network, and then execute different heuristics on the computed trees. In the case of *Fast Failover*, the complexity of this additional heuristic is quadratic, given that for each switch n in the network $(n-1)$ additional operations are performed, thus $n(n-1)$ operations. Consequently the complexity of these additional operations is $O\left(n^2\right)$. In the case of *k-Critical*, the number of additional operations depends on the size of the set of candidate switches. In the worst case (that is, all switches in the network are candidates), the complexity of *k-Critical* is defined by $O\left(n\right)$. This is because, in this analysis *k-Critical* is restricted to $k=1$, in which only 1 controller is selected by the algorithm.

5.7 Simulation and Results

5.7.1 Setup

Both algorithms for controller selection and control plane tree construction (*Fast Failover* and *k-Critical*) are implemented in MATLAB, and their performance is evaluated over a set of randomly generated graphs of fixed size (100 switches) with three levels of switch connectivity:

- sparsely connected graphs.
- medium connectivity graphs.
- strongly connected graphs.

Table 5.1 shows the main characteristics of the generated networks for each category. For each switch connectivity level, results are averaged over 100 graph samples. The adjacency matrices for these graphs are generated by using the Gephi platform [81]. Depending on the value of the network wiring probability (see Table 5.1), Gephi generates networks with close-to-Gaussian distribution of switch degrees (see Fig. 5.3). Table 5.1 summarizes the main characteristics of each network category; Figure 5.3 displays the average distribution of switch degree for each category.

Table 5.1: Simulated networks.

Connectivity	Switch degree interval	Wiring parameter
Sparse	$[1,5]$	0.036
Medium	$[1,10]$	0.050
Strong	$[1,50]$	0.18

For each edge, a random link length is assigned following a uniform distribution within $[1,10]$ km. Delays associated to network links depend on

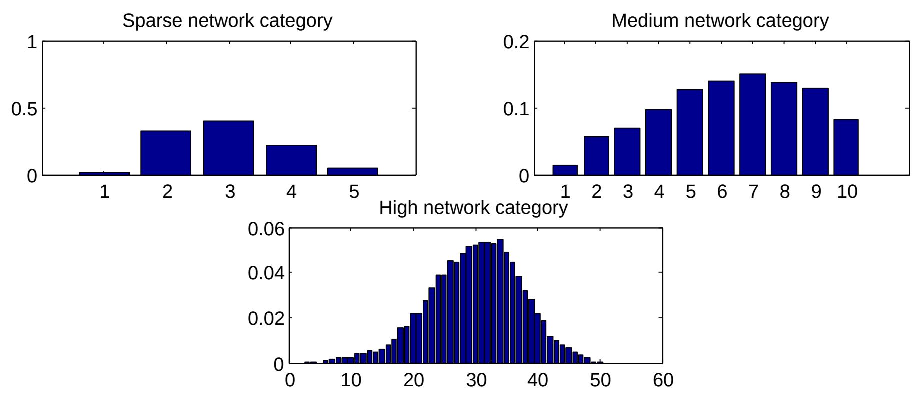


Figure 5.3: Average switch distribution for each network category.

their length, assuming typical optical fiber rates and using the following expression:

$$Delay = \frac{\mathcal{D}_{link}(m)}{\mathcal{V}_{light}(m/sec)}.$$

Both algorithms (*Fast Failover* and *k-Critical* for $k = 1$) are executed over these graphs: controllers are selected and the corresponding delay-based shortest path trees rooted at them are computed. This setup allows the properties of both algorithms to be compared by observing the structure and impact of switch density (measured in neighbour switches) over the induced control plane tree. For the case of *k-Critical* algorithm, the delay value ($\mathcal{D}_{req}$) used in the simulations was high enough to select just one controller ($\mathcal{D}_{req} = 0.20 secs$) for all network categories.

5.7.2 Results

Delay

Fig. 5.4 and Fig. 5.5 display, respectively, the average and the maximum delay between controllers and switches in the generated control plane trees for *k-Critical* and *Fast Failover*. It can be observed that *k-Critical* produces substantially "faster" trees that *Fast Failover*. This effect, which is particularly relevant in sparse SDNs, implies that the dissemination of control information throughout the SDN is performed, both for an average switch and for the most "distant" switch from the controller, within shorter time intervals. This is an expected result, due to the fact that *k-Critical* takes into consideration the $\mathcal{D}_{req}$ requirement and explicitly discards as a controller candidate any switch unable to provide a maximum delay in its tree smaller than $\mathcal{D}_{req}$. As shown in Section 5.4, the *Fast Failover* heuristic selects a controller based only on topological (switch connectivity) considerations.

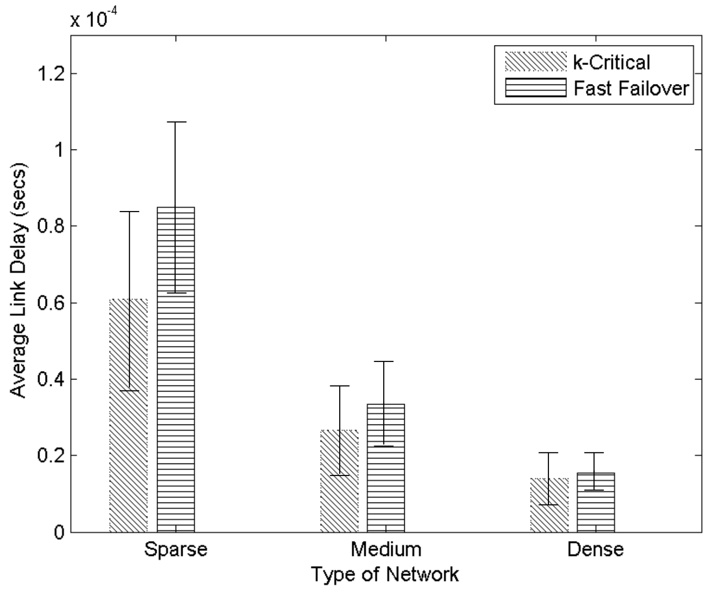


Figure 5.4: Average link delay.

Topology

k-Critical also produces shorter trees (in hops) than *Fast Failover*, that is, trees for which both the longest branch (Fig. 5.6) and the average branch (Fig. 5.7) have less hops than the trees generated from controller selected by *Fast Failover* algorithm. Although the advantage of *k-Critical* over *Fast Failover* naturally decreases with switch density (as, in a full mesh SDN, all trees would have a length 1), it is still observable for dense SDNs.

The fact that tree branches are shorter in *k-Critical* than in *Fast Failover* also implies that a random switch x has, in average, less downstream switches (*i.e.*, switches for which the control path towards the controller traverses x) in the first case (*k-Critical*) than in the second. These downstream switches are the switches affected by the failure of the switch (including those that may recover due to some sort of protection). The average number of downstream switches for both algorithms over different network densities is displayed in Fig. 5.8.

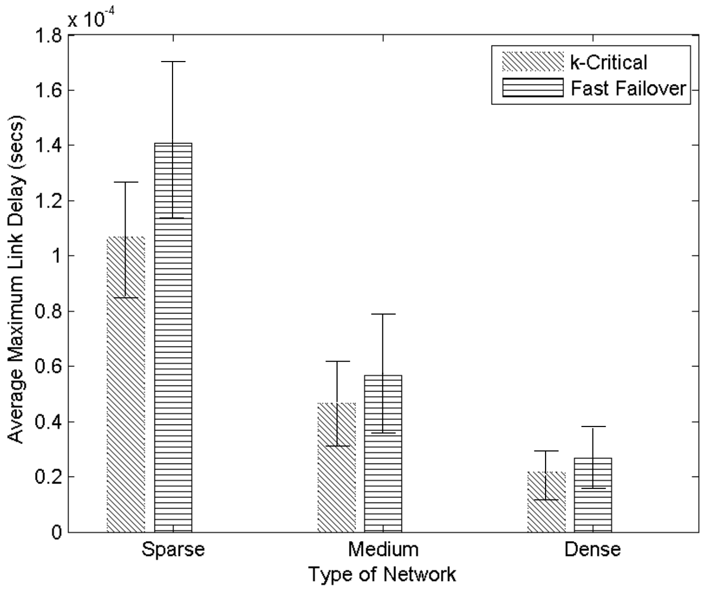


Figure 5.5: Maximum controller-to-switch delay in control plane trees.

The average distribution of switches in function of the number of downstream "attached" switches in the corresponding control plane trees, for each algorithm in the sparse network category, is shown in Figs. 5.9. Although the represented CDFs are very similar, only $2,5\%$ of the switches in *k-Critical* trees have more than 10 downstream switches, while this percentage grows beyond 4% in the case of *Fast Failover*: trees in the first case are thus more homogeneous (in the sense that the number of downstream switches, and therefore the control traffic load, are distributed more equally among SDN switches) than in the second. Results for other categories are consistent.

Robustness

Fig. 5.10 shows the percentage of "locally 1-hop protected switches" in control plane trees generated by *k-Critical* and *Fast Failover*. This is, as detailed in Section 5.4, the fraction of switches that could reconnect with the control tree in case of failure in their upstream link/parent. Although *Fast Failover* is designed to optimize this specific metric, it can be observed that its performance in this aspect is very similar to the one achieved by *k-Critical*.

As it was argued in Section 5.5, the metric of "locally 1-hop protected

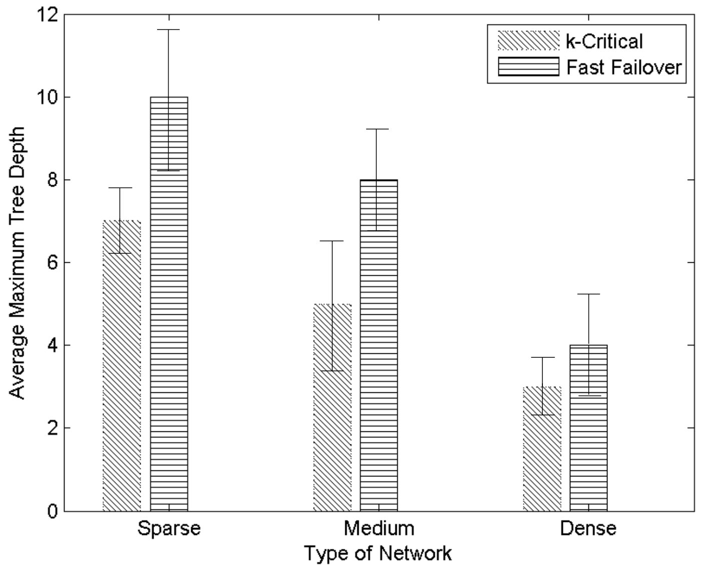


Figure 5.6: Average maximum tree depth, the depth of a tree corresponding to the maximum length of a branch.

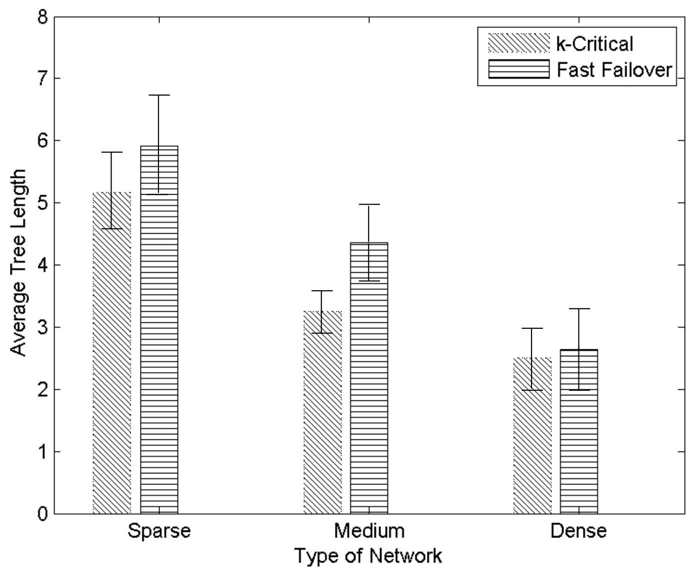


Figure 5.7: Average tree length.

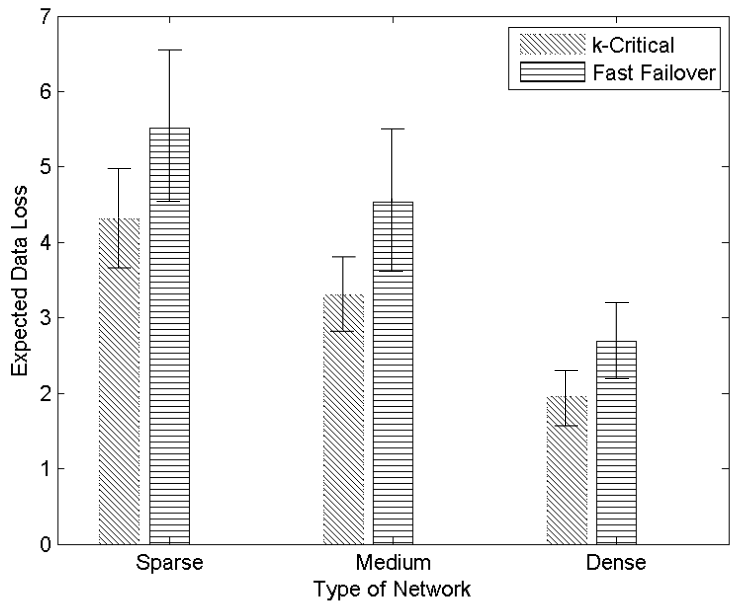


Figure 5.8: Average number of downstream switches/switch on the tree. *These are the affected switches in case of failure of the corresponding switches.*

switches" is not sufficient to evaluate the robustness of a control plane tree. A generalization of this metric is thus proposed to measure, for a particular tree built over an SDN and a particular switch failure, the fraction of downstream switches that could reconnect to the control tree. Fig. 5.11 shows the complementary of this metric, that is, the average (computed over all possible link/switch failures) of the fraction of downstream switches that *can not* recover from the failure, in *k-Critical* and *Fast Failover* control plane trees. Note that this metric behaves as expected when the network density increases – SDNs with higher connectivity are, in general, more robust than those with low connectivity, due to the fact that more links are available, and thus more redundant paths can be leveraged in case of switch failures.

It can be observed that, in the light of this generalized robustness metric, *k-Critical* ($k = 1$) selected controllers produce more robust control trees than *Fast Failover*, in particular in sparse SDNs – that is, in networks where algorithm robustness is more necessary. As the SDN switch density increases, both algorithms tend to provide a similar level of tree robustness.

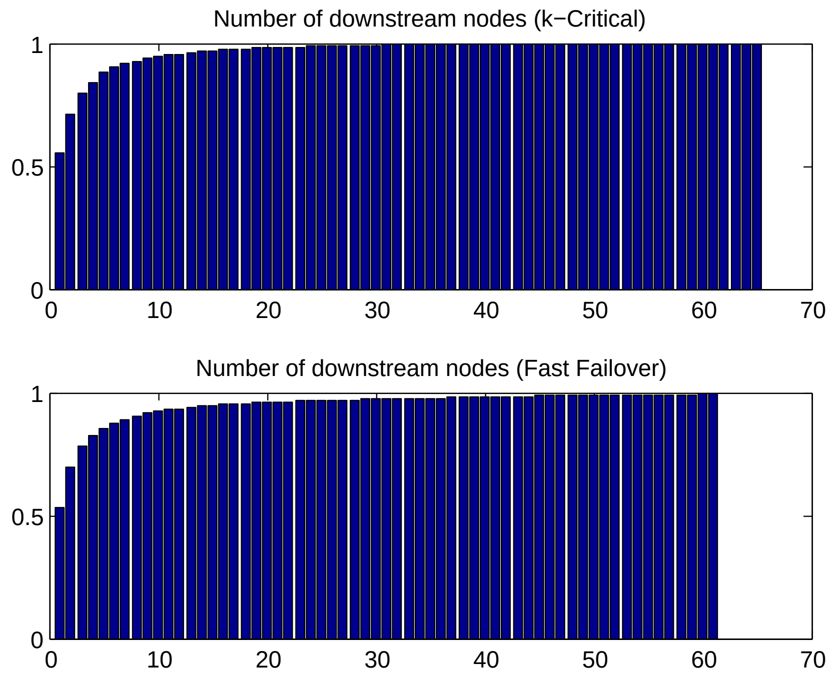


Figure 5.9: Experimental Cumulative Distribution Function (CDF) for the number of downstream switches, for *k-Critical* (top) and *Fast Failover* (bottom), for sparse SDNs.

5.8 Conclusions

The problem of controller placement in Software Defined Networks (SDN) remains open. Some of the most prominent heuristics that have been recently proposed stress the relevance of resilience and robustness in the resulting control plane tree for controller selection. While this intuition remains fundamentally correct, the metric of robustness in this context needs to be further developed.

This chapter examines in particular the *Fast Failover* heuristic, that is aligned with the fast failover mechanisms included in the specification of OpenFlow. In order to evaluate the control plane robustness, in this chapter a robustness metric was proposed and formalized. Based on this metric, the *Fast Failover* heuristic is compared to a simplified version of *k-Critical*, a heuristic designed to select the controller placements that reduces the controller-to-switch delays in SDNs, by way of extensive graph simulations.

First results of these simulations show the different characteristics of con-

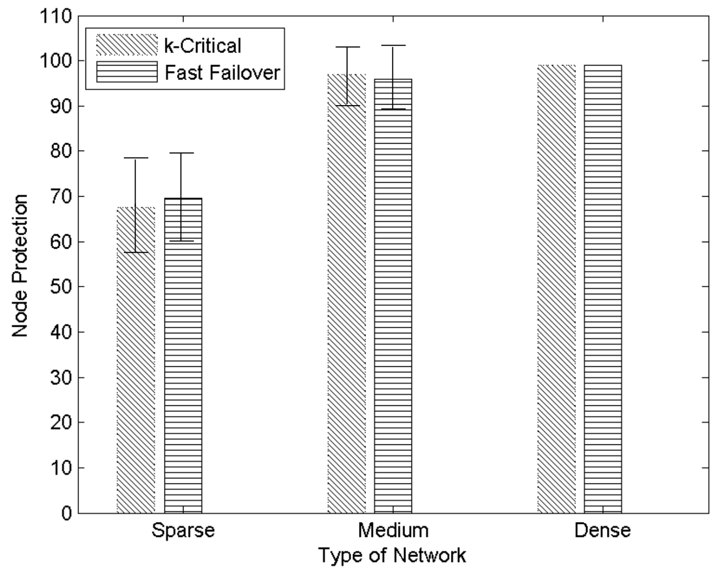


Figure 5.10: Switches locally protected against failures in immediate upstream link/parent.

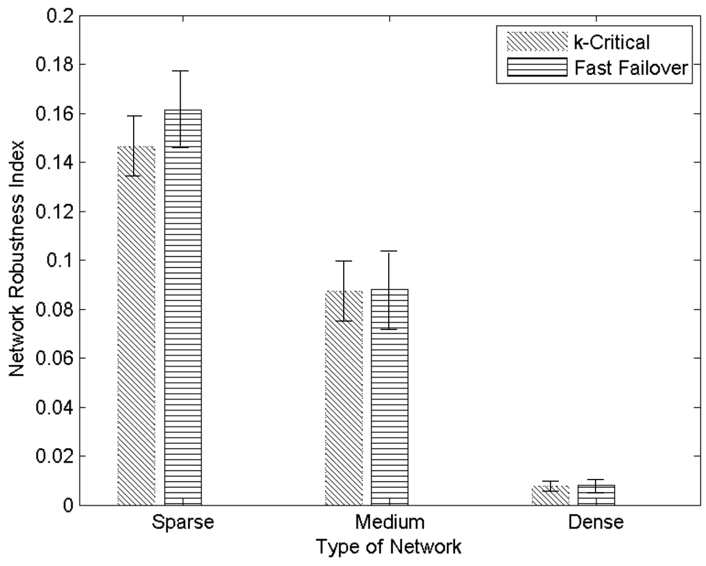


Figure 5.11: Network robustness index. *0=robust, 1=non-robust.*

trol plane trees produced by *Fast Failover* and *k-Critical*: this last heuristic selects controllers such that their delay-based control plane trees are shorter (in hops) and wider than those created by the controllers found by *Fast Failover*. When the underlying network is not densely connected, these differences imply that *k-Critical* trees are more robust against failures, according to the proposed metric, than *Fast Failover* trees. Interestingly, although *k-Critical* has not been explicitly designed to optimize any robustness metric, it shows a very similar performance to *Fast Failover* in terms of the underlying robustness metric. In summary, *k-Critical* shows a good performance for the aspects that are related to its own design (delay optimization), achieves a similar performance to *Fast Failover* in terms of the specific metric (local 1-hop protection) and is able to produce, in general, control plane trees shorter, wider, more homogeneous and, therefore, more robust to random failures than those produced by *Fast Failover*.

Beyond the specific aspects corresponding to the comparison between these two heuristics, results shown in this chapter, while preliminary, indicate that there is substantial room for improvement and optimization of controller placement heuristics in SDN, in particular, in what concerns the definition of an accurate and meaningful control plane robustness metric.

Note that the robustness control plane metric proposed can also be used as a constraint to select the controller placements in a SDN network. This permits the selection of the controller placement that provides the highest switch protection against any switch/link failures in the network.

The results of the performed evaluation show clearly: 1) the implications of the controller selection in the SDN performance, and 2) the control plane topology induced by *k-Critical*, even in its simplified version ($k = 1$), is less prone to failures, more robust according to the proposed metrics and more homogenous than those computed by *Fast Failover*.

The robustness metric presented in this chapter can also be used when considering a backup control path to any controller in the control plane.

Chapter 6

Resource discovery for SDN networks

Communication protocols for SDN networks (e.g., OpenFlow, ForCES, etc.) do not support any dedicated functionality for topology discovery, whose implementation is the responsibility of the controller. Furthermore, there is not official standard that defines a topology discovery method for SDNs. As an alternative to managing SDN networks in a distributed way, this chapter introduces a set of mechanisms that includes: i) a Resource Discovery Protocol through which the controllers discover a partial network topology, ii) a Network topology discovery mechanism through controllers exchange information between them, discovering the whole network topology, iii) an Allocation of switches to controllers, this mechanisms let the controllers allocate the switches between them, based on a defined parameter, finally iv) a mechanism that updates the network topology on controllers. The Resource Discovery Protocol was published in IEEE communication letter [82].

6.1 Outline

Section 6.2 presents briefly the challenges in managing a SDN network and the mechanisms required to maintain the network topology updated on the control plane. Section 6.3 describes the SDN Resource Discovery Protocol and its complexity. Section 6.4 describes a mechanism that allows the controllers to exchange network topology information, while Section 6.5 outlines a mechanism that allows the controllers to re-distribute the network management between them. Section 6.6 introduces a mechanism that maintains updated the network topology on the control plane. Section 6.7 is devoted to the analysis of the protocol through simulations. Finally, Section 6.8 presents the conclusions.

6.2 Network management in SDN

One of the main ideas behind the separation between control and forwarding planes in SDN is to simplify and make more efficient the network management and therefore, its operation. In SDN networks, the intelligence is delegated to a centralized controller, which makes network decisions based on the network state. For scalability purposes, the switch management is distributed between controllers, so that each controller is responsible for configuring the forwarding tables of a set of switches. Controllers can have information about the whole network topology in order to attend and address the requests coming from the switches that they manage. For this purpose, an indispensable condition is that the controllers have information updated about the underlying network topology.

In this context, there are different challenges that must be addressed, such as: **1)** the discovery of the network topology , **2)** the allocation of switches to controllers and **3)** the network topology update on the control plane.

In Section 3.8, some approaches to update the network topology in SDN are described, which assume that:

- controllers have information about the switches of the network topology,
- controllers have information about the set of switches they have to manage,
- the controllers based on the aforementioned information are capable of configuring a switch-to-controller path or control path,
- network information is updated through the LLDP protocol.

Given the capacities associated with the controllers in SDN, it is realistic to believe that the controllers are capable of discovering the network topology and selecting the switches that they should manage based on a specific criterion (e.g., delay, robustness, etc.).

6.2.1 Formulation

Consider a physical network that is modeled using a graph denoted by the tuple $G = (V, E)$, where V is partitioned into the set of switches n and controllers C, and E is the set of links. Then, $C = \{C_1, \ldots, C_k\}$ is the set of distributed controllers that defines the control plane. The set of adjacent neighbours of each switch $i \in V$ is defined by N_i. The control plane tree created by each controller C_i is defined as $G_{C_i} = (n_{C_i}, e_{C_i})$, where $n_{C_i} \subseteq n$ is the set of switches managed by the controller $C_i \subseteq C$, and $e_{C_i} \subseteq E$ is the set of links that defines the control paths. The global control plane is defined by the set of control planes built by the controllers, $G_C = \{G_{C_1}, G_{C_2}, ..., G_{C_k}\}$. It is assumed that each switch has information about the identifier of all its neighbours switches as well as the interfaces through they can communicate each other.

6.3 Resource Discovery Protocol

The process of creating the control plane is executed in two phases, the *forwarding* phase where the controllers announce their presence and the switches that start the creation of the control paths (leaf switches) are discovered, and the *backward* phase where the switches decide which switch to join in direction of a controller, thus creating the control plane. Below, the in-band control messages used by SDN-RDP are described.

- *Announcement messages* (AN): Controllers and switches announce events through these messages. The controllers announce their presence to the network (message Type 0), and the switches announce network changes or events (e.g., broken links, switch failures, etc) to the controllers (message Type 1).
- *Response messages* (RES): These messages are used to create the control paths. Switches send a join message (RES_JOIN) to join to a parent switch, and a leave message (RES_LEAVE) to announce to their neighbours that the switch has already been joined to a switch.
- *Improved Announcement messages* (IAN): By sending these messages the switches can announce to their neighbours that a nearer path to a controller has been discovered.

6.3.1 Forwarding Phase (FP)

To build the control plane each controller advertises its presence by forwarding an *AN message* to all their N_i adjacent switches. This message contains information concerning the controller identifier, the sender switch identifier and its latency to the controller. It is assumed that switches can discover their adjacent neighbours and the latency to them by exchanging control messages (e.g., LLDP). Upon receiving the first *AN message*, the switches update the packet fields and forward the message through every outgoing interface except that which the message was received. This process is repeated for each switch, consequently the *AN messages* are distributed all over the network.

The switches that receive at least one *AN message* through all their neighbours stop forwarding this message, these switches are called *discovered switches.*

Definition 6.1 (Discovered switches (D_n)): these switches discover the latency to a controller through each one of their (N_i) neighbours. Let $D_n = \{D_1, D_2 \ldots\}$ be the set of discovered switches in G. These switches based on the known information can find out if the latency of any N_i switch can be improved via another neighbour.

When a switch $i \in D_n$ discovers that through switch $k \in N_i$ the known latency of switch $j \in N_i$ to a controller can be reduced, it forwards an IAN message to switch j announcing that a better path to a controller has been found. This message is retransmitted back hop-to-hop as long as it encounters a switch that can not improve its latency through this path. Upon receiving this message, the switches update the delay and sender identifier fields in the message. The *discovered switches* that cannot improve the latency of any neighbour to a controller are called *leaf switches.*

Definition 6.2 (Leaf switches (L_n)): the set of leaf switches is defined as $L_n = \{L_1, L_2, \ldots\}$, where $L_n \subseteq D_n$. These switches determine the minimum latency possible to a controller $C_k \in C$, and initiate the control plane construction process (backward phase).

On the other hand, switches that have not received one *AN message* through all their neighbours wait for the answer to the *AN message*s sent through each interface. This answer can be an IAN message or a RES message. The switches that become *discovered switches* can make a decision as explained below in the backward phase. In this context, the IAN message has 2 purposes:

- discover the leaf switches, and
- discover the nearest controller from each switch.

Given that the controllers are not synchronized, it is considered that the controllers that have not started the network discovery process initiate the FP when receiving one *AN message.* In this way, the protocol guarantees that the switch management is distributed among all the controllers, besides simplifying the SDN-RDP resolution. Algorithm 8 describes the process executed by each switch during the forwarding phase.

Algorithm 8 Forwarding Phase for switch i.

```
Require: At least one controller has started the $FP$, and switch $i$ knows its adjacent neighbours and the delay to them, $d(i,j)$
  $d(k, C_k) = 0 \leftarrow$ delay from each switch $k \in N_i$ to a controller $C_k \in C$
  if it has received one $AN$ or a $RES$ message then
    update local delay information, $d(k, C_k)$
    if this is one $AN$ message and this is the first $AN$ received then
      forward the message through every outgoing interface except those which:
      i) the message was received and,
      ii) connect to neighbours whose identifiers were included in this message
    else
      if it has received at least one $AN$ or one $RES$ message from all its neighbours then
        change state to discovered switch
      end if
    end if
  else
    if it has received an $IAN$ message or state is discovered switch then
      for each switch $k \in N_i$ from which an $AN$ or $IAN$ message was received do
        if $d(k, C_l) + d(i,k) < d(j, Cm) + d(i,j)\ \forall j \in N_i,\ j \neq k$ where $j$ are the switchs from which an $AN$ message was received then
          send an $IAN$ message to switch $j$
          update local delay information, $d(j, C_i)$
        else
          change state to leaf switches
        end if
      end for
    end if
  end if
```

6.3.2 Backward Phase (BP)

In this phase the switches decide to which switch to join, leading backwards in the direction of a controller, and as a result, a control plane G_{C_i} on top of each controller $C_i \subseteq C$ is created. This process starts from the *leaf switches* which send a RES_JOIN message to their neighbour switch that is nearest to a controller and a RES_LEAVE message to the rest of the adjacent switches, if needed.

In order to ensure the discovered control paths are the shortest ones and also that all switches are managed, only the switches that:

i) have received a response message to the sent *AN messages*, and

ii) can not improve the latency of any switch $k \in N_i$,

can decide which switch to join in the direction of a controller. Therefore, the switches that satisfy these conditions have to update their identifier into the received RES_JOIN message and forward it to their best neighbour. Consequently, the RES_JOIN messages converge on the controllers, which receive (at most) a join message from each N_i adjacent switch.

There are two situations that prevent condition (i) can be fulfilled. These are:

- forwarding conflicts, and
- network failures.

These cases, as well as the proposed solution, are explained below.

Forwarding Conflicts

As switches work asynchronously, during the forwarding phase some switches i and j can receive and forward over the same link (i, j) an *AN message.* This fact causes that the involved switches may wait indefinitely for a response from each other to make a decision in the BP. Consequently, these switches can not take any decision, stopping the control path building. In order to solve this problem, conditions (6.1) and (6.2) have been defined. These conditions are evaluated independently by each switch. By evaluating (6.1) each switch i and j discovers if the latency to the known controller can be improved through its neighbour.

$$d(i, C_m) \leq d(j, C_l) + d(i, j), \tag{6.1}$$

where $C_m, C_l \in C$ and $d(i, j)$ is the latency in link (i, j). Thus, if condition (6.1) is true for switch i, it will decide that the path to the controller C_m is better than the path to the controller C_l through the neighbour j. In this case, the *AN message* sent by the neighbour j is unanswered and becomes an implicit RES_LEAVE message. This means that, switch i does not need to send the RES_LEAVE message. If condition (6.1) is false for switch i, it will decide that the path to controller C_l through switch j is better than the path to controller C_m. In this case, it will not wait for a response from j to make a decision. By evaluating condition (6.2) switch i knows the decision made by switch j without any dialogue between them.

If condition (6.2) is true for switch i, it will not wait for a response because it knows that switch j will join another switch. On the other hand, if condition (6.2) is false for switch i, it will wait for a response from switch j.

$$d(j, C_l) \leq d(i, C_m) + d(i, j). \tag{6.2}$$

These conditions are valid if both controllers are the same or even if they are different. By evaluating these conditions, each switch discovers which is its best neighbour leading to the nearest controller. In this way, the local conflicts are resolved implicitly without generating any additional traffic and in a minimum time. These conditions also ensure that no cycles exist in the control plane.

Failures

Network failures can prevent the switches receiving the response messages and, as a result, being unable to make a decision. In order to avoid this situation, a timer is activated after the first response message is received by a switch. Therefore, if after a time t the switch has not received a response from an adjacent neighbour, it assumes that the latency to a controller through this neighbour is infinite. After that the switch can decide to which parent switch to join.

Algorithm 9 describes the process executed by each switch during the backward phase.

6.3.3 Partial network topology discovered by each controller

As a result of the SDN-RDP protocol execution, each controller receives a *RES_JOIN* message through each of its interfaces. This message contains the sequence of switch identifiers from the leaf switch to the adjacent switch of the controller through which the information is received. This information has a general tree structure, which is converted to a binary string and forwarded in a linear sequence. In this way, each controller can create the control plane G_{C_k}, that consists of all switches into the *RES_JOIN*. After receiving each *RES_JOIN* message, Hello messages are exchanged between the switches contained in the message and controller upon connection startup (as defined in OpenFlow).

Fig. 6.1(a) and Fig. 6.1(b) illustrate the process executed by the SDN-RDP protocol from controllers placed in switches 1 and 11. Numbers over each link represent the delay on the link in microseconds.

Algorithm 9 Backward Phase for switch i.

```
Require: Switch knows its state; $d(k, C_k) \leftarrow$ delay from each $k \in N_i$ to a controller $C_k \in C$
  if state is discovered switch then
    if it has sent and received through the same interface an AN message then
      evaluate $d(i, C_m) \leq d(j, C_l) + d(i, j)$
      evaluate $d(j, C_l) \leq d(i, C_m) + d(i, j)$
      decide the (implicit or not) response
    end if
  end if
  if state is leaf switch or discovered switch and it has received a response message or an implicit response message then
    forward a $RES\_JOIN$ message to the nearest adjacent switch to a controller and a $RES\_LEAVE$ to the rest of the switches (if it is not implicit)
  end if
```

After a controller finishes the SDN-RDP protocol execution, it configures a switch-to-controller path or control path for each discovered switch. As a result, the shortest tree rooted at each controller is obtained (e.g., by using OpenFlow), the switches and links in each control plane define the controller domain $D(C)$ as shown in Fig 6.1(c). During the tree configuration process (backward phase), the switches discover the controller they have been joined to, announcing this information to their $N(i)$ adjacent switches.

Through the defined control path, each switch forwards the following information to its controller:

- its identifier,
- the identifier of its neighbour switches (N_i) and,
- the identifier of the controller to which its neighbours have been joined.

By aggregating this information the controllers can discover a partial network topology and the control paths among them, as explained in the next section.

6.3.4 Protocol complexity

The complexity of SDN-RDP is defined in terms of time and number of messages to create the control plane G_{C}. Given a network of N switches and k controllers, the network discovery time is defined by the mean number of switches assigned to each controller, $N_c = \lceil \frac{N-k}{k} \rceil$, the mean number of

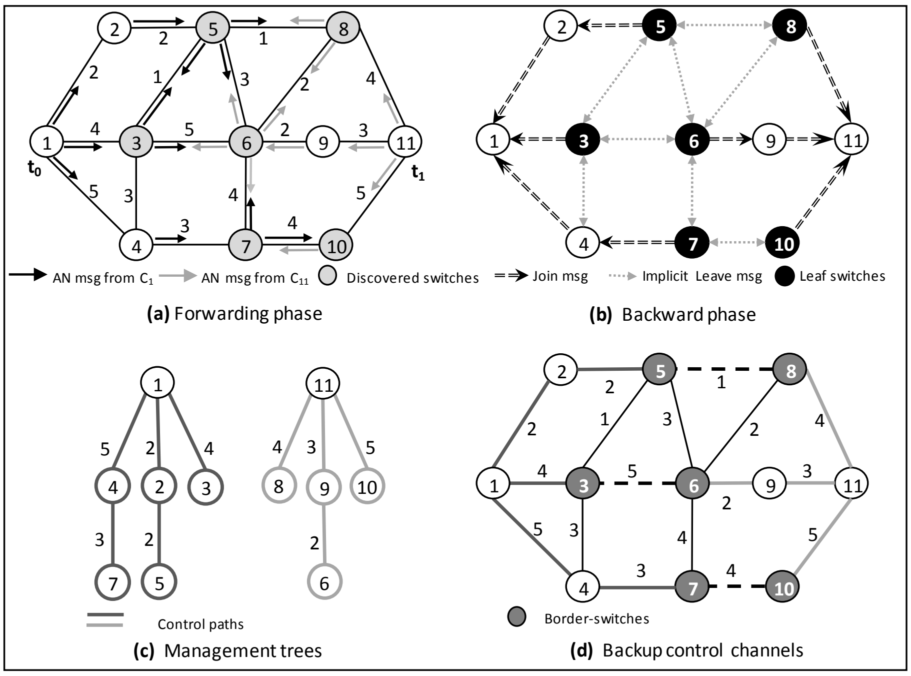


Figure 6.1: Representation of the network topology discovery process.

neighbours per switch, D, and the average time (t_t) defined as the sum of propagation delay (t_p) and transmission time (t_{tx}), assuming that the switch processing time is negligible. Thus, the maximum time to create G_{C} is defined by $2 \times N_c/D \times t_t$. Where 2 represents the time it takes to a message to be forwarded and responded (switch-to-controller) during the RDP protocol. Therefore, the time complexity is $\mathcal{O}\left(\frac{N}{kD}\right)$.

On the other hand, the upper number of messages per controller is bounded by $k \times D + (N-k)\left[2 \times (D-1) + 1\right]$, considering that just one *AN message* is sent by each controller through each one of its output interfaces, the switches forward an *AN message* just one time, and the switches respond to all of the received *AN message*s. Therefore, the message complexity is $\mathcal{O}(ND)$.

6.4 Network topology discovery by controllers

To create a whole map of the network topology on the controllers, each controller has to exchange the network information with all other controllers. As controllers do not have the complete network topology nor the identifiers of all controllers in the network, they are unable to find a path to communicate to each other. However, each controller can find at least one path to a neighboring controller in the network. These are the controllers that discovered the same switch(es) during the SDN-RDP protocol. These switches are called border switches and are defined as follows.

Definition 6.3 (Border switch (Bs)): this is a switch $s \in S$ discovered/managed by a controller $C_i \in C$, $s \in D(C_i)$ that has at least one adjacent switch $x \in N(s)$ managed by a different controller, $x \notin D(C_i)$.

Each controller $C_i \in C$ uses the information forwarded by its discovered switches to identify:

- which controllers they can communicate with, and simultaneously discover...
- through which switches it (controller C_i) can contact these discovered neighbouring controllers.

Controllers establish a temporary path through their border-switches, since these are the switches that have information about how to contact the destination controller, enabling an end-to-end path between controllers to be established. These switches are represented in grey in Fig. 6.1(d). The process to define the control path between two controllers is defined below.

Definition 6.4 (Control path between controllers): a path between two controllers consists of the path between each border-switch to its respective controller and the link between them (these are the dashed links between the border-switches in Fig. 6.1(d)). This path is used to exchange the network information discovered by each one of the controllers.

6.4.1 Finding a path between controllers

To establish a path between two controllers, these have that i) select a border-switch and ii) configure the forwarding table of the border-switch selected.

Each controller has to select a border switch to communicate to each one of its neighbour controllers, given that, each pair of neighbour controllers can have several border-switches to contact each other. In this case, the border-switch selected by each controller is the one that reduces a defined parameter in the end-to-end path between them, as for example, the delay. Although, other parameters can be considered in the selection of border-switches, such as: distance or hops.

After each controller has selected a border-switch to contact a specific neighbour controller, it has to configure the forwarding table of the border-switch, such that when these border switches receive a packet destined to a neighbour controller, they forward the packet to the adjacent switch that is managed by the destination controller (e.g., by using OpenFlow protocol). Process 1 in Fig.6.2 represents this process, where controller C_A configures the forwarding table of its border-switch ($B_{S \to C_A}$), such that it can forward the messages destined to controller C_B through the border-switch ($B_{S \to C_B}$) (Process 2 in Fig.6.2). An end-to-end path between two neighbour controllers can only be established if both controllers executed the aforementioned process. Otherwise, the control messages are dropped. This case is illustrated in Process 2 in Fig. 6.2.

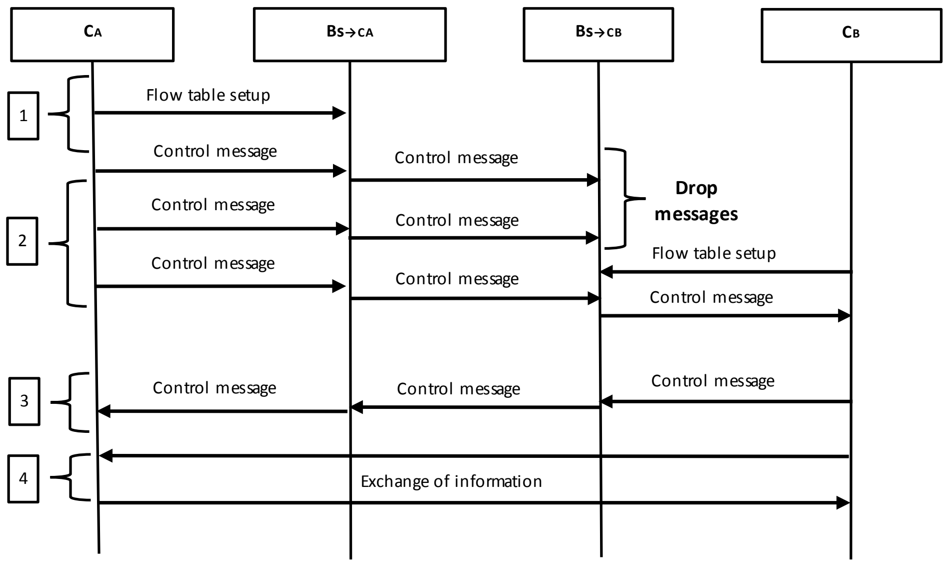


Figure 6.2: Process to discover a path between two controllers, C_A and C_B.

Controllers can forward control messages (e.g. Echo message in OpenFlow) to verify the liveness of a controller-controller connection, which must be replied if the path has been rightly configured. If after a defined time, a controller does not have an *echo message* back from a destination controller, it proceeds to configure the forwarding table of the next border-switch that minimizes the criterion of selection defined. This is the case, if there are more border-switches between the controllers. In the worst case, the forwarding tables of all the border-switches between two controllers will be configured and the controllers will forward *echo messages* until one of them is responded to (Process 3 in Fig. 6.2).

There is an assumption that each pair of controllers have exactly the same information about border-switches to communicate each other. This is realistic as this information is exchanged after each controller finishes the SDN-RDP protocol execution and, this is forwarded from each switch to the controller through a secure path.

By using this path the neighbour controllers exchange the following information:

- switch connectivity (e.g, identifier of switches, switch ports, link identifier, identifier of neighbour switches of each switch discovered), and
- the identifier of the neighbour controllers each controller can communicate with (Process 4 in Fig. 6.2).

Through this exchange of information, controllers not only discover the partial network information found by their neighbour controller(s), but also discover the controllers to which it can communicate with. Therefore, neighbour controllers can request any outstanding information about the network topology discovered by other controllers from their neighbours, and even announce discovered controller identifiers between them.

The procedure is summarized in the flowchart illustrated in Fig. 6.3. This flowchart defines the processes executed by the controllers:

- **Forward information requested**:

  A controller C can receive a request from a neighbour controller $C_i \in N(C)$ asking for information of a controller $C_j \in N(C)$. If controller C does not have this information, it queues the request for a limited time (defined by a timer). If controller C receives information about C_j before the timer is expired, it forwards the information of C_j to C_i. This process is defined in light grey in the flowchart.

- **Request information from their neighbour controllers**:

  A controller can receive information about a new controller identifier. In this case, the controller asks for this information to the neighbour controller that announces this information (the new controller identifier). This process is defined in white in the flowchart.

- **Announce to them newly discovered topology information**: Controllers can discover the information that their neighbour controllers have about other controllers from the information exchanged between them. If a controller has information about identifiers that their neighbour controllers have not discovered, it announces this information (controller identifiers) to them. This process is defined in dark grey in the flowchart.

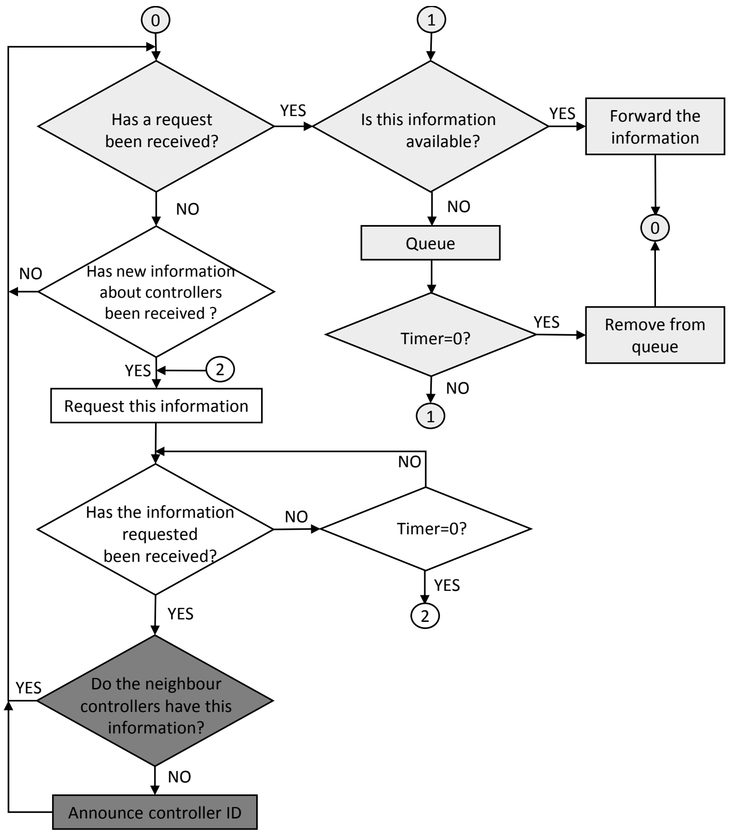

0
1
Has a request been received?
YES
Is this information available?
YES
Forward the information
NO
NO
Queue
0
NO
Has new information about controllers been received ?
Timer=0?
YES
Remove from queue
YES
2
NO
Request this information
1
NO
Has the information requested been received?
NO
Timer=0?
YES
YES
2
YES
Do the neighbour controllers have this information?
NO
Announce controller ID

Figure 6.3: Process to discover the network topology by controllers.

If all the controllers execute this process with the controllers they have information from, at some point, all controllers discover each other, and therefore, will be able to compose the whole network topology.

6.5 Allocation of switches to controllers

For consistency, the criterion used to distribute switches to controllers should be the same as that considered to select the controller placements in the SDN network (e.g., shortest paths, delay, robustness, etc).

The first switch-to-controller distribution is obtained as a result of the $SDN-RDP$ protocol execution. In this case, the resulting allocation of switches-to-controllers not only depends on the network topology, but also on the time each controller starts the protocol execution. That is to say, in an scenario where all controllers start the SDN-RDP protocol execution in a synchronized manner, each controller finds its nearest switches, such that if the criterion to distribute the switches is to minimize the switch-to-controller delay, it is not necessary to re-distribute switches between them.

However, this situation is not realistic, since it is unlikely that all controllers start the protocol execution at exactly the same time. When controllers start the SDN-RDP protocol execution independently, without synchronization, controllers will discover their nearest switches but can also discover switches that are nearer to other controllers. Consequently, the initial allocation of switches to controllers must not satisfy a defined criterion.

6.5.1 Re-distribution of switches-to-controllers

To re-distribute the switch management between the controllers satisfying a specific criterion, it is considered that each controller locally executes a switch-distribution algorithm through which they discover the set of switches it should manage as well as the switches that the other controllers should manage. Fig 6.4 represents this process.

The controllers must execute the switch-distribution algorithm after discovering the whole network topology that is assumed to be equal over all the controllers. From this information each controller must classify the following information, as illustrated in Process 1 in Fig 6.4.

- the identifier of switches and controllers,
- adjacent connections between switches, and
- delays in all of the links in the network.

Criterion to re-distribute switches-to-controllers

The criterion to distribute the switches between controllers must be defined by network providers, as the allocation of switches to controllers should be defined based on the service requirements.

A switch-to-controller assignation algorithm must consider:

- the criterion to distribute the switches-to-controllers, and
- a function $f(\delta)$ that assigns a weight to each switch $s \in S$ with respect to each controller $c_i \in C$ in the network.

Each controller assigns a weight to each switch through a function $f(\delta)$ that weighs up the relation of each switch to each controller with respect to a defined criterion. A number of criterion can be considered, such as: the delay, hops, distance, for instance. As a result, each switch must have the same number of weights as there are controllers in the network. In this way, each controller, based on the weights obtained for each switch, identifies those switches that each controller should manage (process 2 in Fig 6.4). Ideally, it is expected that all controllers obtain the same switch-to-controller assignation as a result.

To make sure the switch distribution found by each controller is the same, when controllers have each switch assigned to a controller, they announce the assignation found through the West-East bound interface (process 3 in Fig 6.4). With this information, each controller checks if the results obtained by the other controllers match its own results (process 4 in Fig 6.4).

If the result found by each controller does not match, the controllers which encounter a mismatch between them, start a re-allocation process to solve the conflict(s) (process 5 in Fig 6.4). For simplicity, conflicts can be solved by the controllers as follows:

- If switches in conflict do not have downstream switches, the switches are assigned to the controller with the lowest identifier,
- otherwise, switches in conflict are assigned to the controller that most reduces the number of downstream switches to be re-allocated.

If there are no mismatches, controllers locally update the switch assignation and proceed to configure their identifier in the set of switches they will manage through the Southbound interface (process 6 in Fig 6.4). After that, each controller can establish the control path with each one of their switches using a communication protocol, OpenFlow, for instance. This process is summarized in Fig. 6.5.

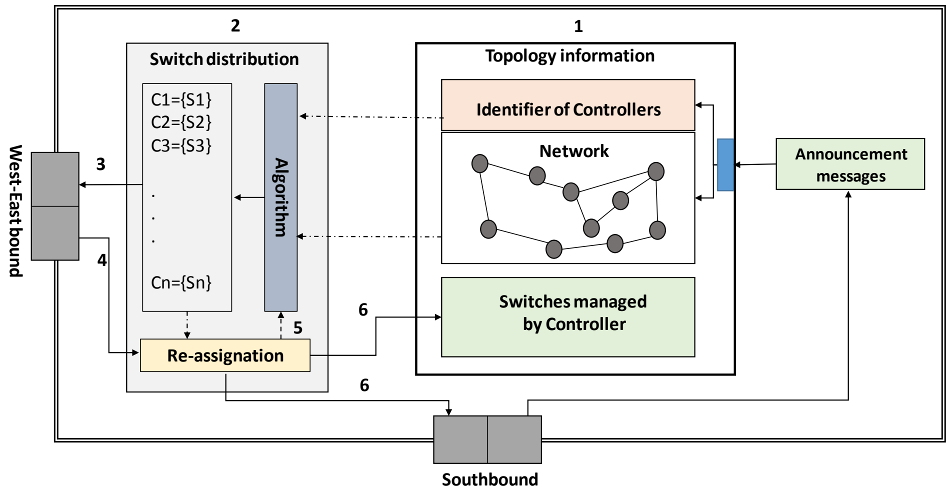


Figure 6.4: Representation of the processes to allocate switches to controllers.

6.6 Updating the network topology

In an operative network, topological changes can happen anywhere and anytime, which must be detected and announced to the respective controllers to maintain the network topology information updated on the control plane. SDN-RDP protocol can keep the network topology updated on the controllers by announcing events through *AN messages*, but this can not detect network changes.

Regarding the importance of failure detection in SDN networks, there is not a detailed mechanism of failure detection on any of the communication protocols defined for SDN networks (e.g., OpenFlow specifications), this is still an open issue.

This section introduces a cooperative mechanism to maintain the network topology updated. It is said to be cooperative because:

- combines both switch and controller fault detection approaches,
- switches announce network changes (or events) to the controllers that are capable of discovering what the events are about and update the network topology, and in addition,
- controllers can announce to a controller when discovering events that involve the switches that it manages,

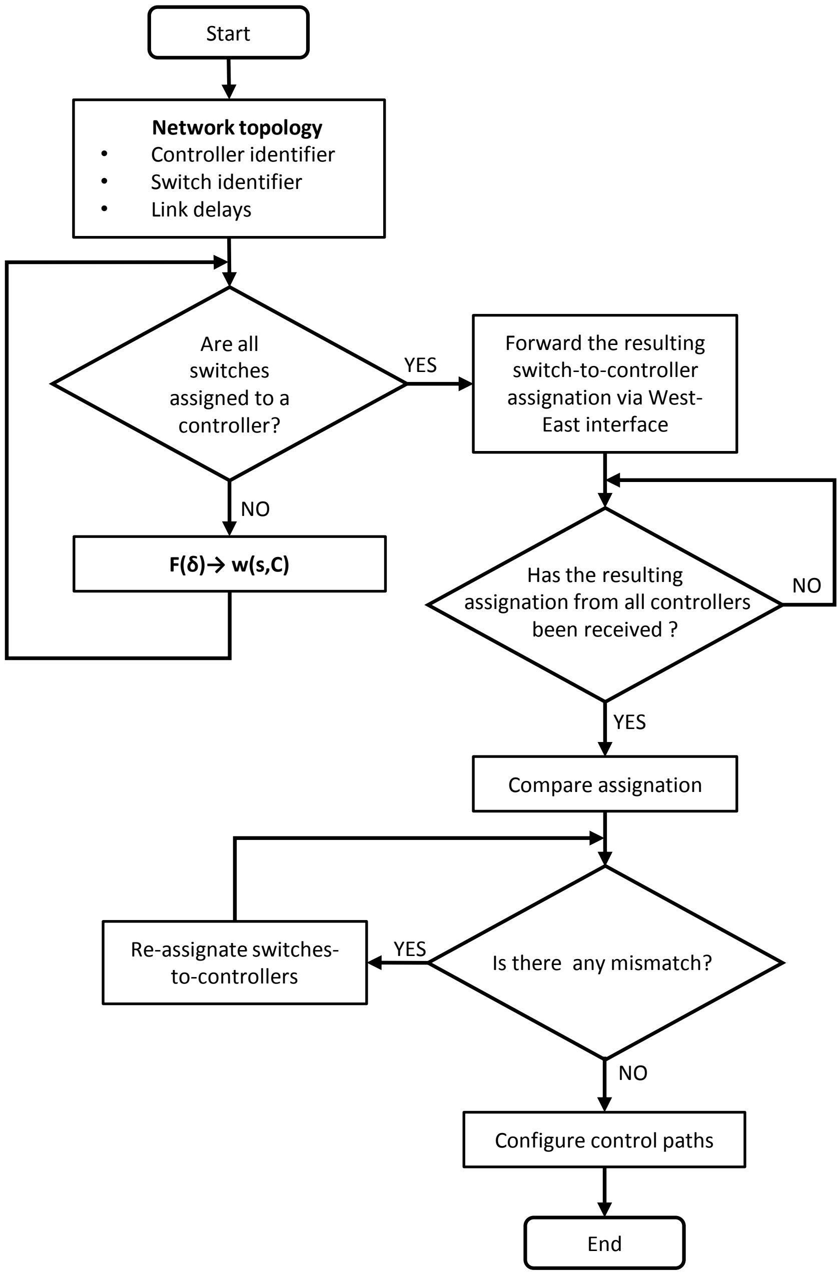

Start
Network topology
Controller identifier
Switch identifier
Link delays
Are all switches assigned to a controller?
YES
NO
F(δ)→ w(s,C)
Forward the resulting switch-to-controller assignation via West-East interface
Has the resulting assignation from all controllers been received ?
NO
YES
Compare assignation
Is there any mismatch?
YES
Re-assignate switches-to-controllers
NO
Configure control paths
End

Figure 6.5: Flowchart: assignation process of switches-to-controllers.

- switches and controllers announce the network events through the SDN-RDP protocol.

6.6.1 Detecting network topology changes

To maintain the network topology updated in a control plane, network changes have to be detected and announced. Two types of network changes are detected, these are: network failures and network devices recovered after a failure.

Network failures: failures can be detected through the BFD protocol. It is considered that the link status is monitored by a BFD session between all adjacent switches. That is, each switch $s \in S$ forwards BFD Echo packets to all its neighbours to one hop $x \in N(s)$, such that all the links in a network are monitored.

Network recovery: despite BFD was only designed to detect failures, it can also be used to detect when a communication between two switches has been recovered after a failure. For this purpose consider that:

- each switch $s \in S$ executes the BFD protocol with each one of its neighbour switches (1 hop),
- each switch $s \in S$ has information about all its connected adjacent switches $N(s)$, and
- each switch has a local list that contains the identifier of the adjacent switches that have failed and have not been recovered, denoted as L_{down}.

When a switch s detects that the communication with an adjacent switch x is down $x \in N(s)$, it adds the identifier of switch x to the L_{down} list. If a switch detects that an adjacent switch included in this list starts the BFD protocol, they remove the switch from L_{down} as the communication with it is no longer down. When a switch detects any of these both events, network failure or a network recovery, it announces the event to the controller through *AN messages.*

Note that the BFD protocol detects network failures, but it does not define if the failure is due to a failed switch or a broken link. To maintain the network topology updated in the control plane, this is required that the controllers not only detect the type of network change, but also if it was due to a link or switch. Below the process to discover if a failure is due to a switch or link failure is explained in detail.

6.6.2 Principles to update the network topology

Each switch $s \in S$ has a set of interfaces defined as $I(s)$. A bidirectional link between two switches $A, B \in S$ is defined through the switch interfaces $a \in I(A)$ and $b \in I(B)$, such that the link is denoted as $a \leftrightarrow b$. It is considered that a switch s may have as many neighbour switches $N(s)$ as interfaces $I(s)$.

In general, each pair of switches A and B monitors each other through ports $a \rightarrow b$ and $b \rightarrow a$, respectively, by exchanging BFD Echo messages as described in [38]. Switch A is said to be the monitor switch of B, and B is said to be the target switch of A. If switch A has N neighbour switches, where $(N = |N(A)|)$, switch A is the target switch of N monitor switches and switch A is the monitor of N target switches.

That is, each switch $x \in S$ acts as:

- *monitor*, when it forwards control messages to its $N(x)$ neighbour switches,
- *target*, when it is monitored by its $N(x)$ neighbour switches.

By monitoring their target switches, each monitor switch $A \in S$ detects when two of the following events occur:

- A **failure** is detected by a monitor switch A when it does not receive a response back from a target switch $B \in N(A)$ to which it has forwarded an BFD Echo message.
- A switch A can detect that a communication has been **recovered**, when it receives an BFD Echo message from a neighbour $B \in N(A)$ that is contained in its L_{down} list.

Events are announced by the monitor switches through *AN messages* to the controller via their control path. This message contains the following information:

- the identifier of the monitor switch, A,
- the identifier of the target switch, B,
- the identifier of the port of monitor switch A through which the target switch B is monitored $(a \rightarrow b)$, defined as a,
- the identifier of the port of monitor switch B through which the target switch A is monitored $(b \rightarrow a)$, defined as b.

Note that through an *AN message*, a monitor switch A announces the identifier of the target switch B with which it lost communication and through which interface it was connected to. Through the information contained in the *AN messages*, the controllers can discover:

1) the location of the event and what the event is about (e.g., failure or recovery event) and,

2) what the network failure is (e.g., link or switch failure), updating the network information on the control plane.

Location of an event in the network

The location of an event in the network is limited to the communication between the monitor switch and the target switch announced in an *AN message*. That is, the event may be related to a failure or recovery of the monitor switch, the target switch or the link between them.

To discover if the event is related to a failure or a recovery communication, it is considered that controllers have a list that contains the identifier of the link interfaces and identifier of the target switch broken, called ID_{list}. Upon receiving an *AN message* and discovering what the failure is, the controllers add in the ID_{list} list, the identifier of: i) the target switch if it has failed or ii) the identifier of the interfaces of the link failed.

Upon receiving an *AN message* a controller discovers what the event is about as follows:

- if the identifier of i) the target switch or ii) the link between monitor-target switches, announced in an *AN message* are not included in ID_{list}, a Controller can discover that there is a **network failure**.
- if the identifier of i) the target switch or ii) the link between monitor-target switches are included in ID_{list}, the event is said to be a **network recovery** event.

Discovering a network failure

By correlating the information contained in the *AN messages* received, the controllers can discover the kind of failure as follows:

Link failure. Consider switches A and B managed by the same controller, this is $A, B \in D(C)$, that have a bidirectional link between them. If the link between these switches fails, it is expected that controller C receives two *AN messages*, one coming from monitor switch A and the other from monitor switch B. If the identifier of the ports in the messages correspond to a bidirectional link between the switches, that is if:

$$\exists\ a \in I(A), b \in I(B) : (a \rightarrow b), (b \rightarrow a) \Rightarrow a \leftrightarrow b,$$

then a controller can deduce that the bidirectional link $a \leftrightarrow b$ has failed.

Switch failure. A switch failure is defined by the status of the links to its N monitor switches. If switch A fails, it is expected that the controller receives $|N(A)|$ *AN messages* coming from the $N(A)$ monitor switches of target switch A, such that:

- the identifier of the target switch in all the *AN messages* is the same, and
- the set of the port identifiers of the monitor switches defined as Pm are included in $I(A)$, $Pm \subseteq I(A)$.

A controller can deduce that switch A is no longer operational if:

$$Pm \setminus I(A) = \emptyset\ .$$

After detecting a failure, the controllers update the network topology information and the ID_{list} list.

The aforementioned conditions defined to discover a link and switch failures can only be satisfied if:

1) all the monitor switches of a target switch have an active control path to their controller to announce the event and,

2) all the *AN messages* are processed by the same controller.

Condition (1) ensures that events are announced to the controllers and condition (2) guarantees that a controller has the necessary information to discover the type of failure. However, these conditions can not always be satisfied, since not all the switches have an available control path to the same controller to announce the events detected.

In the case with condition (1), there are different reasons why switches can not communicate to their controller, and therefore this condition can not be satisfied. This is the case with switches that are isolated after a network failure or switches that do not have an active backup control path to their controller. Fig. 6.6 represents an example for each one of these cases.

- *Isolate switches*: some switches can be isolated from their controller as a consequence of their scarce network connectivity and/or network failures. These are switches $s \in S$ such that $\nexists\ x \in N(s) : p(s, C(s))$. This is the case with switch 5 in Fig. 6.6(a).

- *Disconnected border switches from their controller*: These are switches $s \in BS$ for which do not exist a control path to their controller, $\nexists\ x \in N(BS) : x \in D(C(BS))$. This is the case with border switch 7 in Fig. 6.6(b) that does not have connection to a neighbour switch managed by the same controller.

- *Switches that do not have a backup control path configured*: these are switches that are unable to communicate with their controller even though at least a physical switch-to-controller path exists. This is because, switches do not have this path configured. These are switches $s \in S$ such that $\nexists\ x \in N(s) : p(s, C(s))$. This is the case of switch 6 in Fig. 6.6(c), that can be monitored by the controller through path (1-3-5-7-6).

On the another hand, condition (2) can not be satisfied for the switches that have neighbour switches that belong to a different controller domain that their target switch. This is the case when there are more than one controller in the network, given that the management switch is distributed between them, and as a consequence the border switches have neighbour switches managed by at least a different controller.

For the aforementioned reasons, additional conditions have to be defined in order to satisfy the conditions (1) and (2) and therefore the controllers can discover failures in the network.

In order to minimize the number of switches disconnected from the control plane after a network failure, it is considered that the controllers have configured a disjoint backup path on each switch, if it is possible. Note that despite the number of switches that can communicate with their controller may increment after a network failure, it is not a sufficient condition to guarantee that all switches can announce the network events to its controller (condition 1). Therefore, to monitor the switches that do not have a backup control path to their controller, the following additional conditions are required:

- controllers can monitor switches that do not have a control path, if it is possible, by configuring a BFD session,

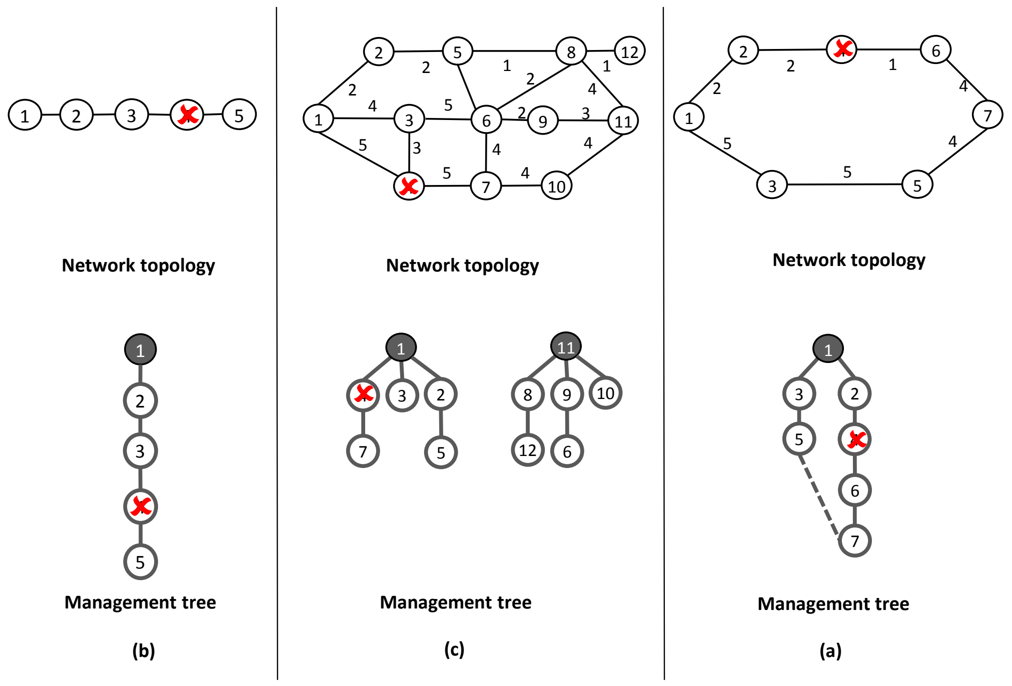


Figure 6.6: Examples of switches disconnected from the control plane, with their control plane representation: (a) isolated switches, (b) switches disconnected from their controller, (c) switches that do not have a backup control path configured.

- otherwise, controllers can deduce what the failure is about from the *AN messages* received.

On the other hand, in order to satisfy condition (2), it is considered that controllers receiving *AN messages* that contain information about a target switch that they do not manage are forwarded to the controller responsible for managing the target switch.

Considering the aforementioned considerations, the mechanism to discover the kind of network changes in a network is generalized below.

6.6.3 Discovering network events and updating the network topology information

The procedure to detect network changes and keep the network topology and the ID_{list} updated can be divided in three different parts: 1) network changes detection, 2) link failure detection and 3) switch failure detection. Fig. 6.7. displays this procedure.

Network changes detection

Each controller C only discovers network changes over the network elements in its domain $D(C)$. When a controller is informed about a network change, through an *AN message*, it checks first if the event announced has occurred in its domain. That is, if the target switch identifier A announced in the *AN message* is a switch in its domain, $A \in D(C)$.

If the target switch is managed by a different controller, the controller receiving this message re-forwards the *AN message(s)* to the controller responsible for managing the target switch. The switches in this situation are the switches identified as border-switches (BS).

Controller receiving an *AN message* related to a network element (switch, interface) in its domain, first checks if the identifier of the target switch or its interface announced in the message are included in the ID_{list}. If this information is included in the list, the controller deduces that the communication between the monitor and target switches announced in the *AN message* has been recovered. In this case, controllers update both the ID_{list} and the network topology information. In addition, controllers re-configure the control path of the recovered target switch (this specific process is played in Fig. 6.8). Otherwise, the *AN messages* were forwarded to announce a network failure.

In case a network failure, to discover what kind of failure happened in the network, the controllers wait within a defined window time, for an *AN message* from the target switch or any of its monitor switches, different to the monitor switch that had announced the event, if any exist.

If a controller does not receive any messages from those switches, it forwards probe packets to monitor the target switch through all its active interfaces, if possible.

Link failure detection

The link failure detection process is illustrated in Fig. 6.9. A controller discovers if the network failure that occurred was a *link failure* if:

- after receiving an *AN message*, a controller also receives a message from the target switch defined in the message, or
- after forwarding probe messages to the target switch a response is received.

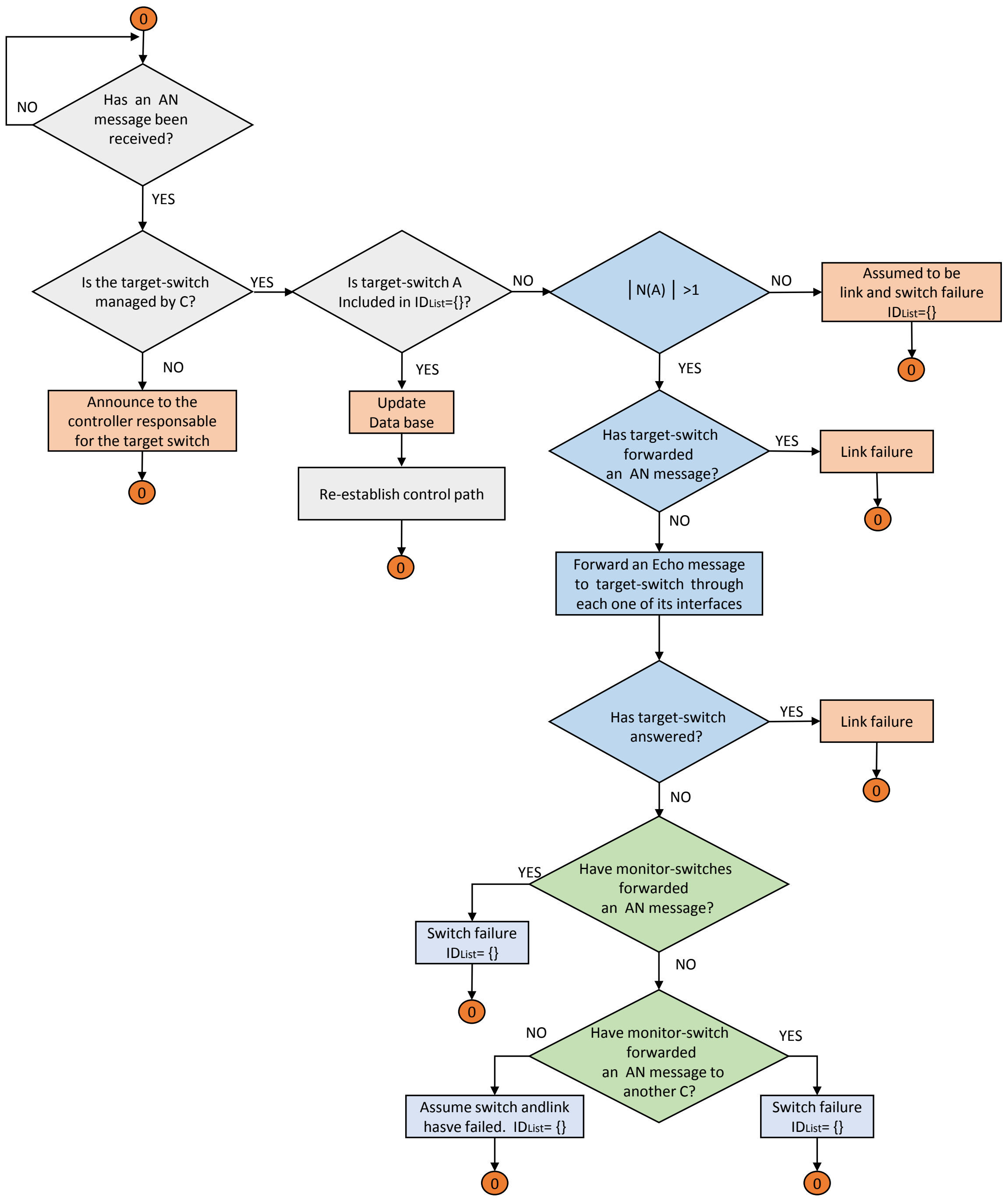

0
NO
Has an AN message been received?
YES
Is the target-switch managed by C?
YES
Is target-switch A Included in IDList={}?
NO
| N(A) | >1
NO
Assumed to be link and switch failure IDList={}
0
NO
Announce to the controller responsable for the target switch
0
YES
Update Data base
Re-establish control path
0
YES
Has target-switch forwarded an AN message?
YES
Link failure
0
NO
Forward an Echo message to target-switch through each one of its interfaces
Has target-switch answered?
YES
Link failure
0
NO
Have monitor-switches forwarded an AN message?
YES
Switch failure IDList= {}
0
NO
Have monitor-switch forwarded an AN message to another C?
NO
Assume switch andlink hasve failed. IDList= {}
0
YES
Switch failure IDList= {}
0

Figure 6.7: Process to maintain updated the network topology.

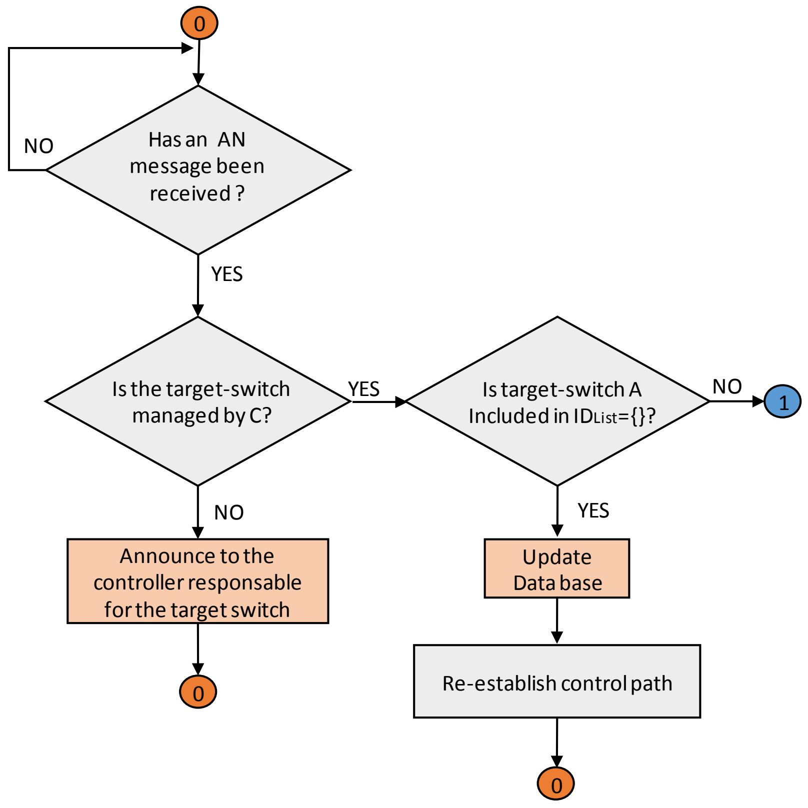

0
Has an AN message been received ?
NO
YES
Is the target-switch managed by C?
YES
Is target-switch A Included in IDList={}?
NO
1
NO
YES
Announce to the controller responsable for the target switch
Update Data base
0
Re-establish control path
0

Figure 6.8: Process to detect network changes.

Switch failure detection

In the case of a *switch failure* (e.g., switch A), a controller can discover it in two different ways:

- If after receiving an *AN message*, a controller receives at least one message from any of the monitor switches $x \in N(A)$ of target switch A, different to original monitor switch that had announced the event.
- If a controller C receives from another controller an *AN message* from a monitor switch of the target switch A monitored, where $A \in D(C)$. This is the case with target switches that are border switches.

This process is illustrated in Fig. 6.10.

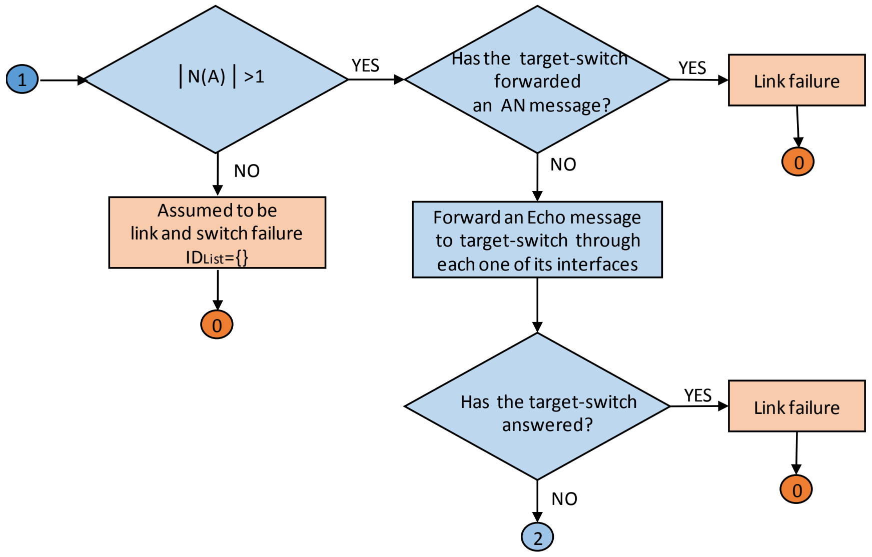

1
| N(A) | >1
YES
NO
Assumed to be link and switch failure IDList={}
0
Has the target-switch forwarded an AN message?
YES
NO
Link failure
0
Forward an Echo message to target-switch through each one of its interfaces
Has the target-switch answered?
YES
NO
Link failure
0
2

Figure 6.9: Process to detect link failures.

There are two cases where a controller can not discover if the failure is due to a link or switch failure, given that neither condition is satisfied. This is the case for:

- switches that have only one neighbour switch, this is $N = 1$. In this situation, a controller assumes that both the link and target switch have failed.
- when a controller does not receive an *AN message* message from neither, the target switch nor its monitor switches. In this case, it is also considered that both the target switch and the link between the target and monitor switch announced in the *AN message* have failed.

Each time a controller detects a switch or a link has failed, it updates its ID_{list} and the network topology on the control plane.

6.7 Simulation and Results

The SDN-RDP protocol has been implemented from scratch in OMNET++. In order to show the efficiency and scalability of this protocol three different evaluations are presented. First, the protocol performance is evaluated when it is executed in a synchronous (Syn) mode. Second, a comparison of the

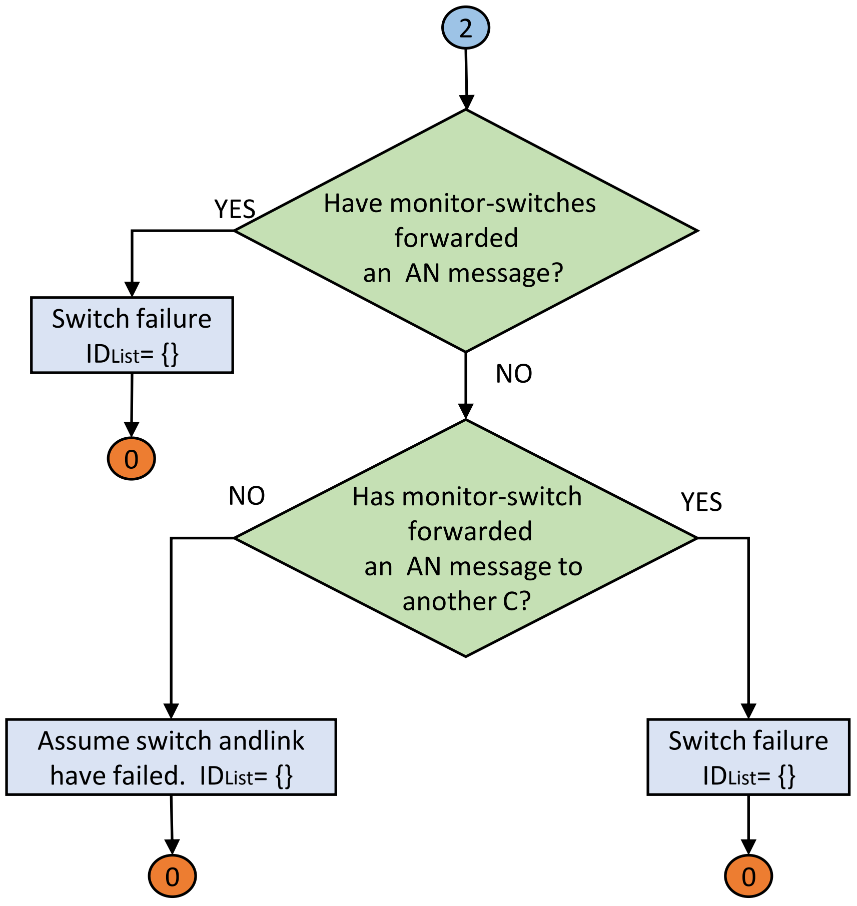

2
Have monitor-switches forwarded an AN message?
YES
NO
Switch failure
IDList= {}
0
Has monitor-switch forwarded an AN message to another C?
NO
YES
Assume switch andlink have failed. IDList= {}
Switch failure
IDList= {}
0
0

Figure 6.10: Process to detect switch failures.

protocol performance when it is executed in synchronous and asynchronous (Asyn) modes is presented. For these both cases, 100 networks that consists of 200 nodes are used. Finally, the scalability of the protocol is evaluated in a synchronous mode using a set of graphs generated randomly that have different sizes, 50, 100, 200 and 500 switches. Each set of networks consists of 100 graphs. For all aforementioned cases, the results for the SDN-RDP are presented together with their respective 95% confidence intervals based on Student-t distribution.

The adjacency matrices for these graphs are generated by using the Gephi software. This tool generates networks with a given number of switches n, which are randomly connected by undirected links. For a random pair of switches, there is a probability p (wiring probability), where $0 < p < 1$, that a link exists which connects them. This implies that the degree of random graphs generated using a fixed value of p increases when n increments. Table 6.1 shows the main characteristics of the generated networks, all of them were generated using a wiring probability of 0.05.

Table 6.1: Information of randomly generated networks.

Size	Avg. Number of links	Avg. Switch degree
50	65	3
100	253	5
200	994	10
500	6243	25

In all these cases, a link capacity of between 100 Mbps to 10 Gbps is assumed and the distances between any pair of switches are selected randomly in a range of 1 to 15 Km. The protocol is executed over k controllers in both synchronous (Syn) and asynchronous (Asyn) modes where k varies between 1 and 8 controllers. The controllers are selected among the 200 nodes in each one of the networks, using the $k - Critical$ algorithm.

6.7.1 Evaluation

The evaluation of the protocol consists of two different aspects: i) the performance of the protocol and ii) the characteristics of the control plane obtained as a result of executing the protocol over the controllers.

The performance of the protocol is evaluated in terms of: i) the average time it takes to discover the whole network topology, and ii) the average number of messages processed by each switch during the network topology discovering process when varying the number of controllers.

- **Average number of messages per switch**: this is the number of control messages (Announcement messages (AN) Response messages (RES) and Improved Announcement messages (IAN)) received and sent by each switch during the SDN-RDP protocol execution.
- **Computing time**: this is the time it takes for the controllers to discover a partial network topology that defines their control plane. This time includes: i) the time it takes the switches to forward the AN *messages* over the network, ii) the time it takes the switches to make a decision about which controller to join to, and iii) the forwarding time of the join message from each switch to its controller.

On the other hand, the characteristics of the resulting control plane evaluated are:

- **Switch distribution**: this defines the number of switches that each controller discovers as a result of the SDN-RDP protocol execution.
- **Robustness**: this parameter provides information about the fraction of unprotected switches in the presence of failures in the network. This is the robustness metric proposed in Section 5.5.
- **Stretch**: this is the ratio of the path length of the shortest path found between a controller and the path length of the path found by the controller as a result of the protocol execution.
- **Expected Data Loss (EDL)**: this metric defines the dependence of a switch with respect to the upstream switches in the control plane.
- **Control tree path delay**: this is the delay of each switch $s \in S$ with respect to its controller $C_A \in C$.

6.7.2 Protocol evaluation in Syn mode

First, Fig. 6.11 shows results of the performance of the protocol and control plane characteristics when the protocol is executed in Syn mode over k controllers , $k \in 1, ..., 8$. In Syn mode, the controllers start the protocol execution at the same time, i.e., $t = 0$.

Fig. 6.11(a) illustrates the mean *computation time* it takes to k controllers to discover the network topology. This time tends to decrease as the number of controllers increases. This is because, as the number of controllers increases, the number of nodes assigned to each controller decreases. If the number of controllers increases, the number of hops the *AN messages* have to be forwarded along the network decreases and consequently, the time the controllers receive a response is smaller. However, when considering more than $k = 4$ controllers the mean computation time does not decrease significantly because the reduction of number of nodes per controller is not so significant.

Fig. 6.11(b) shows the mean *number of control messages* that each node in the network processes (send and receive) during the protocol execution. From this figure, it can be seen that, in Syn mode, the average number of messages on each node remains constant with respect to the number of controllers. This is because: i) IAN messages are not required as switches

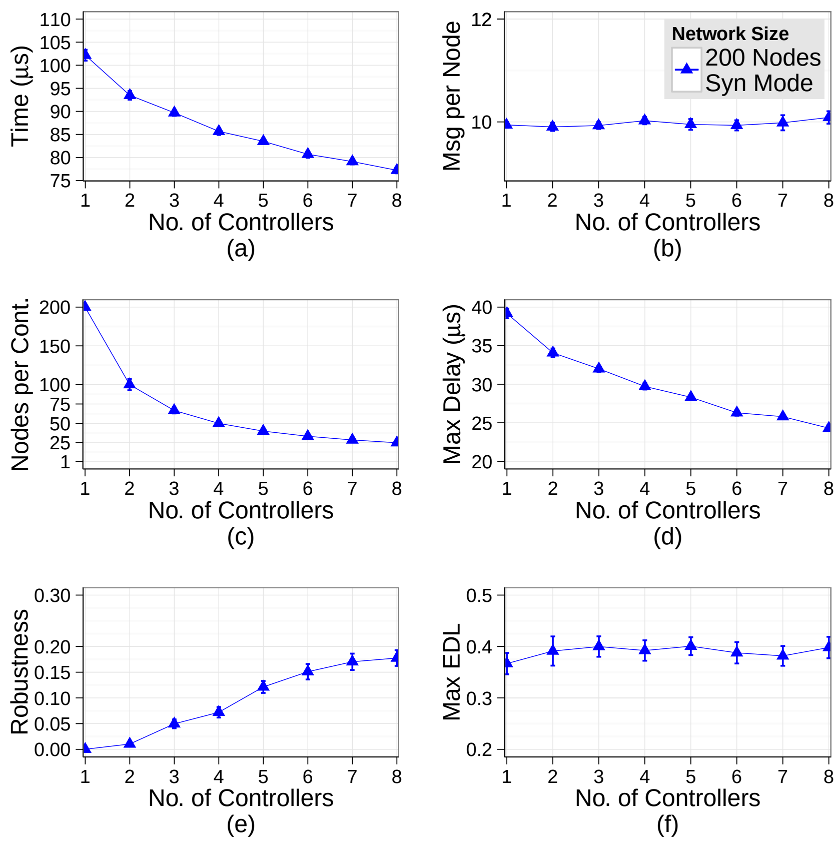


Figure 6.11: Protocol evaluation in Syn mode for networks that consist of 200 switches.

receive the AN message through their shortest path, and ii) conflicts are solved using implicit response messages.

The *allocation of switches between k controller* is illustrated in Fig. 6.11(c). The number of nodes managed by k controllers is reduced when k increases. This is because the nodes tend to be discovered by their closest controller, therefore, the greater the number of controllers, the fewer the number of nodes they discover.

Fig. 6.11(d) shows *the maximum control path delay* found for each switch-to-controller path. Given that RDP-SDN protocol tends to discover the closest switches from each controller, the switch-to-controller delay tends to be reduced as k increases.

The *robustness* of the resulting control plane is represented in Fig. 6.11(e).

As expected, when the number of controllers increases the number of nodes protected against a network failure is reduced. This is because, when the switches are distributed among k controllers, the probability of finding alternative control paths of each switch to its controller is lower compared with the case of having $k-1$ controllers.

Fig. 6.11(f) shows *the maximum expected data loss* per node, which represents the maximum nodes that are disconnected when a failure occurs in the network . From this graph, it can be seen that the mean value for the expected data loss for all number of controllers evaluated is similar. Further analysis of the dispersion of this distribution will be presented in the next figure.

In order to analyze the protocol performance in detail, the results obtained when the protocol is executed in Syn mode are presented through boxplot graphs in Fig. 6.12. This is a standardized way of displaying the distribution of numerical data groups through their quartiles: minimum, first quartile, median, third quartile, and maximum. In the simplest box plot the central rectangle spans the first quartile to the third quartile (the interquartile range or IQR). The middle half of a data set falls within the interquartile range. The larger the IQR, the more variable the data set is. The largest and smallest values shown for each evaluated parameters are called whiskers. Points outside the area defined by the whiskers are called outliers and are represented by a small circle (these are data points more extreme than $3 \times IQR$ above Q3 or below Q1).

The interquartile range is typically reported along with the median to represent variability and central tendency in distributions that either are skewed or have outliers.

The box plot displays the full range of variation (from minimum to maximum values obtained for an evaluated parameter), the likely range of variation (the IQR), and the line dividing the box is the median of the data set obtained. In the box plot each section represents 25 % of the data, which are defined in Table. 6.2.

From Fig. 6.12(a), it can be seen that the dispersion of the *computation time* values are not significantly lower and higher with respect to the median, for the different number of controllers. This suggests that overall controllers have a high level of agreement with each other regarding the time needed to create the control plane tree.

Fig. 6.12(b) illustrates the dispersion of the *number of control messages* processed by the nodes in the network. The median number of messages tends to be constant and is close to the mean value, identifying symmetric distributions. Even though the data dispersion increases as the number of

Table 6.2: Description of quartiles.

Quartile	Description
Q1	25% of the data are less that or equal to this value
Q2	The median. 50 of the switch distribution are less or equal to this value
Q3	75% of the data are less or equal to this value
IQR	The distance between $(Q3 - Q1)$; thus, it spans the middle 50% of the data.

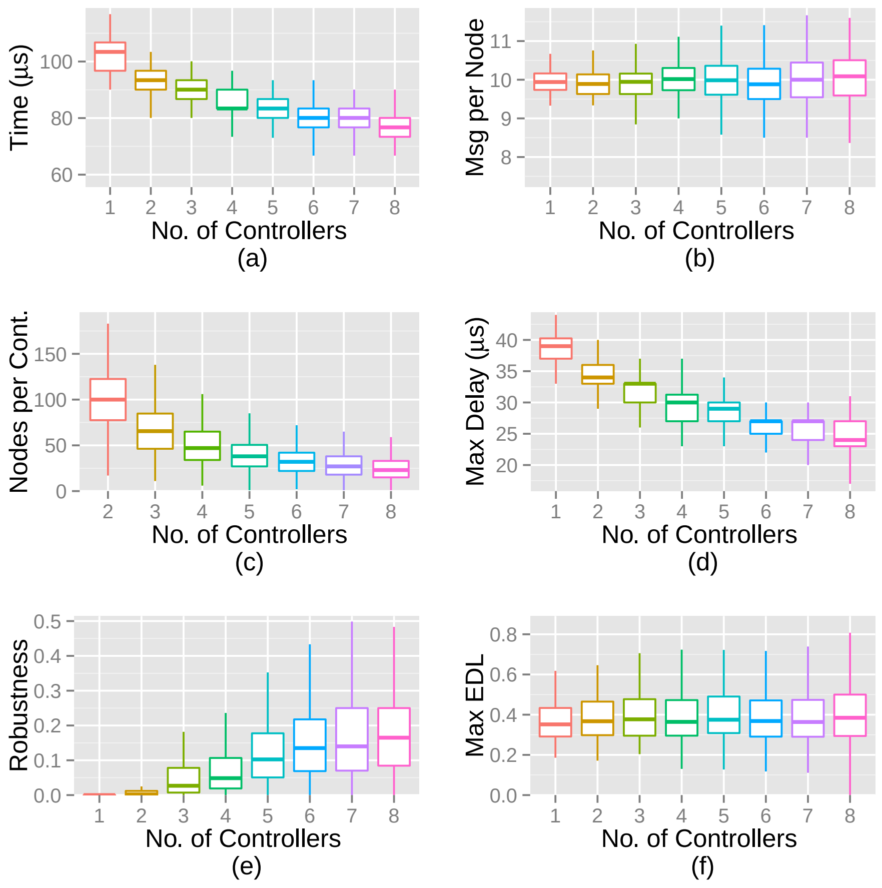


Figure 6.12: Boxplots of the protocol evaluation in Syn mode for networks that consist of 200 switches.

controllers increases, the IQR only varies from 0.42 (1C) to 0.91 (8C), similar to the variation of the computation time parameter.

In addition, Fig. 6.12(c) shows that the dispersion of the *number of nodes per controller* is lower as the number of controllers increases. This figure indicates that this parameter has a high variability, in accordance with the criterion of allocation of switches to controllers, which takes into account the delay, and not the number of nodes.

In the case of the *maximum control path delay* shown in Fig. 6.12(d), the dispersion for k controllers tends to be in the range of 10 $\mu secs$. Moreover, the IQR varies from 2 μs (6C) to 4.2 μs (4C).

In the case of the *robustness* of the Fig. 6.12(e), the dispersion values are wider when the number of controllers increases. This is because the protection of some nodes can be high while the robustness for nodes which their adjacent nodes are managed by other controllers, is low. In this case, the maximum robustness is pretty far away from the box, and the box is located farther to the top. This long upper whisker means that the data is partially skewed. Thus, nodes with low robustness (high values) have varied values of this parameter, while nodes with high robustness (low values) have similar values.

Fig. 6.12(f) illustrates the maximum *expected data loss* found for the networks evaluated. This parameter represents the maximum expected data loss or nodes that are disconnected when a failure occurs in the network. From this figure, it can be seen that the mean value for the expected data loss for all number of controllers evaluated is similar. However, when the number of controllers increases, the data dispersion is higher.

From this result, it can be deduced that when the number of controllers increases ($k \geq 3$), the nodes in the control plane have a high node dependency, compared with the results obtained when k is lower. This is because, depending on the network topology and the condition to select the controllers (e.g., switch-to-controller delay), there are cases where the controllers can be selected to manage a reduced number of switches (critical nodes). As a result, some controllers can manage a high number of switches (which have a high number of downstream nodes), while other controllers can have few switches (switch to one hop of the controller). This deduction is coherent with the other results obtained.

From all these results, it can be seen that the protocol performance is good as the value dispersion for all parameters evaluated varies in a short-range, and hence the protocol behaviour is similar for all nodes.

6.7.3 Protocol evaluation in Asyn mode

In this section a comparison between both protocol operation modes, Syn and Asyn modes, is presented. In Asyn mode, k controllers start the protocol execution randomly that varies from 1 to 5 seconds, but is really initiated by the first *AN message* (of another controller) they receive.

In order to compare the protocol performance different techniques can be used, such as histograms or boxplots. This last case exploits the fact that an equal shape is equivalent to “linearly related quantile functions”. Such a plot is the Quantile-Quantile plot, or Q-Q plot.

A Q-Q plot is a plot of the quantiles of the first data set (Syn mode) against the quantiles of the second data set (Asyn mode). It is a diagnostic tool, which is widely used to assess the distributional similarities and differences between two independent data sets. The Q-Q plot is an effective display of the relationship between corresponding order statistics from two samples: plot the corresponding pairs as points in a scatter plot.

A Q-Q plot is constructed by comparing two distributions, matching like-positioned values (i.e., quantiles) in the two distributions. These plots can reveal outliers, differences in location and scale, and other differences between the distributions.

In this case, the functions qqplot() and abline() of the R tool have been used. Each point (x,y) is a plot of a quantile of one distribution along the vertical axis (y-axis, or Asyn mode) against the corresponding quantile of the other distribution along the horizontal axis (x-axis, or Syn mode). The shape comparison can be clarified by plotting a straight line on the Q-Q plot with the abline() function. The syntax is abline(c(intercept,slope)), with ”intercept” and ”slope” replaced by the appropriate values taken from a linear model. The closer the points are to the straight line, the more similar the shapes of the distributions. The Q-Q plot for the two groups appears to be roughly a straight line (represented by a dashed red line), but clearly the locations and spreads differ.

Computation Time

Fig 6.13 shows the relation of the computation time that each controller, included in the k controllers evaluated, spends to discover a partial network topology in both protocol operation modes.

From this figure, it can be seen that for both cases, Syn and Asyn execution modes, the mean computation time decreases as the number of controllers increases. In Asyn mode, when considering more than $k = 4$

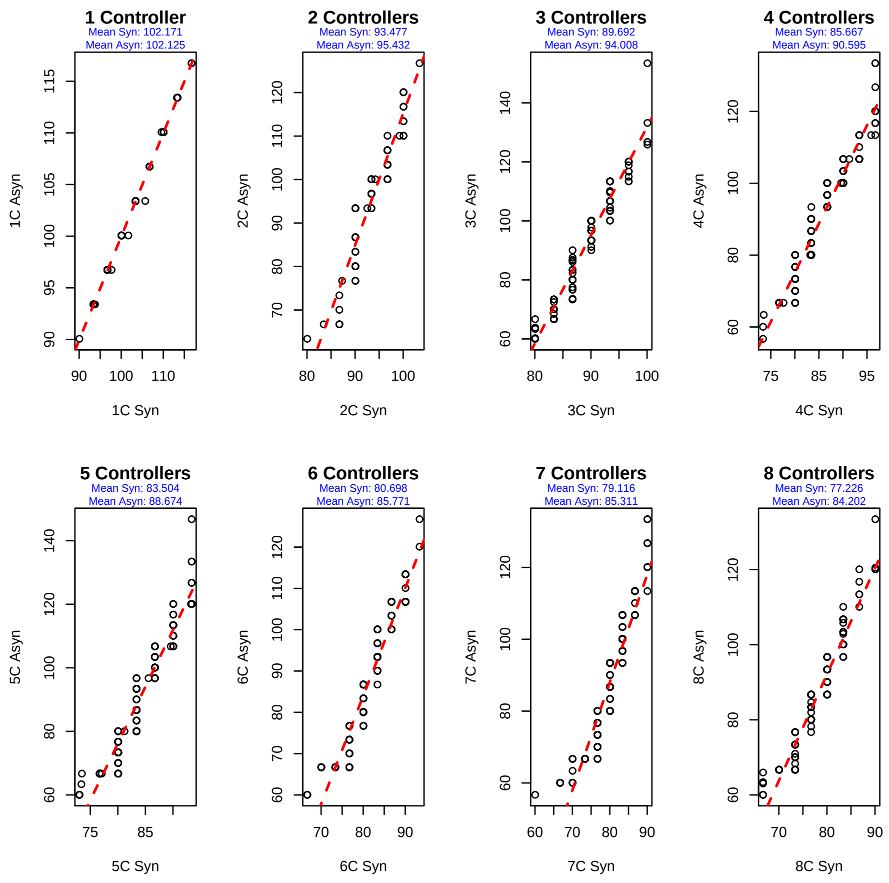


Figure 6.13: Q-Q plots of the computation time (μs) in Syn and Asyn modes for networks that consist of 200 switches.

controllers, the mean time does not decrease significatively, similar to what happened in the Syn mode.

Although the behaviour is similar, the distribution of the dataset obtained from Syn and Asyn protocol mode simulations varies considerably with respect to the linear distribution.

Box plots of Fig 6.14 show the relation between both distributions (Syn and Asyn mode). For all k controllers evaluated, the lowest and the highest topology discovery time is obtained when the protocol is executed in Asyn mode. The highest distribution times are obtained when controllers start the protocol execution sooner, as they tend to discover more switches which translates to a higher delay on the controller (represents the highest time distribution in the box). Therefore, controllers that start the protocol execution later tend to finish the protocol execution sooner (represents the lowest time distribution). In general, the best protocol performance is obtained when it is executed in Syn mode. In this mode, the time the protocol takes to discover the network topology is significatively less than when it is executed in Asyn mode.

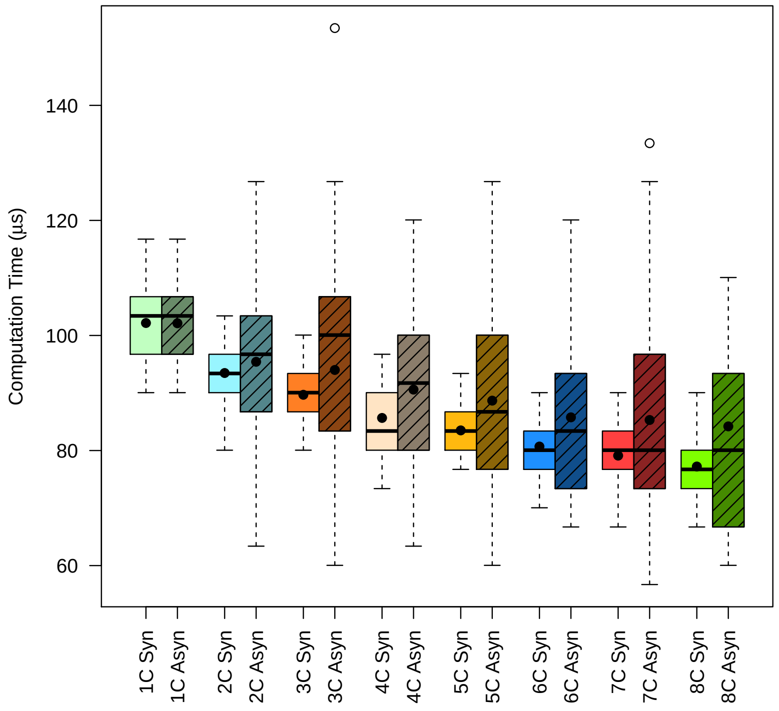


Figure 6.14: Boxplots of the computation time in Syn and Asyn modes for networks that consist of 200 switches. Black points indicate the mean value.

Number of Messages per Switch

Despite the similarity in the mean number of messages per switch, as the number of controllers increases, the number of messages needed to discover the complete network is higher in Asyn mode than in Syn mode. The difference of the number of messages processed by nodes in Syn and Asyn modes is evident when the number of controllers increases, as can be seen in Fig.

6.15.

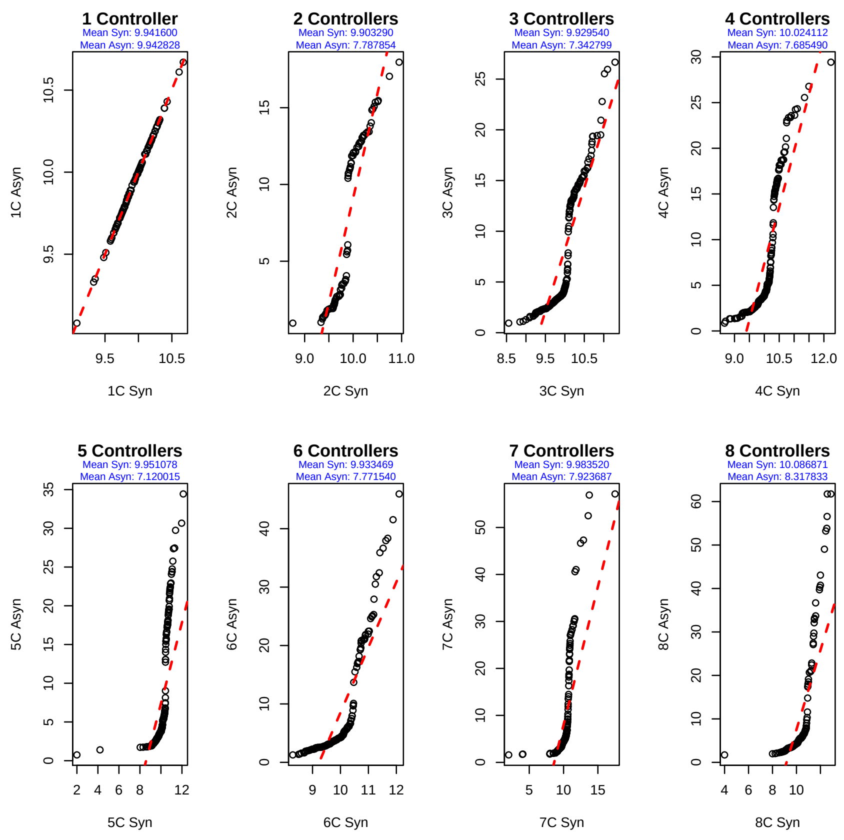


Figure 6.15: Q-Q plots of the number of messages per switch in Syn and Asyn modes in networks that consist of 200 switches.

This is because, when the protocol is executed in Asyn mode, some switches have to process a high number of messages, while other switches have only to process few messages (fewer that in Syn mode). When the protocol is executed in Syn mode, the mean number of control messages that each node in the network processes (send and receive) tends to be constant with respect to the number of controllers as shown in Fig. 6.11(b).

In Asyn mode, it was observed that the controllers were activated by an *AN message* (coming from a controller) before reaching the time programmed to start the protocol execution (1-5 sec). In this scenario, some switches have already made a decision before receiving a message from their closest controller, therefore switches near to the controllers that start the protocol later process few messages. Controllers in this situation discover their partial network topology in few time (as shown in Fig 6.14).

Allocation of Switches to Controllers

Fig. 6.16 illustrates the relation between the allocation of switches to controllers for both protocol operation modes, Asyn and Syn. This figure shows that the mean number of switches distributed for k controllers and for both operation modes is similar.

The results presented in Fig. 6.12(c) show that the allocation of switches to controllers is balanced when controllers execute the protocol in Syn mode. Fig. 6.17 shows that the number of switches managed for k controllers is reduced when k increases, in both cases.

Note that for $k > 5$, there is a high dispersion when more than 50 switches are distributed for each controller. For those cases, controllers that execute the protocol in Syn mode have assigned less that 60 switches. This is represented in Fig. 6.16, where the lineal distribution does not match for the highest switch distribution values.

From Fig. 6.17, it can be seen that when $k < 4$ controllers, the IQR region area (it represents the 50% of the switches distributed) is smaller for controllers that executed the protocol in Syn mode compared with those controllers that execute the protocol in Asyn mode.

The difference between the minimum and maximum switch distribution values is also smaller when the protocol is executed in Syn mode. This means that, when the protocol is executed in Syn mode, the allocation of switches to controllers tends to be more balanced, reducing the dispersion in comparison to the case when the protocol is executed in Asyn mode.

However, as the number of controllers increases ($k > 4$ controllers) the allocation of switches to controllers tends to be similar in both cases. This is expected, since that the network diameter is reduced and in an asynchronous scenario, controllers start the protocol execution before the time they were programmed to start (1-5 s) by receiving an *AN message*.

Fig. 6.18 shows the cumulative distribution functions comparing both execution modes. In this figure, it is also shown that as the number of controllers increases, the switch distribution in Asyn mode tends to be equal to the ideal case (Syn mode), achieving well-balanced network management among controllers.

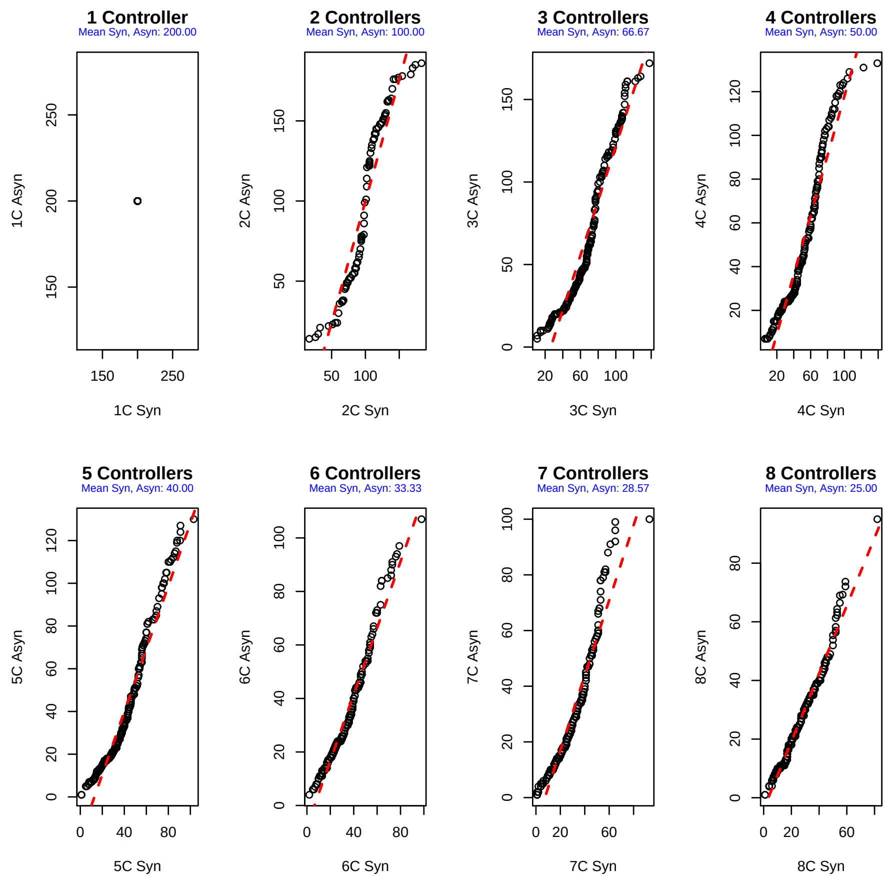


Figure 6.16: Q-Q plots of the number of switches discovered by each controller in Syn and Asyn modes in networks that consist of 200 switches.

Stretch

Fig. 6.19 illustrates the stretch of the switch-to-controller paths found during the protocol execution.

The delay-constrained shortest control path for all switches is found when the protocol is executed in Syn mode, for all the number of controllers evaluated. In this case, the stretch value is always 1. When the protocol is executed in Asyn mode, the switch-to-controller paths tend to be longer than the shortest path in terms of delay; hence, the stretch value is greater than 1.

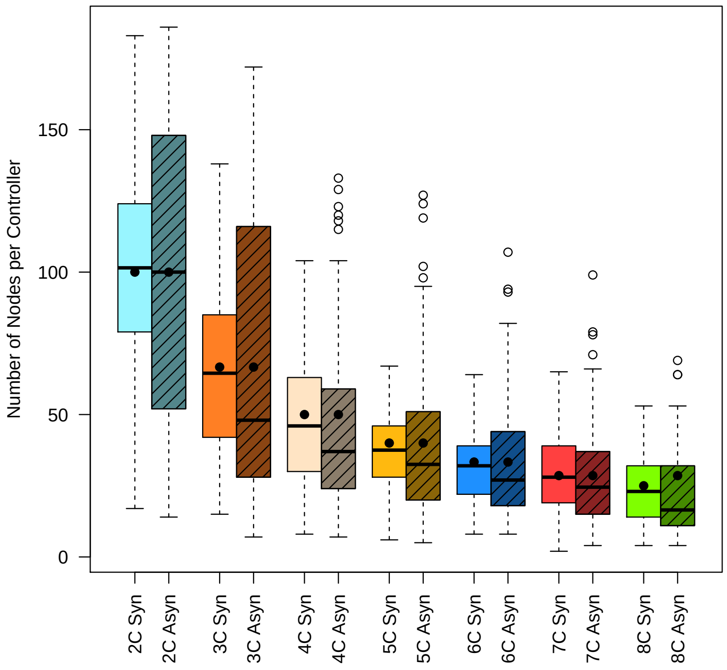


Figure 6.17: Boxplots of the number of switches discovered by each controller in networks that consist of 200 switches. Black points indicate the mean value.

Maximum control path delay

Fig. 6.20 represents the maximum switch-to-controller path delay found by the k controllers evaluated in both operation modes, Syn and Asyn.

Despite the mean of the maximum control path delay found for both modes, Syn and Asyn, and for all number of controllers evaluated is similar, the distribution of the delay values have a wide dispersion (does not match with the straight line), when the protocol is executed in Asyn mode. This is because, in this case, some controllers discover more switches that others (as illustrated in Fig. 6.17), therefore the tendency is that switches whose controllers have more switches discovered, have a longer control path delay. When the protocol is executed in Syn mode, the switches are discovered by their closest controller, therefore, for all cases, in Syn mode the switches have the shortest delay to the controller. This can also be confirmed from results shown in Fig. 6.19. As the number of controllers increases, the dispersion values between both data sets (Syn and Asyn) tends to be higher.

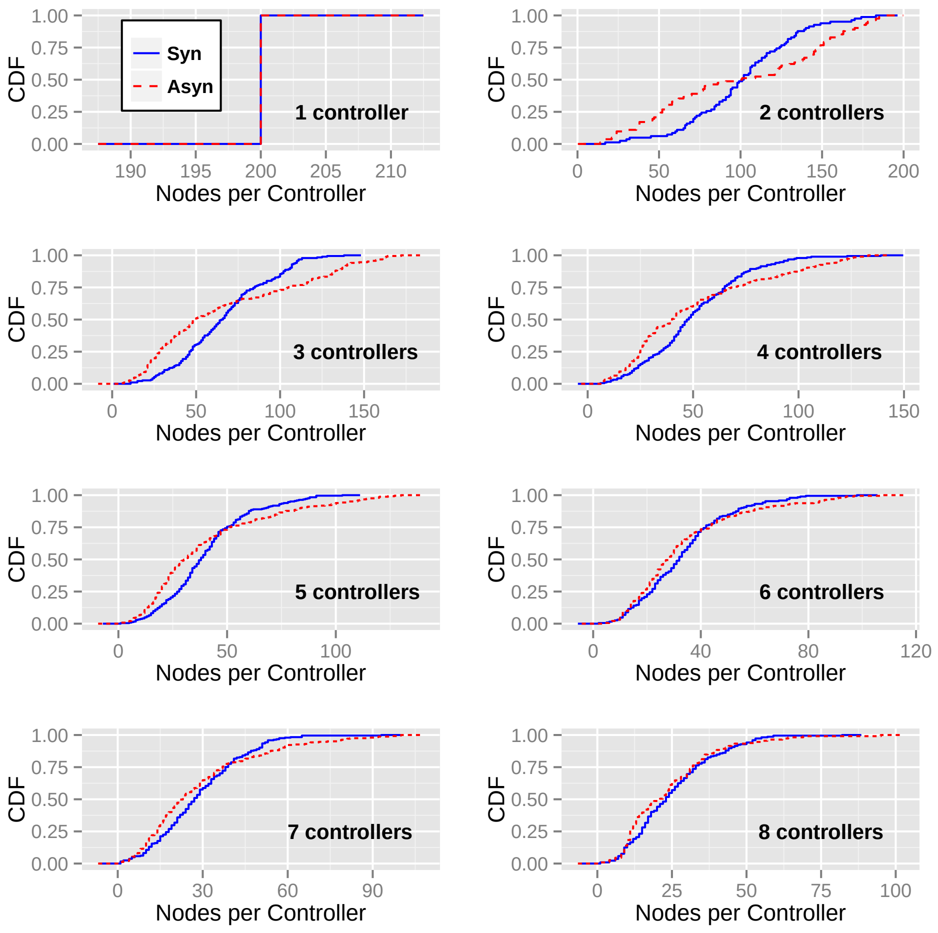


Figure 6.18: Cumulative Distribution Functions of the number of switches discovered by each controller in networks that consist of 200 switches for 1-8 controllers.

Robustness

Fig. 6.21 shows results about the control plane robustness. This parameter is evaluated through the metric defined in Section 5.3. This metric defines the fraction of nodes that could be reconnected with the control plane in the presence of any link or switch failure in the network. The metric value is between zero and one, where zero (0) means that all switches are protected against link and switch failures.

From Fig. 6.21, it can be seen that the control plane created when the

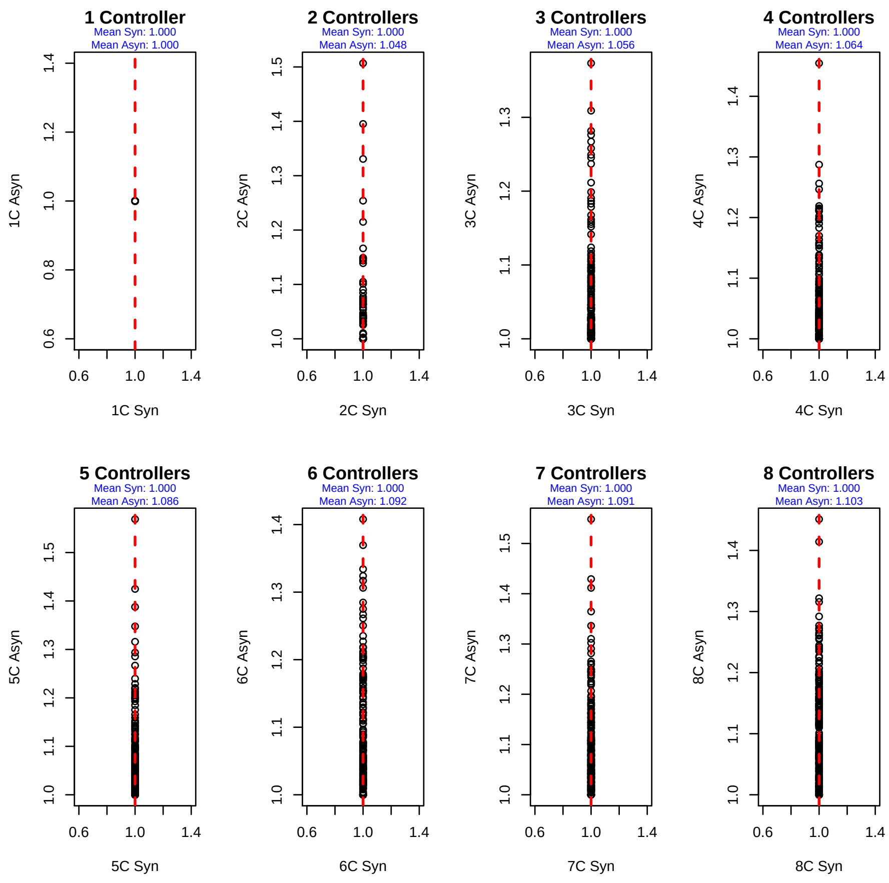


Figure 6.19: Q-Q plots of the stretch in switch-to-controller paths discovered by each controller in Syn and Asyn modes in networks that consist of 200 switches.

protocol is executed in Syn mode is more robust, for all k controllers evaluated. However, as the number of controllers increases, the robustness of the control plane is reduced independent of the protocol operation mode. This is due to the allocation of switches to controllers. When the number of controllers increases, the number of switches discovered per controller tends to be reduced (as illustrated in Fig. 6.17) and therefore, the probability of a switch to re-establish the communication to their controller is also reduced.

Note that for the highest robustness values, the lineal distribution does not match for all number of controllers evaluated. This resulting dispersion

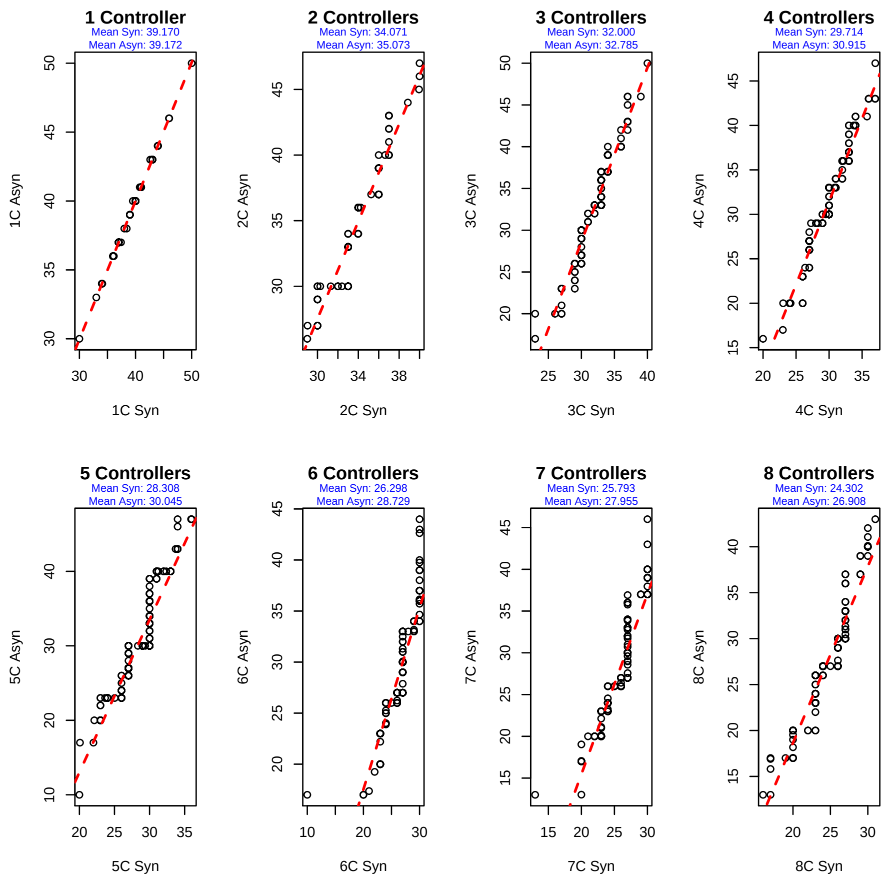


Figure 6.20: Q-Q plots of the maximum switch-to-controller path delay found in Syn and Asyn modes in networks that consist of 200 switches.

can be explained from the switch distribution, since controllers that execute the protocol in Asyn mode discover a high number of switches while other only find few switches. Even though the robustness of the control plane in Asyn mode is not as good as that in Syn mode, the relation between both operation mode (Syn and Asyn) is linear, since the points lie along the straight line, in general.

Note that this metric behaves as expected. When the number of controllers is reduced and the mean number of switches distributed between controllers tends to be homogenous, the control plane is, in general, more robust (with low dispersion) due to the fact that more switches are available,

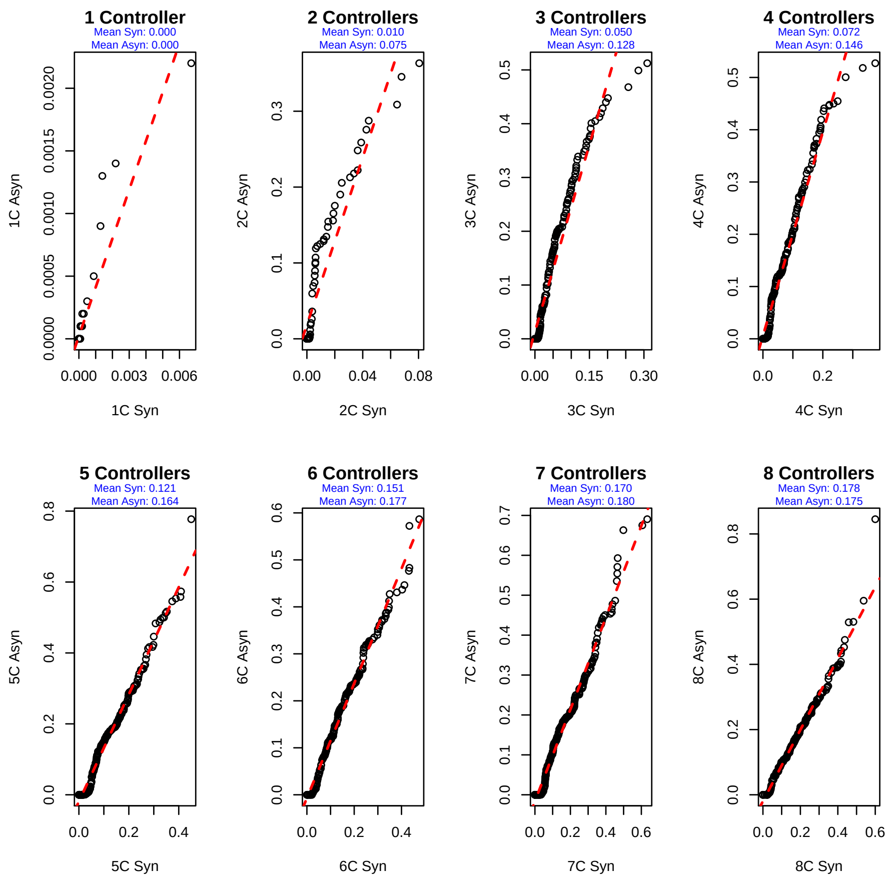


Figure 6.21: Q-Q plots of the control plane robustness found in Syn and Asyn modes in networks that consist of 200 switches.

and thus more redundant paths can be leveraged in case of switch failures.

Expected Data Loss

Fig.6.22 presents a comparison between the maximum expected data loss from the control planes built when the protocol is executed in Syn and Asyn modes. The expected data loss for the control planes created in Syn and Asyn protocol operation modes is similar.

This control plane characteristic depends on the network topology (node connectivity) and the protocol operation mode. This is because control paths for networks with a scarce connectivity may be long in terms of hops, and nodes in a control plane created for networks with a high node connectivity, can have a high number of downstream nodes. For these cases, the expected data loss tends to be high. For the network evaluated, the average expected data loss is similar, since some control plane trees created in Asyn mode have few nodes, while others have a high number of nodes. In the case of a control plane created in Syn mode, the average switch reallocation to controllers tends to be homogeneous, as shown in Fig. 6.17.

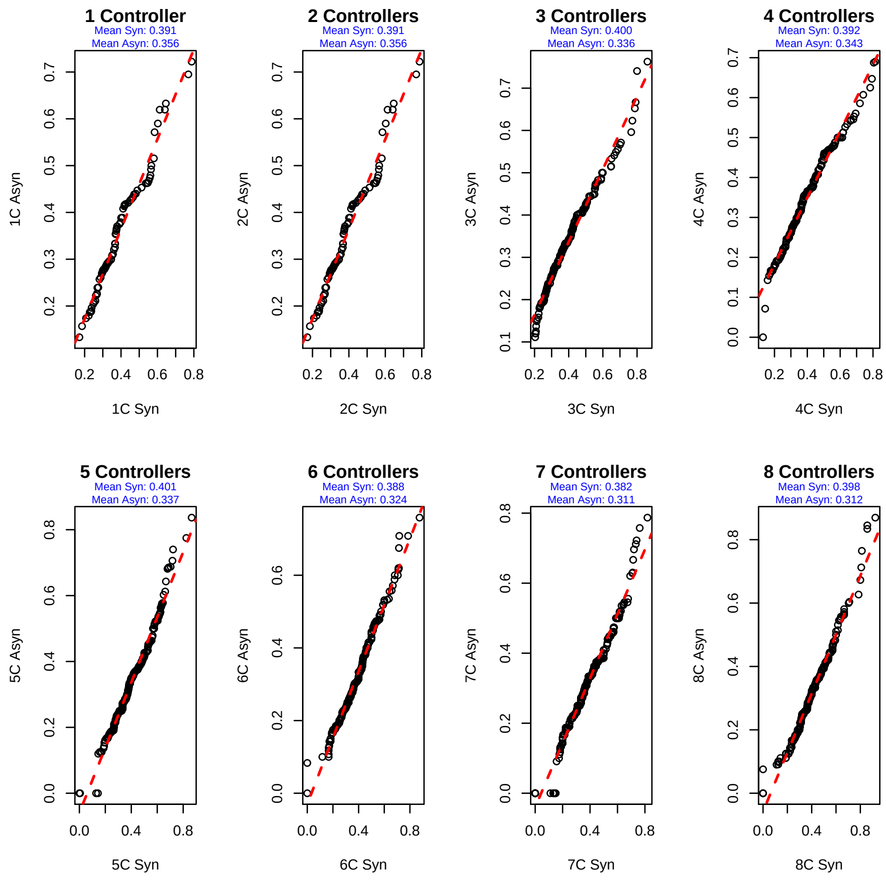


Figure 6.22: Q-Q plots of the maximum expected data loss found in Syn and Asyn modes in networks that consist of 200 switches.

Fig. 6.23 illustrates boxplots of the maximum average expected data loss for the control planes created when the protocol is executed in Syn and Asyn modes.

From this figure, it can be deduced that the nodes in the control plane created when the protocol is executed in Asyn mode have fewer downstream switches compared with the control plane created when the protocol is executed in Syn mode.

Thus, in Asyn mode the control plane has a higher switch-to-controller delay (as shown in Fig. 6.20) but a lower node dependence (downstream nodes), compared with the resulting control plane built in Syn mode.

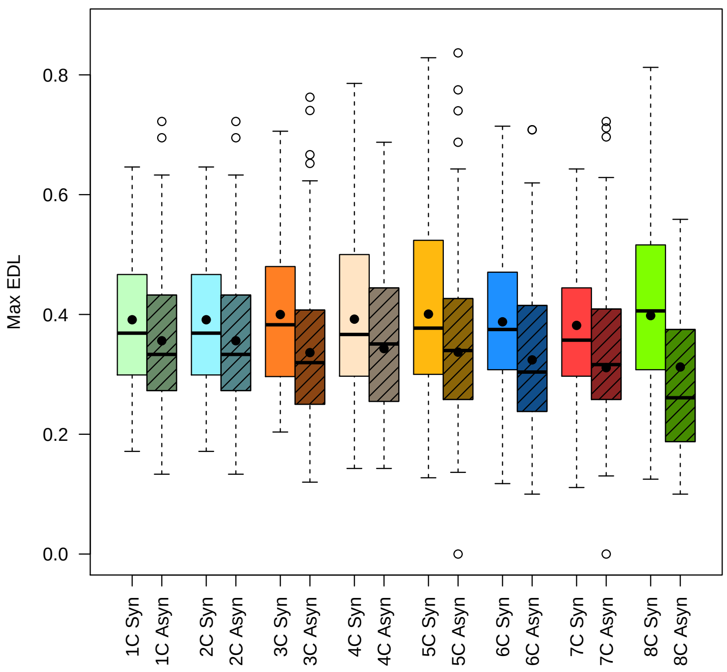


Figure 6.23: Boxplots of the maximum expected data loss in networks that consist of 200 switches. Black points indicate the mean value.

Summarizing all the evaluation, the protocol behaviour is better in Syn mode, but it is also feasible to apply it in Asyn mode in real networks, even with networks of high number of nodes.

Specific example: three controllers

In order to show clearly the characteristics of: i) the protocol when it is executed in Syn and Asyn modes, and ii) the resulting control planes, the

SDN-RDP protocol was executed over a specific network topology that consisted of 200 nodes, from which a set of three controllers were selected using $k-critical$. These are controllers 13, 7 and 62. Table 6.3 presents the results obtained when the SDN-RDP protocol is executed in Syn mode and Table 6.4 presents the results obtained when the SDN-RDP protocol is executed in Asyn mode. In this last case, controllers were programmed to start the protocol execution as follows: controller 7 starts at 1.54 μs, controller 13 starts at 4.1 μs and controller 62 starts at 4.37 μs.

Table 6.3: Control plane characteristics when executing the SDN-RDP protocol in Syn mode over 100 networks that consist of 200 nodes.

Controller	Time (μ sec)	No. Msg	No. Switches	Max. Delay	Mean Delay	Robustness
13	90	9	95	33	18	0.01
7	96	10	47	34	19	0.03
62	90	9	58	33	18	0.02

From Table 6.3, it can be seen that similar results are obtained for each one of the controllers evaluated. Despite the controllers start the protocol at the same time, controller 13 is the one that discovers the highest number of switches and therefore build the most robust control plane. Controller 7 spent the longest time in executing the protocol, however, this is the controller that discovers the lowest number of switches and generates the highest number of messages over the switches. These characteristics are due to the network topology, the SDN-RDP protocol execution mode, and the connectivity of the switches discovered by each controller.

Table 6.4: Control plane characteristics when executing the SDN-RDP protocol in Asyn mode over 100 networks that consist of 200 nodes.

Controller	Time (μ sec)	No. Msg	No. Switches	Max. Delay	Mean Delay	Robustness
13	107	3	23	30	15	0.17
7	113	19	83	37	22	0.0
62	107	4	94	33	20	0.01

When the protocol is executed in Asyn mode, controllers need more time to discover their partial network topology and to create their control plane than when executing the protocol in Syn mode. It was observed that controller 62 started the protocol before the time it was programmed, it starts at 1.55 μs instead of 4.37 μs. It was activated through an *AN message* from controller 7. Controllers that start the protocol first (controllers 13 and 7) spent the longest time in executing the protocol, discovering their partial network topology and creating their control plane trees. These controllers discover the switches that have the longest delay in the resulting control plane. Despite that controller 62 starts the protocol after controller 7, it finds the highest number of switches. However, controller 7 builds the most robust control plane tree. Control plane robustness not only depends on the number of switches each controller manages, but also on the switch connectivity.

From Tables 6.3 and 6.4, it can be seen that the protocol performance and control plane characteristics not only depend on the network topology, but also on the controller placements and protocol execution mode. Results shown in these tables are coherent with the results presented previously.

- In Syn mode, the time the protocol takes to discover the network topology is significatively less than when it is executed in Asyn mode.
- The average number of messages on each switch tends to be constant when the protocol is executed in Syn mode. In Asyn mode, the average number of messages on switches has a high dispersion.
- On average the switches distributed among controllers is higher and balanced when controllers execute the protocol in a Syn manner.
- The shortest control path for all switches is found when the protocol is executed in a Syn manner.
- Robustness not only depends on the number of switches discovered by each controller but also on the switch connectivity.

From the results presented, it can be seen that the controllers are selected to guarantee a switch-to-controller delay lower than 37 μs.

6.7.4 Evaluation of the scalability of the SDN-RDP protocol

In order to evaluate the scalability of the SDN-RDP protocol, a set of generated graphs of different size, 50, 100, 200 and 500 switches, are used. The protocol is executed in Syn mode and the number of controller varies from 1 to 8.

SDN-RDP scalability

Fig. 6.24(a)(b) shows that networks with high connectivity (or density) reduce the control plane creation time with respect to networks with low connectivity. In networks with high connectivity, the number of forwarding hops of the messages decreases because the network diameter is small. Thus, the *AN message*s are quickly spread, discovering the leaf switches in few hops.

Number of messages per switch

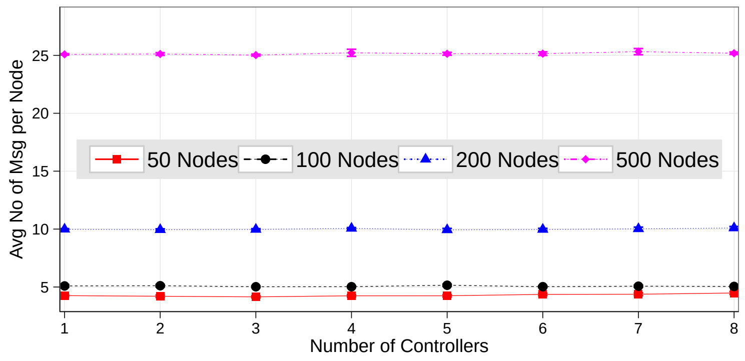
Fig.6.24(b) show that networks with high connectivity (or density) process a higher number of messages per each switch than networks with low connectivity, and also reduce the control plane creation time with respect to networks with low connectivity. This behaviour can also be compared with the protocol complexity presented in section 6.3.4. That is, as the number of controllers increases the protocol complexity is reduced.

In networks with high connectivity, the number of forwarding hops of the messages decreases because the network diameter is small. Thus, the *AN message*s are quickly spread, discovering the leaf switches in few hops. Networks with low connectivity reduce the message conflicts in the network, but increase the construction time of the control plane since the diameter is large. That is, the convergence time of the response messages to the controller is limited by the longest delay time of all the shortest paths found. In this scenario, for each network size, the average number of messages on each switch remains constant even if the number of controllers is low (Fig.6.24(a)). This is because, i) conflicts are solved using implicit response messages and ii) IAN messages are not required as switches receive the AN message through their shortest path.

The number of messages that switches process (forward and receive) during the protocol execution depends on the switch connectivity, the number of controllers in the network and the protocol execution mode (Syn or Asyn).

Controller selection

The implications of the controller selection in the $SDN-RDP$ performance are also evaluated. Fig. 6.25 shows the control plane creation time for different number of controllers selected randomly (dashed lines) and using $k-Critical$ (solid lines). As can be seen in Fig. 6.25, the control plane creation time required by controllers selected randomly was significantly higher than the time spent by controller selected using k-Critical in all cases.



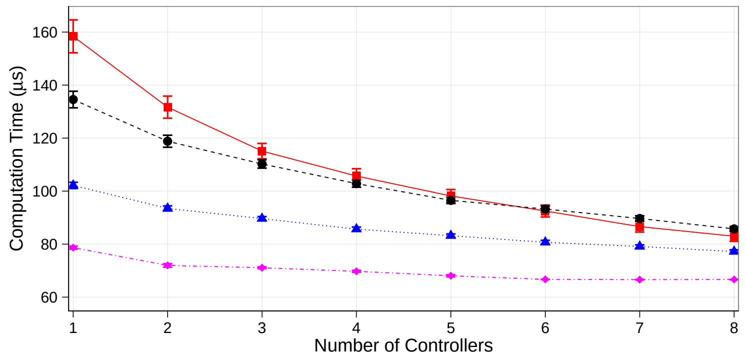


Figure 6.24: (a) Mean number of messages to create a control plane (b) Control plane computation time. (Networks with different sizes, varying the number of k-Critical controllers.)

As controllers selected randomly may be geographically close or far way among them, some controllers may have more load than others, increasing the creation time of the control plane and therefore affecting the SDN-RDP performance.

The average number of messages on each switch, for each network size, is slightly higher when the control plane is created by controllers selected randomly, but in both cases (controller selected randomly and using $k - critical$) the average number of messages remains constant on the switches.

$SDN - RDP$ has been designed to be scalable over a wide range of network sizes and operational modes (Syn and Asyn). Presented results indicate that, for a given network, the average number of required messages to create the control plane is invariant with respect to the number of controllers for networks of any density when the protocol is executed in Syn mode. However, when the protocol is executed in Asyn mode, switches process more messages when $k \leq 4$, but when k increases the number of messages required is less than those processed by switches in Syn mode.

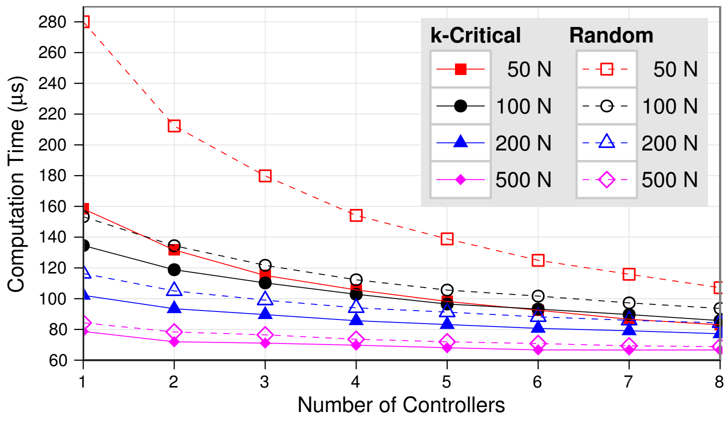


Figure 6.25: Control plane computation time; CI are omitted to improve readability. (Different number of switches, k-Critical and randomly selected controllers.)

With respect to the control plane computation time, improvement in the case of networks with low connectivity is only observed when multiple controllers are used. However, the control plane creation time is reasonable even when using just 1 controller. According to the evaluation, it can be deduced that the resulting control planes deal with real network requirements for all network sizes evaluated for both cases, Syn and Asyn modes. For instance, the data plane fault recovery may be achieved in a scalable way within 50 ms, the time required in transport networks.

6.8 Conclusions

In this chapter a set of mechanisms was presented to discover the network topology and maintain it updated, that includes:

1) the $SDN - RDP$ protocol through which controllers discover the network topology in a distributed way,

2) a network discovery mechanism that enables each controller to discover the complete network topology,

3) a mechanism that reallocates switches-to-controllers is executed locally by each controller to select the set of switches to be managed by them,

4) finally, a collaborative mechanism to maintain the network topology updated on the control plane, where switches detect network changes that are announced to their controllers, while the controllers with this information are capable of discovering what the event is and updating the network topology information on the control plane.

First, the SDN-RDP protocol distributes the network management among k controllers, creating a control plane on top of them. The resulting local control plane on each controller is the shortest tree, as each switch is discovered by its nearest controller. Through these paths or control paths the switches:

- announce their switch connectivity to their controller, and
- can be monitored by their controller,

Although switches are assigned to a controller taking into account the delay, other parameters may be considered when building the control plane. The results obtained show that the proposed protocol works efficiently on large networks in terms of time and load. SDN-RDP is a protocol designed to discover the SDN network topology from scratch, independently of any communication protocol and network topology.

Second, as a result of the SDN-RDP protocol execution, each controller has limited information about the network topology, therefore a mechanism that enables controllers to exchange the partial network information discovered is defined. This mechanism is executed locally by each controller, which discovers from the partial network information known, a path to communicate with at least one of its neighbour controllers. Each time controllers communicate with each other they get information about the existence of at least one controller and the neighbour controller through which it is connected. Each controller asks the partial topology information from the discovered controllers and as a result, each controller discovers the whole network topology information.

Third, as a consequence of the asynchronous SDN-RDP protocol execution, the allocation of switches to controllers is not homogeneous (in terms of load or delay). To get a fair switch distribution, a mechanism that redistributes the load on controllers is defined. This mechanism is executed locally by each controller that discovers, based on a defined parameter, the switches that each controller should manage. After computing the switch distribution, the controllers exchange the result obtained. Ideally, the result obtained by each controller should be the same given that all controllers share the same network topology information.

The traditional fault detection mechanisms executed by both controllers and switches can not be applied to maintain the network topology updated on a control plane, because these approaches:

- were designed with a different purpose, and
- only detect that a failure has happened, but do not identify the kind of failure and its location.

Therefore, a mechanism to keep the network topology updated has been designed, which: i) detects network changes, ii) discovers what the event is about, and ii) announces the events to the responsible controller.

- this mechanism uses a switch-based fault detection to discover network changes in the network that are announced to the controller,
- a controller-based fault detection is also considered that is executed to discover what the event is about, if this is required,
- in addition, controllers can communicate to each other events. When a controller receives an event that is not related to the switches it manages, it can announce this event to the responsible controller.

The set of approaches presented in this chapter to distribute and update the network topology has several desirable characteristics:

- the ability to monitor the complete network in an efficient time,
- the detection and location of any link and switch failure in the network,
- the exclusion of controllers in the monitoring process,
- the maintenance of information on the control plane updated.

This fault detection mechanism together with the SDN-RDP protocol, can maintain the network topology updated on the control plane.

Results presented in this chapter show that the SDN-RDP protocol is scalable and efficient, when it is executed in both a synchronous and an asynchronous way. There is a tradeoff between the switch-to-controller delay and load balance over controllers for any kind of network topology. The network discovery process for networks with a high connectivity takes less time than networks with sparse connectivity. However, the number of messages required to discover a dense network is higher than in the case of networks with a sparse connectivity.

Chapter 7

Conclusions

This manuscript has addressed some of the most challenging problems in the management of SDNs. All the contributions are original, designed from scratch. This means that non-existing mechanisms were adapted for their design.

Research on SDN networks, their properties, requirements and challenges to tackle the current operational problems in the traditional networks have been intensively studied during years, in particular since the ONF defined the standard in 2000. This standard is focused on the communication operations between a centralized controller and the switches, by means of the OpenFlow protocol.

Programmable networks like SDN have been evolved quickly, given that these type of networks are able to manage i) the increasing heterogeneous traffic on the network and, ii) the operation between different network technologies and network providers. SDN has been proposed as a promising solution to solve most of the current network problems related to its management. However, it is still necessary to solve different network challenges related to the i) controller scalability, ii) control plane robustness and, iii) discovery of the network topology, among others. This thesis proposes a solution for each one of these challenging topics.

In SDN paradigm, controllers play a relevant role. These are considered as the brain of the network, provided that they program the switches and consequently, define the network behaviour. Therefore, any aspect related to the controller, i.e., its location with respect to switches and between them, its capacity, traffic managed, affects the network performance. This manuscript describes the implications of the controller placement in different aspects, such as: setup time, recovery time and robustness.

The controller placement in a network must be selected carefully based on not only the application requirements but also the provider requirements. It can be assumed that a controller has enough capacity to manage the switches and it could be located in any network place.

7.1 Summary of Contributions

The main contributions of this thesis are:

- In Chapter 4, the scalability of the controller is tackled from the point of view of the controller placement. In this chapter, a novel controller placement approach called $k - critical$, has been proposed. It has been designed to find both, the number of controllers needed to satisfy a defined network requirement and their placement to create a robust control plane for any kind of network topology. This approach can be applied to find the controller location in a scalable way on any network topology. It is because, $k - critical$ limits the network area where a controller is searched, reducing its complexity in terms of number of processes and resolution time. To compare the performance of $k - critical$ against existing clustering solutions, such as $k - center$ and $k - median$, the $k - critical$ algorithm has been stated and modeled as a heuristic problem. Results obtained show that $k - critical$ not only is scalable but also reduces the number of controllers required when comparing it with the aforementioned approaches.

- In Chapter 5, a robustness metric has been proposed. This metric has been designed with two purposes, first, to measure the control plane resilience, and second, to select the controller placement that maximizes the control plane robustness.

  Through this metric, the resilience of any control plane can be measured. To do that, the data loss is computed by considering that one or different failures happen at the same time in the network, and measuring the number of switches that are unprotected or disconnected from their controller. It is considered that a switch is protected if it has a backup path to the controller that is not affected by a network failure. To build a robust control plane, this metric can be used to evaluate the resulting control plane from the set of switches where a controller can be installed.

  The robustness of the control plane created by using the controller placement approach proposed in Chapter II, i.e. $k - critical$, has been evaluated using this robustness metric. The results obtained have been compared with an existing solution, called $Fast - Failover$, proposed with the goal of selecting the controller that maximizes the control plane robustness. Results obtained show clearly that $k - critical$ not only reduces the time resolution, but also selects the controllers that besides building robust control plane also reduce the switch-to-controller delay. It is because, $Fast - Failover$ limits the network protection of a switch to failures of its parent switch and not to failures in any upstream switch, which can lead to a switch disconnection from the controller.

- In Chapter 6, a protocol to discover the network topology in a cooperative way by all controllers, as well as different network mechanisms to maintain the network topology information consistent in the control plane have been proposed, implemented and evaluated. Firstly, a protocol called $SDN - RDP$ executed by one or more controllers to discover any network topology has been designed. This protocol discovers the network topology in an efficient time because it can be executed by several controllers in an asynchronous way, where each controller discovers a partial network topology. In addition, this protocol uses few network resources due to control messages are forwarded in a controlled way through the network, and each switch needs to forward only one message to decide to which controller it is going to join to. This protocol is robust given that it considers network failures can happen during its execution.

  Secondly, controller scalability and consistency of the network topology information is considered through the design of a set of mechanisms, which are: i) the reallocation of switches-to-controlles, ii) network detection failures and, iii) updating of network topology information.

7.2 Further improvement of proposed contributions

Some of the contributions presented in this manuscript have still room for further improvement. In Chapter 4, the *k-Critical* algorithm only considers the delay and the tree topology created from the candidates switches to select the controllers. Future work can be devoted to include other parameters such as traffic and bandwidth, in order to avoid bottlenecks. In addition, as part of the future work, the robustness of the data plane with respect to the control plane should also be considered in the selection of the controller placements. This implies to take into account the switch connectivity with respect to multiple controllers, improving the control plane reliability.

In dynamic networks, a static switch-to-controller configuration may result in uneven load distribution among the controllers. To address this problem, future work can be devoted to design a distributed controller architecture in which switches can change their controller dynamically according to traffic conditions, so that the load on the control plane can be dynamically shifted among them.

A metric to measure the number of switches that can recover the communication to their controller has been proposed in Chapter 5. Future work can include: i) the comparison of this metric with other existing ones and, ii) the modification of this metric by considering several protection paths between a switch to any controller.

The resource discovery protocol RDP-SDN presented in Chapter 6 is, to the best of our knowledge, the first protocol to discover a SDN network topology on the control plane. Some aspects to be considered in future work are: i) the evaluation of this protocol in real network topologies to have a more clear idea about its performance, ii) the implementation of this protocol in real network devices (SDN switches) and, iii) the implementation of the mechanisms proposed in this chapter (the reallocation of switches to controllers, the failure detection mechanisms and the network update mechanism).

7.3 Future Work

Apart from the aforementioned improvements to the contributions presented in this thesis, several key challenges of the management of SDNs remain unexplored.

The main challenge is being able to move today's management practices into the automation realm. The impact on the network performance when an application requests resources from an SDN must be understood. This is important given that SDN needs to replicate traditional functions of capacity planning, monitoring, troubleshooting, security, and other critical management capabilities.

An incipient topic in SDN is the communication between controllers. In a distributed control plane composed of different SDN domains, it is needed the interaction between them to create a whole network view, negotiate end-to-end communications, migrate load, among other functions.

Future networks will become increasingly more heterogeneous, interconnecting users and applications over networks ranging from wired, infrastructure-based wireless (e.g., cellular based networks, wireless mesh networks), to infrastructure-less wireless networks (e.g. mobile ad hoc networks, vehicular networks). As mobile devices with multiple network interfaces become commonplace, users will demand high quality communication service regardless of location or type of network access.

Efficient network management over wireless access networks and heterogenous environments in general, will become essential, and wireless networks may become a prevalent part of the future hybrid Internet. A major challenge for SDNs is the efficient utilization of resources, especially the case in wireless multi-hop ad-hoc networks as the available wireless capacity is inherently limited. The heterogeneous characteristics of the underlying networks (e.g., physical medium, topology, stability) and nodes (e.g., buffer size, power limitations, mobility) also add another important factor when considering routing and resource allocation.

Bibliography

[1] H. Farhady, H. Lee, and A. Nakao, "Software-defined networking: A survey," *Computer Networks*, vol. 81, pp. 79 – 95, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615000614

[2] D. Kreutz, M. Fernando, V. Ramos, P. Veríssimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *CoRR*, vol. abs/1406.0440, 2014. [Online]. Available: http://arxiv.org/abs/1406.0440

[3] K. Butler, T. Farley, P. McDaniel, and J. Rexford, "A survey of bgp security issues and solutions," *Proceedings of the IEEE*, vol. 98, no. 1, pp. 100–122, Jan 2010.

[4] D. Tennenhouse and D. Wetherall, "Towards an active network architecture," in *DARPA Active NEtworks Conference and Exposition, 2002. Proceedings*, 2002, pp. 2–15.

[5] M. Casado, M. Freedman, J. Pettit, L. Jianying, N. Gude, N. McKeown, and S. Shenker, "Rethinking enterprise network control," *Networking, IEEE/ACM Transactions on*, vol. 17, no. 4, pp. 1270–1283, 2009.

[6] M. Kind, F. Westphal, A. Gladisch, and S. Topp, "Split architecture: Applying the software defined networking concept to carrier networks," in *World Telecommunications Congress (WTC), 2012*, March 2012, pp. 1–6.

[7] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, and M. Zhu, "B4: Experience with a globally-deployed software defined wan," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, 2013, pp. 3–14.

[8] L. Tie, T. Hwee-Pink, and T. Quek, "Sensor openflow: Enabling software-defined wireless sensor networks," *Communications Letters, IEEE*, vol. 16, no. 11, pp. 1896–1899, 2012.

[9] L. Li, Z. Mao, and J. Rexford, "Toward software-defined cellular networks," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, 2012, pp. 7–12.

[10] E. Haleplidis, K. Pentikousis, S. DenazisJ, D. Meyer, H. Salim, and O. Koufopavlou, "Software-defined networking (sdn): Layers and architecture terminology. rfc 7428," 2015.

[11] O. A. working group, "Sdn architecture, onf tr-521," 2016.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[13] D. Avri, H. S. Jamal, H. Robert, K. Hormuzd, W. Weiming, D. Ligang, G. Ram, and H. Joel, "Forwarding and control element separation (forces) protocol specification. rfc 5810 (proposed standard)," 2010.

[14] T. Limoncelli, "Openflow: A radical new idea in networking," *Communication. ACM*, vol. 55, no. 8, pp. 42–47, Aug. 2012. [Online]. Available: http://doi.acm.org/10.1145/2240236.2240254

[15] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-defined internet architecture: Decoupling architecture from infrastructure," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XI. New York, NY, USA: ACM, 2012, pp. 43–48. [Online]. Available: http://doi.acm.org/10.1145/2390231.2390239

[16] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent updates for software-defined networks: Change you can believe in!" in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets-X. New York, NY, USA: ACM, 2011, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/2070562.2070569

[17] S. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 136–141, February 2013.

[18] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 4, pp. 1955–1980, Fourth quarter 2014.

[19] J. Manar, S. Taranpreet, S. Abdallah, A. Rasool, and L. Yiming, "Software-defined networking: State of the art and research challenges," *CoRR*, vol. 1406-0124, 2014. [Online]. Available: http://arxiv.org/abs/1406.0124

[20] H. Akram, G. Aniruddha, B. Pascal, S. Douglas, and G. Thierry, "Software-defined networking: Challenges and research opportunities for future internet," *Computer Networks*, vol. 75, Part A, pp. 453 – 471, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614003703

[21] A. Ian, L. Ahyoung, W. Pu, L. Min, and C. Wu, "A roadmap for traffic engineering in sdn-openflow networks," *Computer Networks*, vol. 71, pp. 1 – 30, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614002254

[22] H. Fei, H. Qi, and B. Ke, "A survey on software-defined network and openflow: From concept to implementation," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 4, pp. 2181–2206, Fourthquarter 2014.

[23] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A retrospective on evolving sdn," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 85–90. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342459

[24] H. Jie, L. Chuang, L. Xiangyang, and H. Jiwei, "Scalability of control planes for software defined networks: Modeling and evaluation," in *Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of*, May 2014, pp. 147–152.

[25] R. Veisllari, N. Stol, S. Bjornstad, and C. Raffaelli, "Scalability analysis of sdn-controlled optical ring man with hybrid traffic," in *Communications (ICC), 2014 IEEE International Conference on*, June 2014, pp. 3283–3288.

[26] F. Bari, A. Roy, S. Chowdhury, Z. Qi, M. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Network and Service Management (CNSM), 2013 9th International Conference on*, Oct 2013, pp. 18–25.

[27] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in

Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, ser. Hot-ICE'12, 2012, pp. 10–15.

[28] D. Erickson, "The beacon openflow controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 13–18.

[29] K. Masayoshi, S. Srini, P. Guru, A. Guido, L. Joseph, v. R. Johan, W. Paul, and M. Nick, "Maturing of openflow and software-defined networking through deployments," *Computer Networks*, vol. 61, pp. 151 – 175, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138912861300371X

[30] S. Shah, J. Faiz, M. Farooq, A. Shafi, and S. Mehdi, "An architectural evaluation of sdn controllers," in *Communications (ICC), 2013 IEEE International Conference on*, June 2013, pp. 3504–3508.

[31] K. Phemius and M. Bouet, "Openflow: Why latency does matter," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on.* IEEE, 2013, pp. 680–683.

[32] C. Andrew, M. Jeffrey, T. Jean, Y. Praveen, S. Puneet, and B. Sujata, "Devoflow: Scaling flow management for high-performance networks," in *In ACM SIGCOMM*, 2011.

[33] S. H. Yeganeh and G. Yashar., "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. ACM, 2012, pp. 19–24.

[34] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, ser. INM/WREN'10, Berkeley, CA, USA, 2010, pp. 3–3.

[35] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.

[36] S. James, H. David, C. Egemen, J. Abdul, R. Justin, S. Marcus, and S. Paul, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Networks*, vol. 54, no. 8, pp. 1245 – 1265, 2010.

[37] A. Schaeffer-Filho, P. Smith, A. Mauthe, and D. Hutchison, "Network resilience with reusable management patterns," *Communications Magazine, IEEE*, vol. 52, no. 7, pp. 105–115, July 2014.

[38] N. Van Adrichem, B. Van Asten, and F. Kuipers, "Fast recovery in software-defined networks," in *Third European Workshop on Software Defined Networks, EWSDN 2014, Budapest, Hungary, September 1-3, 2014*, 2014, pp. 61–66. [Online]. Available: http://dx.doi.org/10.1109/EWSDN.2014.13

[39] F. Ros and P. Ruiz, "On reliable controller placements in software-defined networks," *Computer Communications*, 2015.

[40] ——, "Five nines of southbound reliability in software-defined networks," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14, 2014.

[41] M. Tatipamula, N. Beheshti-Zavareh, and Y. Zhang, "Controller placement for fast failover in the split architecture," 2014. [Online]. Available: http://www.google.com/patents/US8804490

[42] Y. Jiménez, C. Cervelló-Pastor, and A. García, "On the controller placement for designing a distributed sdn control layer," in *Networking Conference, 2014 IFIP*, 2014, pp. 1–9.

[43] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in sdn-based core networks," in *25th International Teletraffic Congress (ITC)*, Shanghai, China, sep 2013.

[44] H. Yannan, W. Wang, G. Xiangyang, Q. Xirong, and S. Cheng, "Reliability-aware controller placement for software-defined networks," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, May 2013, pp. 672–675.

[45] G. Tarnaras, E. Haleplidis, and S. Denazis, "Sdn and forces based optimal network topology discovery," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, 2015, pp. 1–6.

[46] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in software defined networks," in *Signal Processing and Communication Systems (ICSPCS), 2014 8th International Conference on*, 2014, pp. 1–8.

[47] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: State distribution trade-offs in software defined networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 1–6.

[48] F. Botelho, F. Valente Ramos, D. Kreutz, and A. Bessani, "On the feasibility of a consistent and fault-tolerant data store for SDNs," in *Proceedings of the 2013 Second European Workshop on Software Defined Networks*, ser. EWSDN '13. IEEE Computer Society, Oct 2013, pp. 38–43.

[49] Y. Guang, B. Jun, L. Yuliang, and G. Luyi, "On the capacitated controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 18, no. 8, pp. 1339–1342, Aug 2014.

[50] H. Yan-nan, W. Wen-dong, G. Xiang-yang, Q. Xi-rong, and C. Shi-duan, "On the placement of controllers in software-defined networks," *The Journal of China Universities of Posts and Telecommunications*, vol. 19, pp. 92–171, 2012.

[51] H. Yannan, W. Wendong, G. Xiangyang, Q. Xirong, and C. Shiduan, "On reliability-optimized controller placement for software-defined networks," *Communications, China*, vol. 11, no. 2, pp. 38–54, 2014.

[52] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12, 2012, pp. 7–12.

[53] T. Ul Huque, G. Jourjon, and V. Gramoli, "Revisiting the controller placement problem," NICTA, Sydney, Australia, Tech. Rep., jul 2015.

[54] Z. Ying, N. Beheshti, and M. Tatipamula, "On resilience of split-architecture networks," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, 2011, pp. 1–6.

[55] A. García, C. Cervelló-Pastor, and Y. Jiménez, "A modular simulation tool of an orchestrator for allocating virtual resources in sdn," *International Journal of Modeling and Optimization*, no. 2, April 2014.

[56] A. S. da Silva, P. Smith, A. Mauthe, and A. Schaeffer-Filho, "Resilience support in software-defined networking: A survey," *Computer Networks*, vol. 92, no. P1, pp. 189–207, Dec. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2015.09.012

[57] H. Rath, V. Revoori, S. Nadaf, and A. Simha, "Optimal controller placement in software defined networks (sdn) using a non-zero-sum game," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, June 2014, pp. 1–6.

[58] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Elasticon: An elastic distributed sdn controller," in *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '14. New York, NY, USA: ACM, 2014, pp. 17–28. [Online]. Available: http://doi.acm.org/10.1145/2658260.2658261

[59] ——, "Towards an elastic distributed sdn controller," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 7–12, Aug. 2013. [Online]. Available: http://doi.acm.org/10.1145/2534169.2491193

[60] G. Liu and C. Ji, "Scalability of network-failure resilience: Analysis using multi-layer probabilistic graphical models," *Networking, IEEE/ACM Transactions on*, vol. 17, no. 1, pp. 319–331, 2009.

[61] N. Beheshti and Y. Zhang, "Fast failover for control traffic in software-defined networks," in *Global Communications Conference (GLOBECOM), 2012 IEEE*, Dec 2012, pp. 2665–2670.

[62] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale sdn networks," *Network and Service Management, IEEE Transactions on*, vol. 12, no. 1, pp. 4–17, March 2015.

[63] S. Rahmat, Z. Jasni, and E. Abdullah, "A comparative agglomerative hierarchical clustering method to cluster implemented course," *CoRR*, vol. abs/1101.4270, 2011. [Online]. Available: http://arxiv.org/abs/1101.4270

[64] V. Yazici, M. O. Sunay, and A. O. Ercan, "Controlling a software-defined network via distributed controllers," in *Proceedings of the Conference on Implementing Future Media Internet Towards New Horizons*,

ser. 2012 NEM SUMMIT. Heidelberg, Germany: Eurescom GmbH, Oct 2012, pp. 16–22.

[65] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi, "SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains," Internet Draft, Internet Engineering Task Force, June 2012. [Online]. Available: http://tools.ietf.org/id/draft-yin-sdn-sdni-00.txt

[66] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *Communications Letters, IEEE*, vol. 19, no. 1, pp. 30–33, Jan 2015.

[67] A. Dekker and B. Colbert, "Network robustness and graph topology," in *Proceedings of the 27th Australasian Conference on Computer Science - Volume 26*, ser. ACSC '04. Australian Computer Society, Inc., 2004, pp. 359–368.

[68] J. Rohrer, A. Jabbar, and J. Sterbenz, "Path diversification for future internet end-to-end resilience and survivability," *Telecommunication Systems*, vol. 56, no. 1, pp. 49–67, May 2014. [Online]. Available: https://cdn.jprohrer.org/documents/publications/Rohrer-Jabbar-Sterbenz-2012.pdf

[69] J. Sterbenz, D. Hutchison, E. Çetinkaya, A. Jabbar, R. Justin, S. Marcus, and S. Paul, "Redundancy, diversity, and connectivity to achieve multilevel network resilience, survivability, and disruption tolerance (invited paper)," *Springer Telecommunication Systems Journal*, 2012, (accepted April 2012). [Online]. Available: https://cdn.jprohrer.org/documents/publications/Sterbenz-Hutchison-Cetinkaya-Jabbar-Rohrer-Scholler-Smith-2012.pdf

[70] [Online]. Available: www.opennetworking.org

[71] F. Andrade, A. Neves Bessani, F. Ramos, and P. Ferreira, "Smartlight: A practical fault-tolerant SDN controller," *CoRR*, vol. abs/1407.6062, 2014. [Online]. Available: http://arxiv.org/abs/1407.6062

[72] D. Peleg., *Distributed Computing: A Locality-Sensitive Approach.* Society for Industrial and Applied Mathematics, 2000.

[73] L. Shuai, W. Hua, Y. Shanwen, and F. Zhu, "Ncpso: A solution of the controller placement problem in software defined networks," in *Algorithms and Architectures for Parallel Processing*, vol. 9530, 2015, pp. 213–225.

[74] E. Borcoci, R. Badea, S. Georgica, and M. . Vochin, "On multi-controller placement optimization in software defined networking-based wans," in *ICN 2015 : The Fourteenth International Conference on Networks*, 2015, pp. 261–266.

[75] [Online]. Available: http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol

[76] U. Kozat, G. Liang, and K. Kokten, "On diagnosis of forwarding plane via static forwarding rules in software defined networks," *CoRR*, vol. abs/1308.4465, Aug 2013. [Online]. Available: http://arxiv.org/abs/1308.4465

[77] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, and P. Skoldstrom, "Scalable fault management for openflow," in *Communications (ICC), 2012 IEEE International Conference on*, June 2012, pp. 6606–6610.

[78] Y. Jiménez, C. Cervelló-Pastor, and A. García, "Defining a network management architecture," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, 2013, pp. 1–3.

[79] S. Schmid and J. Suomela, "Exploiting locality in distributed sdn control," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. ACM, 2013, pp. 121–126.

[80] J. McCauley, A. Panda, M. Casado, T. Koponen, and S. Shenker, "Extending sdn to large-scale networks," *Open Networking Summit*, pp. 1–2, 2013.

[81] M. Bastian, S. Heymann, M. Jacomy *et al.*, "Gephi: an open source software for exploring and manipulating networks." *ICWSM*, vol. 8, pp. 361–362, 2009.

[82] Y. Jiménez, C. Cervelló-Pastor, and A. García, "Dynamic resource discovery protocol for software defined networks," *IEEE Communications Letters*, vol. 19, no. 5, pp. 743–746, 2015.