Contributions to Topology Discovery, Self-healing and VNF Placement in Software-Defined and Virtualized Networks

Thesis submitted in fulfillment of the
requirements for the degree of

Doctor of Philosophy

Universitat Politècnica de Catalunya (UPC)

Castelldefels School of Telecommunications and Aerospace Engineering

Department of Network Engineering

Author: Leonardo Ochoa Aday

Ph.D. Advisor: Cristina Cervelló Pastor

Barcelona, January 2019

Challenges are what make life interesting and overcoming them is what makes life meaningful.

Joshua J. Marine

This work is licensed under the *Attribution-NonCommercial-ShareAlike 4.0 International License (CC BY-NC-SA 4.0)*. To view a copy of this license, visit `https://creativecommons.org/licenses/by-nc-sa/4.0/`.

A digital copy of this document can be downloaded from TDX (Theses and Dissertations Online, `http://www.tdx.cat/`), the repository of theses managed by the *Consorci de Serveis Universitaris de Catalunya (CSUC)* and sponsored by the Government of Catalonia.

Abstract

The evolution of information and communication technologies (e.g. cloud computing, the Internet of Things (IoT) and 5G, among others) has enabled a large market of applications and network services for a massive number of users connected to the Internet. Achieving high programmability while decreasing complexity and costs has become an essential aim of networking research due to the ever-increasing pressure generated by these applications and services. However, meeting these goals is an almost impossible task using traditional IP networks.

Software-Defined Networking (SDN) is an emerging network architecture that could address the needs of service providers and network operators. This new technology consists in decoupling the control plane from the data plane, enabling the centralization of control functions on a concentrated or distributed platform. It also creates an abstraction between the network infrastructure and network applications, which allows for designing more flexible and programmable networks. Recent trends of increased user demands, the explosion of Internet traffic and diverse service requirements have further driven the interest in the potential capabilities of SDN to enable the introduction of new protocols and traffic management models.

This doctoral research is focused on improving high-level policies and control strategies, which are becoming increasingly important given the limitations of current solutions for large-scale SDN environments. Specifically, the three largest challenges addressed in the development of this thesis are related to the processes of topology discovery, fault recovery and Virtual Network Function (VNF) placement in software-defined and virtualized networks. These challenges led to the design of a set of effective techniques, ranging from network protocols to optimal and heuristic algorithms, intended to solve existing problems and contribute to the deployment and adoption of such programmable networks.

For the first challenge, this work presents a novel protocol that, unlike existing approaches, enables a distributed layer 2 discovery without the need for previous IP configurations or controller knowledge of the network. By using this mechanism, the SDN controller can discover the network view without incurring scalability issues, while taking advantage of the shortest control paths toward each switch. Moreover, this novel approach achieves noticeable improvement with respect to state-of-the-art techniques.

To address the resilience concern of SDN, we propose a self-healing mechanism that recovers the control plane connectivity in SDN-managed environments without overburdening the controller performance. The main idea underlying this proposal is to enable real-time recovery of control paths in the face of failures without the intervention of a controller. Obtained results show that the proposed approach recovers the control topology efficiently in terms of time and message load over a wide range of generated networks.

The third contribution made in this thesis combines topology knowledge with bin packing techniques in order to efficiently place the required VNF. An online heuristic algorithm with low-complexity was developed as a suitable solution for dynamic infrastructures. Extensive simulations, using network topologies representative of different scales, validate the good performance of the proposed approaches regarding the number of required instances and the delay among deployed functions. Additionally, the proposed heuristic algorithm improves the execution times by a fifth order of magnitude compared to the optimal formulation of this problem.

Acknowledgements

I would like to thank everybody who has made me possible to undertake this Ph.D. First and foremost, I would like to thank my advisor Dr. Cristina Cervelló Pastor, who gave me the opportunity to start this amazing adventure in the BAMPLA Research Group. I will forever be thankful for the trust, guidance and knowledge you have provided me throughout my time in Barcelona. Her advice during my doctoral research endeavors for the past three years has continuously forced me to remain focused on achieving my goal. I was extremely fortunate to have her as an advisor.

A special mention goes also to Prof. Sebastià Sallent Ribes. I very much appreciated his enthusiasm, intensity, and willingness to share his technical expertise, even during the long hours at the networking lab. I have benefited immensely from his stress-free mentoring style, both academically and otherwise. Thank you very much for the collaborative atmosphere and all your hints!

My gratitude is also extended to my reading committee members Dr. Chrysa Papagianni and Prof. Reza Nejabati for their interest, technical reviews and helpful comments. A particular thanks also to Dr. Jaime Galán Jiménez, who was a member of my oral defense committee, for the time and effort dedicated to assessing my work.

I am also very grateful to Dr. Paola Grosso for hosting me as a visiting researcher at the SNE Research Group in the University of Amsterdam (UvA). I truly enjoyed all the inspiring discussions, from which I learned a great deal. A large part of Chapter 3 in this Thesis is based on work done while I was in Amsterdam. Special thanks to the fellow researchers at SNE for all the great moments and the inestimable support provided during my Research Stay. The time that I spent in Amsterdam would not be the same without them. The SNE group has been a source of friendships as well as good advice and collaboration.

I would like to express my most sincere thanks to my colleagues from the Network Engineering department at UPC, for their companionship, support and for all the experiences that we have shared (both academically and socially). Thanks also to all people I had the chance to meet along these three years that directly or indirectly inspired me with new ways to conduct my research.

My deepest gratitude goes to my incredible family for bringing me up, educating me and made me the man I am today. For your endless love, support, and encouragement over the years I am extremely grateful. I also would like to thank my brother, for always been a source of real support and for inspiring me to pursue a Ph.D. at the UPC. Although my family knows little about my work technically speaking, I would like to dedicate this to them, especially to my parents who have always wanted me to become a "doctor."

My heartfelt appreciation to my family-in-law, for the words of trust, encouragement and support they have always provided me. Last, and definitely not least, I want to thank my beloved Adri for her unconditional love and care. I could not have imagined having a better companion to undertake this exciting experience.

Author

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

AC	Autonomic Computing.
ANM	Autonomic Network Management.
API	Application Programming Interface.
BA	Barabási-Albert.
BFD	Bidirectional Forwarding Detection.
CAPEX	Capital Expenditures.
CDP	Controller Dependent Proactive.
CI	Confidence Intervals.
CIP	Controller Independent Proactive.
COTS	Commercial off-the-shelf.
CPU	Central Processing Unit.
DNS	Domain Name System.
eTDP	Enhanced Topology Discovery Protocol.
ETSI	European Telecommunications Standards Institute.
ForCES	Forwarding and Control Element Separation.

FPGA Field-Programmable Gate Array.

G-TOP Generalized TOPology.

GNFC Greedy Network Function Consolidation.

ID Identifier.

IETF Internet Engineering Task Force.

ILP Integer Linear Programming.

IoT Internet of Things.

IP Internet Protocol.

IS-IS Intermediate System to Intermediate System.

ISG Industry Specification Group.

ISP Internet Service Provider.

IT Information Technology.

LLDP Link Layer Discovery Protocol.

LOS Loss of Signal.

MAC Media Access Control.

MANO Management and Orchestration.

MAPE Monitor, Analyze, Plan and Execute.

MILP Mixed Integer Linear Programming.

NAT Network Address Translation.

NFC Network Function Consolidation.

NFV Network Function Virtualization.

NFVI	NFV Infrastructure.
NOS	Network Operating System.
OFDP	OpenFlow Discovery Protocol.
OMNeT++	Objective Modular Network Testbed in C++.
OPEX	Operating Expenditure.
OSH	Optimized Self-Healing.
OSPF-TE	Open Short Path First-Traffic Engineering.
PCE	Path Computation Element.
PDU	Protocol Data Unit.
POF	Protocol Oblivious Forwarding.
QoS	Quality of Service.
RADIUS	Remote Authentication Dial-In User Service.
RR	Rapid Recovery.
RTT	Round-Trip-Time.
SD-WAN	Software-Defined Wide Area Network.
SDN	Software-Defined Networking.
SDN-RDP	SDN Resource Discovery Protocol.
SFC	Service Function Chaining.
SHP	Self-Healing Protocol.
SNDlib	Survivable fixed telecommunication Network Design.
sOFTDP	Secure and Efficient Topology Discovery Protocol.

TAP-VNF Topology Aware Placement of Virtual Network Functions.

TEDP Tree Exploration Discovery Protocol.

TLV Type-Length-Value.

ToR Top-of-Rack.

UI User Interface.

VIM Virtualized Infrastructure Manager.

VLAN Virtual Local Area Network.

VM Virtual Machine.

VNF Virtualized Network Function.

VNS Variable Neighborhood Search.

Chapter 1

Introduction

Currently, the enormous demand for Internet services (such as big data, cloud services, the Internet of Things (IoT) and video traffic, among others) across large-scale networks with multi-domains is generating a large amount of revenue for service providers and network operators. In addition to new services, this significant increase in traffic is also a result of the growing number of connected users and communication devices, such as terminal computers, smartphones and sensors [1]. Meeting these exponential demands is, therefore, one of the biggest challenges for network management at present.

To be able to address these high demands from users, network operators will require emerging solutions to effectively manage their network resources in a dynamic and flexible manner. In addition, in order to deploy high-level policies in traditional networks, operators need to configure each element of the network. This often occurs via specific, low-level commands from manufacturers because the plane that determines how to manage traffic (the control plane) and the plane that forwards traffic in accordance with the decisions of the control plane (the forwarding plane) are vertically integrated into a single network device. This feature of current networks significantly hampers innovation and flexibility in network infrastructure.

In order to overcome these issues efficiently, novel mechanisms of autoconfiguration of network elements according to new policies or business requirements are needed, which is an almost impossible challenge for existing Internet Protocol (IP) networks. In this context, the use of programmable networks has attracted attention for its suitability to leverage flexibility and reduce management complexity in network infrastructures.

1.1 Programmable Networks: A Brief Overview

The term "programmable networks" is usually employed to describe the desired future of networking. In essence, a network is said to be programmable if the behavior of its network devices and its traffic control are managed by software that operates independently from the network's physical infrastructure. Moving from closed, proprietary-based computer hardware to software-oriented (and thus programmable) networks provides the opportunity for networking innovation, making it possible and more straightforward to evolve network capabilities and deploy new services.

Commonly used as a synonym for programmable networks, Software-Defined Networking (SDN) is an emerging network architecture intended to address the increasing needs of data centers and campus networks, as well as the requirements of service providers in carrier environments [2]. The term "SDN" was initially used to describe the work behind the OpenFlow project, which was developed in a campus network and oriented to improve network capabilities in order to experiment with novel applications [3, 4]. However, since then, SDN concepts have matured and evolved to become a significant commercial success in large-scale adoptions (e.g. B4 Software-Defined Wide Area Network (SD-WAN) [5]).

In contrast to traditional IP networks, SDN enables researchers and network administrators to design highly nuanced network control that better corresponds to the current and dynamic demands of users. The novel network design proposed by SDN, i.e. decoupling the control plane from the underlying forwarding plane, evolves traditional network infrastructures from configurable to programmable [6]. Consequently, it enables the introduction of new protocols and traffic management models with logically centralized control policies across multi-vendor and multi-layer networks. However, making a network programmable goes beyond separating the control and forwarding planes.

Toward the same goal of boosting network intelligence, the Autonomic Computing (AC) initiative introduces a new concept to leverage self-management properties (i.e. self-configuration, self-optimization, self-healing and self-protection) in complex scenarios seen in modern and heterogeneous environments [7–9]. Inspired by the autonomic nervous system of the human body, AC aims to enable networks to work in a completely unsupervised manner [10]. In essence, autonomic networks should be capable of adapting their behaviors dynamically to meet the specific, changing needs of individual users and high-level application goals [11]. Moreover, AC seeks to dramatically decrease the complexity and costs associated with reliable deployments of network and communication services [12, 13].

To that end, Autonomic Network Management (ANM) [14], a particular type of AC, enables autonomous real-time management of network infrastructures and replaces traditional manual and semi-automatic managing approaches, which are already costly and time-consuming. Besides sharing a common objective, i.e. enabling real-time programmable, self-adaptable and cost-effective networks and services, SDN and ANM can complement each other to provide high-level operator objectives such as enhanced fault-tolerance, cyber-attack mitigation and performance guarantees [15, 16].

In addition, today's networks rely on a mixture of service functions (e.g. firewall, load balancing, Network Address Translation (NAT), intrusion detection, Domain Name System (DNS), etc.) implemented on proprietary hardware appliances. Under such conditions, network operators may benefit from Network Function Virtualization (NFV), an emerging technology that further develops the programmability of the network [17]. Network Function Virtualization transforms current network services in Virtualized Network Functions (VNFs), replacing vendor-specific hardware implementation with software embedded into commodity servers [18]. As a result, VNFs can be placed (or instantiated) within a Virtual Machine (VM) that consumes different resources (e.g. Central Processing Unit (CPU), memory, etc.) from servers built out of special-purpose.

Although NFV and SDN are conceptually independent and each can exist without the other, both technologies have become much closer in recent years and greater value can be obtained by combining them into one networking solution [19–21]. In fact, the envisioned convergence between networking and information technology Information Technology (IT) industries is now possible through advances in SDN and NFV. By taking advantage of the integration of these two complementary technologies, a software-defined and virtualized network can offer greater programmability and automation in service provision and service models [22].

Driven by the recent trends of exponential demand growth and increasing heterogeneous service requirements, operators aim to use the suitability of these technologies (i.e. SDN, ANM and NFV) to provide flexibility, cost effectiveness and easier management of their networks. In this context, these technologies should indeed work together to create an overall programmable network solution where SDN delivers the architecture, ANM drives its management and NFV handles the services.

1.2 Research Problems and Objectives

Research efforts in SDN have, so far, mostly focused on management and traffic engineering tasks performed over OpenFlow-based data planes. Less attention has been paid to improving high-level policies and strategies on the control plane, such as topology discovery and failure management. These elements, however, are becoming increasingly important given the limitations of current solutions for large-scale SDN adoptions.

Discovering network elements in a dynamic and optimized manner and being able to contend with ever-growing network traffic is a key requirement for current SDN environments. In SDN, the controller collects the topology information from the data plane and maintains an abstract view of the entire network. This internal service runs in the background of all controllers and is crucial for the proper functioning of any SDN-managed network as well as supported topology-aware network applications (e.g. network configuration and monitoring, traffic engineering, attack detection and load balancing, among many others). However, there is still the need for an enhanced protocol for automatic discovery and mechanisms of autoconfiguration of network elements according to new policies and business requirements.

In addition, the widespread adoption of SDN in heterogeneous and failure-prone deployments (e.g. data centers, clouds, etc.), is putting increasing pressure on control plane survivability strategies to guarantee the plane's resilience at all times. In fact, improving reliability in SDN has been identified as one of the next crucial objectives for research and industry efforts, and this becomes more challenging when in-band implementations are also considered. Moreover, an adequate solution should also address scalability concerns and allow for rapid responses to network events, requirements that can be met by eliminating controller intervention in the failure recovery procedure. To address these issues, autonomic principles of self-healing can be combined with SDN to develop resilient SDN-managed networks.

Likewise, in NFV the placement of VNF within the physical network is one of the main technical challenges. In addition to considering the limited physical resources of the network elements and their efficient usage, VNFs should be (re)allocated in accordance with the changing pattern of incoming flows. To this end, strategic deployments of VNF can be dynamically achieved by taking into account topology information available at the controller in SDN/NFV environments.

To overcome the aforementioned problems, this research focuses on the design and evaluation of novel techniques related to the topology discovery, self-healing and VNF placement in software-defined and virtualized networks. The main objectives encompassed by this thesis scope have been defined as follows:

1. Design an enhanced topology discovery protocol suitable for SDN environments with multi-domains and in-band control, without the need for prior controller knowledge of the network or specific network configurations of the forwarding devices.

2. Provide an inherent fault-recovery mechanism for control plane survivability while maintaining an accurate global network view at the controller through autonomic principles applied in self-healing SDN environments.

3. Develop a quick and efficient strategy to dynamically allocate chained VNFs, reducing the number of required VNF instances and the delay among deployed functions in software-defined and virtualized networks.

In order to provide support in attaining the main goals, we also specified a set of secondary objectives:

1. Develop mechanisms for supporting multi-domain SDN in the procedure for discovering and maintaining the whole network, thereby dynamically decreasing the control plane overhead associated with network state and topology information.

2. Propose generalized protocol solutions by focusing on the study of frame structures and fault recovery techniques to ensure robustness and compatibility with current SDN systems and services.

3. Develop heuristic and optimization algorithms based on mathematical formulations that are capable of adapting to different autonomic requirements and SDN/NFV system constraints.

4. Evaluate the proposed solutions in an experimental simulation environment to study the practical limits of our design and test the performance of the proposed approaches against the existing baselines in the literature.

1.3 Methodology

The general research strategies utilized in this thesis was undertaken are presented below.

Systematic Literature Review

Being able to identify, appraise and synthesize all the significant research endeavors in the field of topology discovery, fault management and VNF placement in SDN/NFV was a fundamental task in our research. To accomplish this, we applied a fully systematic assessment of the existing literature to select primary studies according to their relevance to our research objectives. Moreover, a thorough review of selected papers enabled us to be aware of the available existing solutions, the implications these solutions have (i.e. contributions and drawbacks) and provided us with theoretical support to propose new methods. By using this procedure it is possible to reduce the time and complexity of reviewing studies and solutions while still achieving a comprehensive study of current developments in the research area.

Protocol Design

This method is a suitable solution for this research as it is intended to improve the topology discovery service and fault tolerance in SDN environments. We sought to provide specific networking rules describing the required steps to ensure reliable operation to discover and maintain an accurate network view for the SDN controller. To accomplish this, several components must be defined, such as communication patterns, the role of each network element, the message format and structure, as well as the communication sequences. In addition, the proposed protocols are intended to be flexible and fully compatible with current SDN-managed ecosystems that are comprised of multiple domains.

Programming Optimization

In order to find exact solutions, optimization models involve the mathematical formulations of the problem, where the objective, decision variables, input parameters and constraints are defined. In this way, complex systems can be represented by analytical or numerical models, including relationships between variables and performance metrics. In particular, Integer Linear Programming (ILP) is a valuable and efficient technique that can be employed easily and supports identifying satisfactory solutions to the considered problem. In addition, practical recommendations based on the computed solution must be provided in order to consider real-world implementations.

Heuristic Algorithms

To address larger scale problems and manage the different system requirements considered in the problem, heuristic approaches are needed to find near-optimal solutions in polynomial time. Although this method will not always identify the optimal solution, effective heuristic algorithms can achieve a high level of concordance with the exact algorithm while reducing convergence times. In addition to the near-optimal results, heuristic algorithms can handle large network sizes for which the exact method cannot find solutions in an acceptable time, improving the scalability of the solution.

Experimental Simulations

The ability to simulate the proposed approaches in an experimental environment, where the practical limits of our theoretical design can be analyzed, was essential to prove the feasibility of our solutions. Likewise, this evaluation method enables further comparing of the performance of our solutions against other existing related approaches. In addition, this procedure yields a faster path to the implementation of the proposed contributions into a real system.

1.4 Summary of the Contributions

The contributions made as part of this research include different fields of knowledge. The main fields of knowledge encompassed by this thesis are SDN, Topology Discovery, ANM, Fault

Tolerance, NFV and Service Function Chaining (SFC). Fig. 1.1 summarizes the relationship between the proposed contributions and their corresponding fields of knowledge.

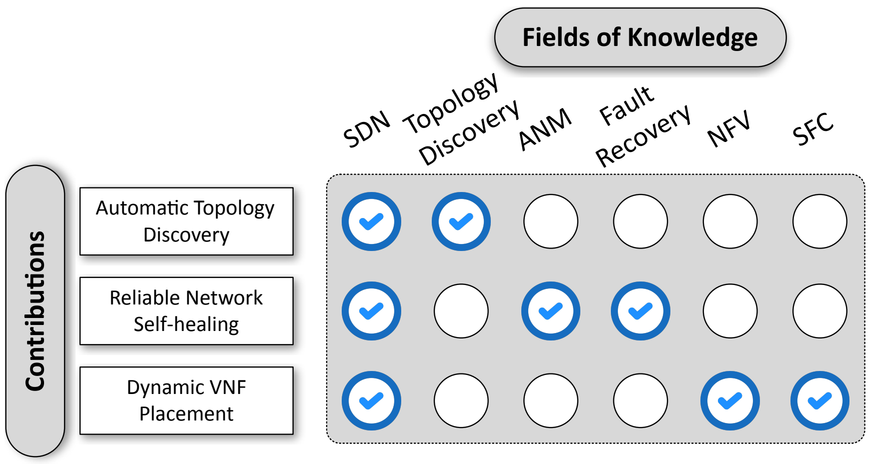


Fig. 1.1: Schema of the Contributions of this Thesis

Our first contribution focuses on topology discovery, which is a critical service provided by the control layer for the proper functioning of applications and network services. This contribution further develops state-of-the-art conventional topology discovery mechanisms in programmable networks. To do this, we optimized the design of the topology discovery service in SDN and extended the boundaries of current OpenFlow-based mechanisms with distributed layer 2 discovery and minimal communication overhead.

In our second contribution, we propose a self-healing technique to boost the control plane resilience in SDN-managed environments without overburdening the controller performance. In essence, the presented solution enables an autonomous real-time recovery of control paths in the event of failures. To the best of our knowledge, the proposed protocol is the first to propose a unified mechanism for discovering physical topology and providing inherent fault recovery in the control plane without the intervention of the controller.

Finally, our third contribution combines topology knowledge with bin packing techniques to efficiently allocate the VNFs in software-defined and virtualized networks. The proposed heuristic algorithm is designed to reduce the number of required instances and the delay among deployed functions while improving the execution times by a fifth order of magnitude compared to the optimal formulation. Additionally, two new metrics (i.e. consolidation and aggregation) were identified to validate the efficiency of the proposal in the experimental simulations.

The contributions focused on topology discovery were presented in [23–25]. The contribution published in [26] also involves the topics of ANM and self-healing. Finally, the contribution presented in [27] integrates the topics of SDN, NFV and SFC. In general, these contributions address all the topics considered in this thesis.

1.5 Outline of the Thesis

In accordance with the thesis scope, this document is structured as shown in Fig. 1.2.

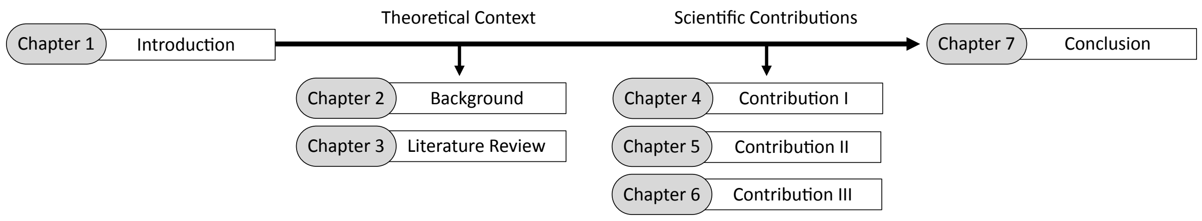


Fig. 1.2: Outline of this Thesis

The Scientific Contributions of this research correspond to the original and exclusive research conducted during the development of this Doctoral Thesis. A more detailed description of the contents of each chapter is given below.

Chapter 2 provides an overview of the leading network technologies that we used throughout this thesis. In particular, the basic concepts behind SDN, ANM and NFV are outlined and some of their most relevant issues, according to this thesis scope, are presented. Some more specific topics regarding the topology discovery process, the self-healing property and the SFC are also discussed. The aim of this chapter is to summarize the fundamental concepts that may be useful for this research in order to quickly introduce the required background knowledge to contextualize the work presented in this document.

Chapter 3 presents the literature review, which contains research relevant to the thesis scope. Specifically, for each considered topic, existing proposals are categorized according to the different features of their corresponding solutions. By doing this, the chapter provides an updated perspective and classification of the current state of research in the field and assesses the main contributions and drawbacks of the research topics considered. Finally, the identified research challenges addressed in this study are discussed, with the innovative aspects of our contributions highlighted.

Chapter 4 describes the design and simulation of an enhanced topology discovery protocol in order to improve the current service in SDN environments. To this end, the operation of the proposed layer 2 protocol is discussed in detail. Experimental simulations with real parameters for this novel protocol are also provided in this chapter. Obtained results show that our enhanced protocol discovers the control topology efficiently in terms of time and message load over a wide range of generated networks.

Chapter 5 proposes a self-healing technique to boost the control plane resilience in SDN-managed environments without overburdening the SDN controller performance. To do this, we leverage forwarding devices with autonomic attributes in order to recover the network from failures in an autonomous and stable fashion by only taking actions at the switch level. As a result, the scalability issues of traditional autonomic systems are avoided and the time required to recover the control plane in the event of failures is reduced.

Chapter 6 presents a novel, low-complexity strategy relying on topology knowledge combined with bin packing techniques to efficiently allocate the VNFs in SDN/NFV environments. As a complement, a general formulation of the network function placement is provided using the SFC concept. The obtained results confirm that the proposed solution delivers a more suitable performance for dynamic cloud-based environments and also outperforms existing approaches based on traditional bin packing schemes.

Chapter 7 summarizes the main conclusions derived from this work. The chapter also exposes some directions for future research.

Chapter 2

Theoretical Background

In this chapter we present some background information about the main technologies used throughout this thesis. In the first section, we describe the SDN architecture, with an emphasis on the topology discovery service. The second section provides an overview of the ANM concept, with special attention placed on the self-healing property and failure management reality in SDN. Finally, the main concepts and characteristics behind NFV, as well as the SFC requirement, are discussed in section three.

2.1 Software-Defined Networks

As a new technological development, SDN has emerged as a promising approach for managing complex and heterogeneous network infrastructures [28]. In essence, this new paradigm proposes decoupling the control plane from the forwarding plane by centralizing intelligence, state of the network and control functions in an entity called the controller or Network Operating System (NOS) [29].

The controller creates an abstraction layer between underlying network infrastructure and business applications while continuously maintaining a global network view. In this way, SDN allows operators to make dynamic configurations and innovations in the network via applications programmed into the top of the SDN architecture, greatly simplifying network operations and management. As a result, network operators and service providers can increase automation and network control, allowing them to build flexible and programmable networks that adapt quickly to the dynamic needs of businesses and end users.

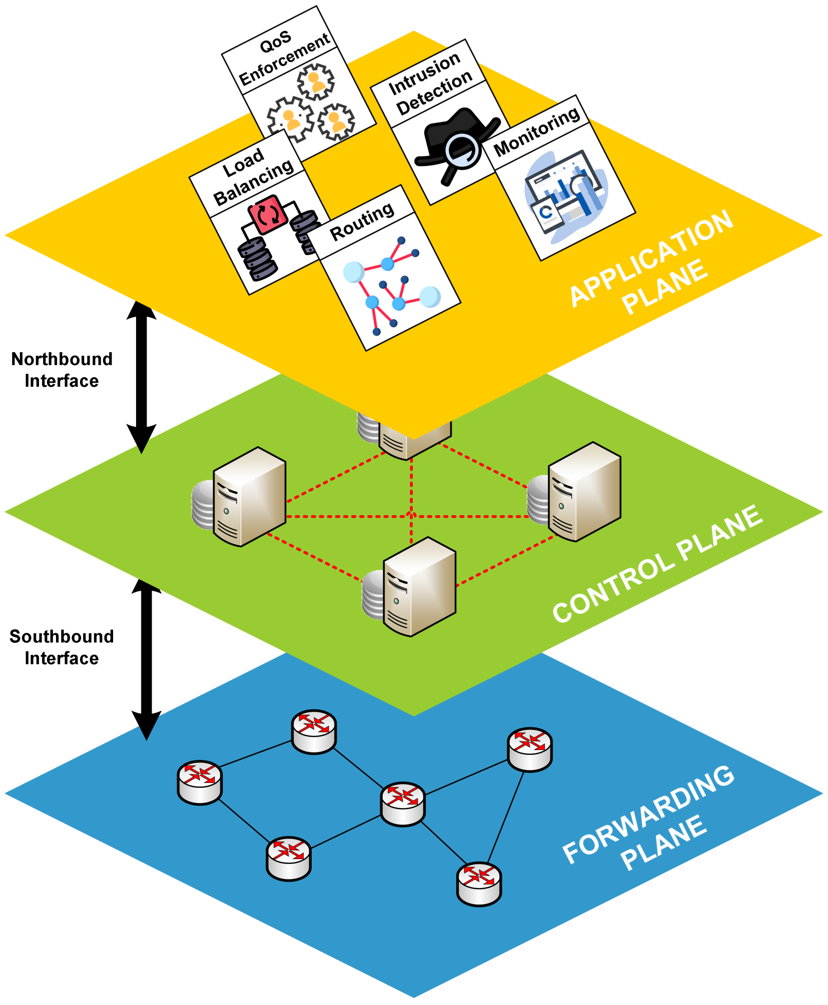


Fig. 2.1: Layered Architecture of the SDN

The logical architecture scheme (in layers) proposed by SDN is shown in Fig. 2.1. The infrastructure layer, or data plane, is composed of forwarding devices (e.g. networking hardware based on Field-Programmable Gate Array (FPGA)) and is situated at the bottom. These physical devices become simple forwarding elements that can be programmed.

The control plane is dedicated to the NOS. This software platform, implemented on commodity server technology, provides the essential resources and abstractions to facilitate the programming of forwarding devices based on a logically centralized approach.

The application layer is at the top of the architecture and contains applications that define the control and logic operation of the network. In essence, this layer allows defining policies and business requirements through programmed applications via a well-defined northbound Application Programming Interface (API). The instructions received from these applications are then used to modify the behavior of forwarding devices based on the established policies.

The exchange of control messages between the NOS and forwarding devices is possible through an open southbound interface. Different protocols have been proposed for this communication (e.g. OpenFlow [3, 30], Forwarding and Control Element Separation (ForCES) [31], Cisco OpFlex [32], Protocol Oblivious Forwarding (POF) [33, 34]). Of this group of APIs, OpenFlow is the most notable example and it is currently the most widely supported protocol by commercial forwarding devices [35]. OpenFlow relies on the existence of flow tables at the switches to handle incoming traffic and the use of basic instructions (e.g. dropping, forwarding, modifying) to define the controller instructions.

Although the SDN architecture is meant to provide centralized control, this capability can also be implemented through the use of multiple servers [36]. Each of these controllers may be organized inside a centralized cluster, or they may be physically distributed across several SDN domains. While distributed control plane implementations are more resilient and scalable, they require reliable and robust communication mechanisms to maintain the required consistency of data updates.

The connectivity between the controllers and the forwarding devices in SDN was initially conceived through a dedicated control network [37]. This operational mode, referred to as out-of-band control, relies on the use of a separate network to interconnect each forwarding device with one controller. Therefore, the control messages are exchanged independently from the data traffic. However, this approach quickly becomes a practical limitation for the deployment of SDN in scenarios with geographically distributed nodes or with cost restrictions. For such scenarios, the in-band control represents the most suitable approach [38, 39]. In this case, an additional network to interconnect the different entities (i.e. the controller and forwarding nodes) is not needed, since the same links are used by both data and control plane traffic. Consequently, novel methods are required to support this connectivity mode concerning topology discovery, control path establishment and fault recovery management.

2.1.1 Topology Discovery Service in SDN

Network topology discovery is a description of the physical structure of the network. This description contains information about the connectivity, latency and link capacity between the network devices at the lowest level (i.e. the data plane layer). This physical infrastructure represents the overall resources of the network.

In general, topology discovery is highly important in several computer network areas such as routing, network management, resource allocation and configuration, Quality of Service (QoS), diagnosis and fault recovery, among others. For this reason, discovering the current topology of a network is a compulsory task for every network operator. Moreover, collecting this real-time information efficiently and automatically is critical for significant networking problems such as enhancing network connectivity and resolving network congestion. In order to improve the performance of these network services, preserving an accurate view of the network topology at all times is also an important task. However, maintaining a comprehensive view of large networks generates a considerable amount of state information from the forwarding plane [40]. Furthermore, a substantial volume of state information represents considerable pressure for the central controller and, as a consequence, scalability issues might appear [41]. Schemes in which each forwarding node must send the topology information periodically could overload controller performance.

In SDN, the controller maintains a holistic view of the network through the topology discovery service. This topological knowledge is crucial for the correct operation of other internal controller services such as hosts tracking and network configuration, as well as for other network applications (e.g. traffic engineering, network monitoring, attack detection and routing protocol, among others) that run on top of the presented architecture [42]. Although discovering the network topology is an essential service of the SDN controller, at the time this writing no official standard defines the topology discovery mechanism in SDN [43]. Due to this issue, most current controllers implement a topology discovery mechanism based on the popular southbound protocol OpenFlow. This discovery mechanism, referred to as OpenFlow Discovery Protocol (OFDP) [44, 45], implements a topology discovery mechanism that uses the frame format defined by the Link Layer Discovery Protocol (LLDP) [46]. Except for the frame format, OFDP does not have much in common with LLDP.

OpenFlow Discovery Protocol is based on packet-out and packet-in messages sent from the controller and switches, respectively. This method does not take real advantage of the existing hardware information that can be directly extracted using LLDP. Moreover, sending messages periodically from the controller to each OpenFlow switch increases the network traffic and latency between the control plane and the forwarding plane, and can also lead to network limitations and outages [47, 48].

2.2 Autonomic Network Management

Recently, there has been growing interest in improving the networks of the future using autonomic principles. In 2001, Paul Horn introduced the concept of AC to the National Academy of Engineers at Harvard University [7]. This idea was inspired by the autonomic operation of the nervous system in the human body, which is able to make independent choices to modify its behavior in the face of different stimuli [10]. This ability to make independent choices defines an autonomic entity and this definition can be applied to multiple contexts.

In terms of computer networks this paradigm has been translated into the ANM. The main idea underlying this proposal is the leveraging of self-management properties (i.e. self-configuration, self-healing, self-optimization and self-protection) in complex scenarios as heterogeneous network environments [49]. The four aforementioned properties, referred to in existing literature as the "self-CHOP properties," collectively define an autonomic system and have been attracting growing attention from both academia and industry.

The general framework defined in ANM follows a similar logic as the SDN architecture. The framework is composed of a set of managed elements distributed in the network, which are centrally governed by an autonomic manager [50]. The former group (i.e. managed elements) is intended to serve as an interface with the system, while the latter (i.e. autonomic manager), manages the different operations of the components. In essence, the autonomic manager is responsible for performing the required adaptations in order to achieve a set of higher-level goals. To accomplish this, a global view and knowledge regarding the managed entities and their management operations are required by this centralized entity.

Besides using its holistic knowledge, the autonomic manager needs to perform four main tasks, namely: monitoring the managed entities, analyzing their performance, plan appropriate management operations and execute them [51]. These functionalities, also known as the Monitor, Analyze, Plan and Execute (MAPE) loop, are of paramount importance to enhance the network performance and obtain solutions to current or anticipated problems. Another approach to attain the autonomic principles is to distribute these functionalities over the set of managed components, which interact with each other to provide convergent autonomic management enabling the self-adaptation to the environment changes.

2.2.1 Self-healing SDN Environments

As previously mentioned, self-healing is one of the four main properties conceived in the autonomic paradigm [52, 53]. This term refers to the capability of the network to restore its operations, independently and without external intervention, when any failure occurs. Every system with self-healing properties has the capability to discover, diagnose and react to failures. The primary objective of integrating self-healing features into any network operation is improving its reliability and maintainability. These quality attributes are traditionally heightened in self-healing systems [54, 55].

As failure management is a critical aspect for every network operation, substantial effort has been devoted to the implementation of different strategies. Traditional failure recovery strategies are classified into two groups, restoration and protection [56]. The former strategy is reactive and requires the online computation and dynamic installation of alternate routes after failure detection. In the protection strategy, however, backup paths are pro-actively configured across the network. Therefore, while the restoration scheme requires higher recovery times, the protection approach imposes higher memory requirements and raises scalability concerns.

Software-Defined Networking provides flexibility to network systems and increases the opportunities for innovation and development, but there is no guarantee that these networks are robust [57]. In fact, implementing crucial functions in SDN environments, such as failure resilience, is a highly complex task given that controller intervention implies non-negligible delays and signaling overheads (due to the propagation delay of failure notifications and reactive failure recovery countermeasures). Moreover, failures in the control plane will have a significant impact on network performance, since these failures may cause that new flow entries cannot be handled promptly. Therefore, failure resilience is clearly one determinant requirement that must be addressed for the successful adoption of SDN technology.

OpenFlow-based fault recovery approaches have been mainly focused on restoring failed data paths by locally detouring individual flows [58–60]. These mechanisms, also referred to as fast-failover techniques, avoid controller intervention during recovery, reducing the incurred recovery time. Although local reactions to failures are faster than path-based end-to-end reallocations, this scheme has some crucial drawbacks. First, it can be used only if alternative path rules are available at the node that detects the failure. Moreover, it requires instantiating multiple alternate path rules for each flow entry on each link, which implies an inefficient resource allocation and may be impractical in some cases. Lastly, in large topologies, an extensive computation of a backup alternative for each flow passing through each node may overload the centralized controller and create a processing bottleneck.

A reliable and scalable mechanism to recover a link or node failure has additional requirements in the context of in-band SDN. With in-band control, an additional physical control network is not needed since the control traffic is sent with the data traffic over the same infrastructure [61]. In such scenarios, failures in the interconnection between forwarding devices are likely to also affect the control plane. In fact, it is highly possible that a failure in a link or node will disconnect several switches from the controller, making the recovery task much more complicated. Therefore, adequate solutions must not only try to recover the control plane connectivity within the shortest possible time, but also be able to achieve this even when the controller is unreachable.

In this work, the term "self-healing SDN environment" is used to refer to a system that proactively monitors its service parameters and network elements in different segments in order to recover from errors after a failure has been detected [62]. In essence, it allows reaction to network component (i.e. links or nodes) failure by reconfiguring traffic allocation in order to make use of the surviving network infrastructure able to provide services. Moreover, based on internal information about appropriate metrics, the system can forecast future service failures and propose preventative actions before the service fails. In this way, it is possible to avoid any outage of essential services such as the network topology discovery.

2.3 Network Function Virtualization

Traditionally, network services have been deployed using proprietary equipment for each function required for a given service. These hardware-based appliances are placed across the network following strict ordering in accordance with the chaining imposed on the service components. Such service modeling results in slow service provisioning, long product cycles and heavy dependence on vendor-specific hardware, leading to high and undesired values of Capital Expenditures (CAPEX) and Operating Expenditure (OPEX).

Motivated by this reality, NFV [17, 63] is an emerging technology based on the standard

IT virtualization concept. In essence, NFV increases the programmability of the network by turning network functions (e.g. routing, firewall, load balancing, NAT, intrusion detection and DNS) into software installations placed on commodity hardware located inside the network. In this way, NFV leverages service provisioning capabilities from specialized hardware to general software hosted on Commercial off-the-shelf (COTS) platforms [64, 65].

By decoupling network functions from proprietary physical equipment, NFV transforms the way a network is conceived. This separation of software and hardware allows for independent evolution, helping to accelerate service innovation and provisioning [66]. Moreover, it eliminates the need for specialized appliances and transforms network equipment into standard components with computing, storage and networking capabilities. Subsequently, the required NFVs can be provided using VMs or containers inside this general purpose equipment.

In summary, the introduction of NFV has the following potential benefits.

- Agile implementation and easier deployment of new network services with reduced time to market and improved service models.
- Flexible computational and networking resource allocation, reducing the need for over-provisioning infrastructures.
- Ability to dynamically scale resource allocations both up and down according to the arrival and departure of traffic flows.
- Automation of operational processes, improving efficiency and reducing network outages due to maintenance-related activates.
- Significant reductions in OPEX and CAPEX through space and energy savings and optimizing infrastructure utilization.

The European Telecommunications Standards Institute (ETSI) is leading the standardization work behind NFV development. Since 2012, the Industry Specification Group (ISG) for NFV has published more than 100 documents (including specifications, use cases and proofs of concept) and the group currently includes over 300 companies. The high-level NFV reference architectural framework [67] proposed by this community is shown in Fig. 2.2.

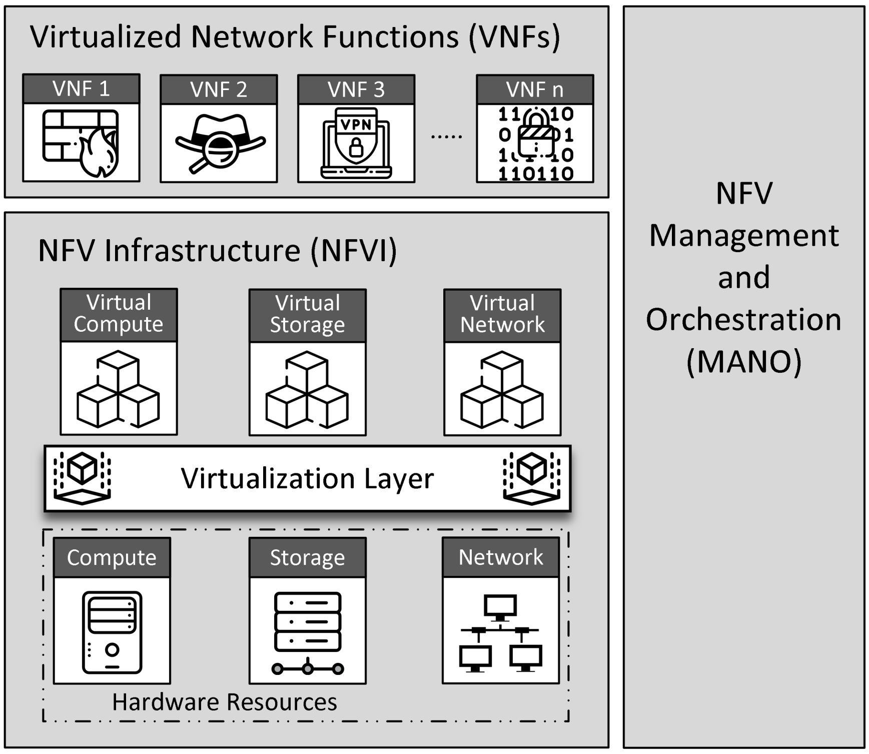


Fig. 2.2: High-level NFV Framework

The high-level NFV framework is composed of three major functional blocks, namely NFV Infrastructure (NFVI), Virtualized Network Functions (VNFs) and NFV Management and Orchestration (MANO) [68].

The NFVI contains all the hardware and software elements included in an otherwise geographically distributed NFV environment. Hence, it is composed of both hardware and virtual resources, which are linked through the virtualization layer. The function of this layer is abstracting the physical resources so that they can be logically partitioned and provided to the virtualized functions.

The VNFs are software implementations of the network functions that are deployed over the virtual resources of the shared NFVI. They may be allocated using one or more VMs running on different physical hosts, according to the resource availability and its efficient usage.

The NFV MANO module performs all the management tasks explicitly required for the virtualization of the NFV framework. To achieve this, the NFV MANO module is comprised of the Virtualized Infrastructure Manager (VIM), which controls the software and hardware components deployed across the NFVI as well as their interactions; the VNF Manager, which focuses on managing the lifecycle of VNFs (e.g. instantiation, update, query, scaling, suspension and termination); and the NFV Orchestrator, which is responsible for the creation and required resource coordination of end-to-end network services.

2.3.1 Service Function Chaining

Deploying a network service over a shared virtualized infrastructure is the ultimate goal of NFV [69]. However, the execution of such a task requires additional considerations according to the network service requirements. In general, network services include a number of network functions that should be applied to incoming traffic flows according to a predefined classification. For example, before reaching its destination given traffic being routed from one location to another may require passing through a load balancer, a firewall and an intrusion detection system.

In addition, in order to enable a complete end-to-end service, the considered functions may be required to occur in a fixed order, creating a specific sequence of logical steps that must be followed by some user requests. This sequence defines the required interconnection between VNFs allocated in the infrastructure and, as a result, it becomes an abstraction of the physical network devices directly connected between them by cables.

The ordered set of two or more instances of network functions interconnected to form a network service that must be applied to some traffic flows, is known as SFC [70]. The aforementioned traffic steering mechanism is based on the prior classification of packets as they enter the network, and their subsequent forwarding is managed by the appropriate network functions defined in the chain.

Taking advantage of the implementation of NFV-based ecosystems, the SFC approach allows operators to deploy tailored services for their customers and simplifies the dynamic allocation of virtual functions according to changing end-user demands and network parameters [71]. An important point to note is that the SFC mechanism is not tied to network services with a linear sequence of VNFs, and SFCs with more than one branch can also be created.

In this context, the optimization of VNF allocation is considered an important goal for achieving efficient implementation of SFC [72]. To do this, different optimization models may be considered for the placement of network services in the given network infrastructure to minimize certain problems (e.g. end-to-end latency, resource utilization, costs and energy consumption).

Chapter 3

Literature Review

This chapter reviews the current state of research regarding the problems addressed by this doctoral thesis. It is divided into three main parts. In each section relevant contributions to the literature are introduced. At the end of this chapter we identify current issues and propose some guidelines to overcome these challenges.

3.1 Topology Discovery in SDN

In this section, we first outline a general survey of OpenFlow-based discovery mechanisms in programmable networks. After this, we provide a brief description of the proposals in existing literature that have considered applying non-OpenFlow solutions to the topology discovery problem in SDN.

3.1.1 OpenFlow-based Approaches

Currently, in SDN infrastructures, after OpenFlow compliant switches are turned on the SDN controller establishes an initial control connection with each forwarding device [3, 73]. This initial handshake is used by the SDN controller to request capabilities from the switches, such as configuration information, the number of active interfaces (i.e. network ports), corresponding Media Access Control (MAC) addresses, etc [73].

After this, the controller initiates the de-facto topology discovery in SDN, called OFDP. For this protocol, the SDN controller begins by sending LLDP frames encapsulated in packet-out messages to each active interface on each OpenFlow switch in the network. By default, after receiving an LLDP packet from ports other than the controller port, each active switch must send a packet-in message that contains the received LLDP to the controller.

Both OpenFlow packet-in and packet-out messages are essential for current topology discovery mechanisms. The number of packet-out messages that an OpenFlow controller must send is equal to the total number of ports in the network. Meanwhile, the total packet-in messages it receives is twice the number of active links in the network, as there is one packet for each direction. Therefore, due to OFDP, the controller load is determined by the number of packet-out and packet-in messages that the controller must process. The related works discussed below propose improving the efficiency of the OFDP mechanism based on the reduction of packet-out messages sent from the SDN controller to OpenFlow switches.

Pakzad et al. [74,75] evaluate the efficiency of the OFDP mechanism implemented by current SDN controllers. The authors propose simple and practical modifications to reduce controller overhead during the topology discovery procedure and implement an OFDPv2 based on the ability of OpenFlow switches to rewrite packet headers. As a result, the number of packet-out messages sent by the controller can be reduced to only one message per OpenFlow switch. In [74], after implementing the improved approach using a POX controller [76] and the Mininet emulator [77], results showed a reduction in the controller overload of up to 45%. Testing the proposed modification in a specific topology in the OFELIA SDN testbed [75] showed a reduction in controller overload of up to 40%, while the physical topology is presented as in the legacy OFDP mechanism.

In [78], the authors revisit the current OpenFlow-based topology discovery protocols, taking into account the effects of retransmitting the discovery packets until the SDN controller identifies the entire map of the network. To that end, the authors re-implemented the OFDPv2 proposed in [74,75] and compared it with the standard OFDP protocol. Experimental simulations revealed different patterns when retransmission of OpenFlow packets is taken into consideration. Specifically, although retransmission doubles the number of required packets, their implementation outperforms the basic OFDP in terms of bandwidth consumption.

The Tree Exploration Discovery Protocol (TEDP) is proposed in [79]. This protocol gathers topology information and simultaneously provides the shortest paths among forwarding devices without adding additional messages, as compared to current OFDP. After describing two possible implementations for TEDP, the authors also list some desired features that should ideally appear in future SDN platforms according to their research.

Azzouni et al. [80] introduce Secure and Efficient Topology Discovery Protocol (sOFTDP) as a novel and efficient alternative protocol to the current OFDP. This proposed protocol requires minimal changes to OpenFlow switch design and eliminates serious vulnerabilities in the topology discovery process. The authors implemented this solution as a topology discovery module in a Floodlight controller [81] and confirmed that it reduces the topology discovery time by several orders of magnitude in proof of concept experiments.

Taking into account multi-controller environments, the authors in [82] designed and implemented an automatic network topology discovery mechanism across various OpenFlow domains. To do this, the authors propose modifications to the discovery, topology and LAVI modules within the NOX controller [83]. Furthermore, modifications to the display User Interface (UI) of the ENVI module were made in order to connect all deployed NOXs and obtain the status of their OpenFlow switches. The authors implemented their solution in a large-scale OpenFlow testbed and the experiment displayed the entire topology across various domains within the same UI.

In [74, 75, 78–80, 82] the controller needs, as previous knowledge, the network identification (i.e. IP address) of each active SDN switch in the network. This knowledge is obtained through the establishment of an initial connection between the controller and each forwarding device. In addition, if the network topology changes, the OFDP mechanism can generate an excessive quantity of state messages. As a result, the network can experience service limitations and outages.

In contrast to this, we propose a topology discovery protocol that enables discovering the network nodes before the initial establishment between the controller and the SDN switches over a secure connection. Based on this procedure, the SDN controller can discover the network topology before it establishes a connection with the switches. Our protocol enables SDN controllers to be connected in already deployed networks without previous configuration, which provides an approach for quick installation.

3.1.2 Non-OpenFlow Approaches

Most topology discovery mechanisms proposed thus far for SDN have focused on improving the current OFDP mechanism. However, other possibilities have also been studied due to the aforementioned limitations of OpenFlow-based solutions. In this study we have also analyzed some research contributions that discover the network topology in SDN using non-OpenFlow mechanisms.

Tarnaras et al. [84] proposed an automatic topology discovery algorithm, which takes into account a better usage of the LLDP protocol. This protocol is used locally on the data plane of switches for updating their local neighbors table periodically. The authors obtained the topology map using the Internet Engineering Task Force (IETF) ForCES framework for extracting LLDP data directly from the network devices (i.e. the LLDP local system management information base). This procedure automatically reports any change in the physical topology to the controller as a triggered event. After implementing the proposed algorithm, the simulation results showed that the average time to discover a new switch (i.e. 12 ms) during the topology discovery process is 90% less than the OpenFlow-based solution (i.e. 100 ms). In [85], the authors implemented this topology discovery mechanism in a small-scale testbed.

Likewise, the authors in [86] proposed the SDN Resource Discovery Protocol (SDN-RDP) as an alternative to distribute the management of the network state among several SDN controllers. Each controller discovers a portion of the network topology in order to ensure the distribution of node management and also simplify the protocol resolution. The described mechanism is asynchronous, lacks a global initialization process and does not require previous knowledge of the network. Based on simulation results, the proposed protocol achieves an efficient reduction of the controller overload.

In addition to these proposals, there are other centralized approaches, such as Path Computation Element (PCE). This network entity behaves similarly to an SDN controller. The PCE is capable of computing optimal end-to-end paths for each network switch in real time. These routes could not be calculated without the topology information stored in the PCE.

Choi et al. [87] proposed a topology discovery protocol called Generalized TOPology (G-TOP) for a stateful PCE. This protocol allows the PCE to automatically construct the network topology as a controller without using a distributed routing protocol (e.g. Open Short Path First-Traffic Engineering (OSPF-TE)). The proposed protocol proactively collects the topology information from the switch and reactively updates the topology changes through an out-of-band control channel. After implementing the proposed protocol, the total time for updating the topology was about 10 ms for the testbed system described in the paper. However, the authors used out-of-band management, which might not be possible to deploy in some real, large-scale scenarios.

Although these mechanisms [74, 75, 78–80, 82, 84–86] can discover the network topology and quickly detect any changes in the status of physical connectivity (i.e. link or node failure), they all require previous IP configurations (i.e. layer 3 address) within the network. Hence, if not every forwarding device has been configured with an IP address in the network, the described mechanisms cannot discover the network topology. Consequently, there is still a lack of more flexible topology discovery mechanisms in SDN based on layer 2 techniques. In addition, the mechanisms discussed above cannot provide any procedure or technique for recovering the network. Our research is therefore intended to solve both of these issues.

After discovering the network topology, the next critical task in current SDN deployments is to maintain an accurate, up-to-date view of the network topology without overburdening controller performance. To achieve this, we analyze the current state-of-the-art approaches towards fault management in SDN.

3.2 Fault Management in SDN

In this section we first discuss related works published in the area of fault management in SDN based on improving the standard OpenFlow solution. After, we analyze some research efforts focus on leveraging self-healing frameworks in SDN.

3.2.1 OpenFlow-based Frameworks

Improving the robustness of SDN has been identified as one of the most important tasks to be addressed by SDN research [88]. However, it has thus far been one of the least researched topics.

Capone et al. [89, 90] propose fast and reliable detour planning for failure management in SDN. Their framework relies on OpenState, an OpenFlow extension that enables switches to perform match-actions rules depending on states triggered by packet-level events. In this way, the control logic on SDN controllers related to failure management is in part offloaded onto the forwarding devices. Furthermore, the authors also define optimization models for the computation of backup paths, considering both single link and single node failures. Simulation results show the suitability of this approach compared to a classic end-to-end path protection scheme and with respect to an approach based on the OpenFlow fast-failover mechanism [58].

Enhanced local detouring mechanisms, with flow grouping and aggregation methods for rapid and lightweight failure handling in OpenFlow networks, are also investigated in [91, 92]. Based on the flow grouping strategy, the authors propose the Controller Independent Proactive (CIP) and Controller Dependent Proactive (CDP) recovery schemes. Through the performance evaluation in [91] it was identified that the proposed recovery schemes achieve a 99% reduction in flow storage for alternate path setup using Virtual Local Area Network (VLAN) tagging and reduce the failure recovery time up to 4 ms and 20 ms respectively, satisfying the 50 ms total failure recovery time required in carrier networks.

Similarly, a fast (i.e. sub 50 ms) failover scheme was introduced in [93]. This scheme relies on link-failure detection by combining primary and backup paths configured by a central OpenFlow controller. Moreover, authors implement a per-link failure detection using Bidirectional Forwarding Detection (BFD) [94], a protocol that identifies failures by detecting packet loss in frequent streams of control messages. Performance measurements in a hardware switch OpenFlow-based testbed show that recovery times of sub 50 ms can be achieved by configuring the BFD transmit interval at 15 ms. The faster recovery time of 3.3 ms was obtained after further decreasing the BFD interval to 1 ms. The experimental evaluation confirmed that recovery times achieved are independent of path length and network size.

In [95], the authors study the impact of network failures on the deployment of load balancing mechanisms in intra-datacenter networks based on the OpenFlow protocol. They use an active probing method to detect and manage failures, exploiting the load balancing among equal cost multiple paths. By exploiting this technique, all of Top-of-Rack (ToR) switches can perform local configuration modifications and act independently of the central controller, avoiding the saturation and scalability issues of the SDN controllers.

Although the aforementioned proposals [89–93,95] eliminate the drawbacks of SDN controller interventions (in terms of packages overhead and control latency) by taking actions at the switch level only, these solutions are limited to recovering the system from failures and do not consider performance guarantees after restoring the network.

3.2.2 Self-Healing Frameworks

The widespread adoption of SDN in heterogeneous and failure-prone deployments (i.e. data centers and clouds) has raised a general interest in providing SDN with the self-healing paradigm [14, 53, 55]. In relation to this, some researchers have proposed novel frameworks to improve the resiliency and predictability of SDNs.

Thorat et al. [96] proposed a self-healing SDN framework that optimizes recovery by applying autonomic principles. The proposed framework includes a Rapid Recovery (RR) mechanism on the switch level and an Optimized Self-Healing (OSH) module on the control plane. Rapid recovery is based on link protection schemes and could be implemented through the OpenFlow group table concept. After a failure occurs, the RR mechanism must recover the network connectivity as soon as possible to minimize service disruption time. Then, the OSH module uses the network information to calculate new optimal paths. Based on the analytical model proposed, the authors proved a reduction in backup flow entries after a failure of 99% per switch.

The vulnerabilities of SDN and NFV from a fault management perspective are also analyzed in [97]. The authors proposed a self-healing-based framework to ensure the resiliency and availability of end-to-end services and resources in 5G networks. This framework interacts with the three planes of SDN (i.e. the application, control and data planes) by taking observations from the network and launching recovery actions. The proposed self-healing framework for SDN/NFV-based networks defines two types of actions, namely those that heal the SDN architecture and those that cooperate with the NFVI to avoid any service interruption (i.e. proactive actions). The authors translate part of the self-healing framework into a specific SDN platform by proposing a diagnosis block in the control plane. They proved that this module can detect any unavailability of a multicast service as well as reactively resolve malfunctions at several levels.

In [98, 99], the same authors proposed a generic self-healing approach based on Bayesian Networks models for a diagnosis block. In [98] the authors developed an algorithm into a self-healing module in an in-band centralized SDN architecture. This infrastructure was emulated on Mininet with POX as the SDN controller. To prove the functionality of the proposed module in the presence of failures, the authors ran a video streaming service delivery through the fixed network topology emulated on Mininet. Based on this experiment, the authors claim that the self-healing module can detect, diagnose and repair faults of different natures, such as physical failures, streaming services, OpenFlow crashes and drops on any interface [99]. As a result, if the streaming service behaves abnormally, the module detects this, diagnoses the root cause and applies the corresponding actions to reestablish the service.

In [96–99], functional frameworks to provide the self-healing paradigm in SDN are described. However, we believe that there is still room for integrating the use of self-healing techniques with the topology discovery procedure to leverage the reliability and accuracy of the centralized network view and control plane topology in SDN. Our proposal considers the integration of SDN with autonomic properties in order to provide native topology discovery and fault recovery within the control plane. This is achieved by only taking actions at the switch level, without overburdening the controller, and is suitable for SDN scenarios with in-band control.

3.3 Network Function Virtualization

In this section, we begin by presenting an overview of papers that propose methods to address the VNF allocation problem in SDN/NFV environments. Subsequently, we analyze some recent works related to the SFC requirement in this research area.

3.3.1 Virtual Network Function Allocation

The placement of VNF within physical resources is one of the main technical challenges of NFV, and it has received much attention in recent research. Although comprehensive surveys have already been provided in [69, 70], in this section we identify some of the existing works related to VNF placement.

In [100], Cohen et al. studied the problem of virtual functions placement within a physical network and provided near optimal approximation algorithms. This solution is based on the combination of the facility location problem and the generalized assignment problem. The performance of the solution is evaluated with respect to two measures: the distance cost between the clients and the virtual functions by which they are served, as well as the setup costs of these functions. However, the model used by Cohen et al. does not consider the order of VNF processing.

The consolidation of VNFs (i.e. their strategic deployment for efficient use of physical resources) is also studied in [71]. The authors formulate the Network Function Consolidation (NFC) problem as an ILP model that aims to minimize the total number of deployed VNFs. A heuristic algorithm called Greedy Network Function Consolidation (GNFC) is also proposed to solve this problem in large-scale cases. This algorithm attempts to identify the network reconfiguration scheme with the maximum decrement in the number of VNFs.

Similarly, in [101] the authors proposed a consolidation algorithm with the objective of power consumption minimization as part of the routing and resource dimensioning problem in NFV architectures. Their solution is based on the migration of the VM implementing the VNF instances and is able to achieve power consumption savings by turning off the unused servers. Heuristics are proposed for both cases of off-line and online traffic demand.

Despite the efficiency of these models [71, 101], the authors do not consider the distance between VNFs deployed in the network. In contrast, our proposal jointly considers the consolidation of the requested VNF instances (in order to reduce the number of deployments) in conjunction with the distance between deployed VNFs (to decrease the utilization of network resources, such as link bandwidth). Regarding the use of bin packing approaches for the VNF placement problem, previous solutions have been proposed.

In [102], the authors proposed (for the first time) a VM placement strategy that reduces the cost of both the server and the network. Authors solve the VNF allocation problem by reducing the cost of the network with a higher priority than the cost of servers. However, the authors do not consider a bin packing algorithm suitable for the chained VM placement problem.

3.3.2 Service Function Chaining

Coupled with the VNF placement, the proper interconnection of service functions must be addressed in order to guarantee the flow of packets in the network. To that end, contributions in the field of NFV have introduced the novel paradigm of SFC, which investigates the proper interconnection between VNFs across one or multiple data cloud centers. Despite the nascent development stage of SFC, several related studies have investigated the use of SFC to dynamically deploy VNFs over software-defined and virtualized networks.

Before deploying SFCs in a network, the first challenge is to formalize a request for chaining several VNFs together. For this purpose, the authors in [103] defined a model for formalizing the chaining of network functions using a context-free language. This approach enables network operators to deploy SFC requests and construct VNF graphs that can be mapped to the network while considering the possible dependencies between the services. Furthermore, the authors performed a Pareto set analysis to investigate the possible trade-offs between different optimization objectives (e.g. maximizing data rate on network links and minimizing the number of used nodes or latency of created paths). They proposed a Mixed Integer Linear Programming (MILP) strategy for determining the placement of the network functions, taking into account the different operators' deployment objectives. Although online placements are very important for operators, the authors do not provide a solution to VNF placements in a more dynamic approach.

In [104], Luizelli et al. solved the network function placement and chaining problem using an ILP and a heuristic procedure for large infrastructures. Their model aims to minimize the number of VNF instances mapped on the infrastructure while ensuring that end-to-end latency constraints on mapped SFCs will be met, considering both link transmission delays and VNF processing delays. Similarly, the authors in [105] proposed an algorithm incorporating a Variable Neighborhood Search (VNS) meta-heuristic in order to explore the placement and chaining solution space efficiently. Here, the algorithm minimizes the required resource allocation, while meeting network flow requirements and constraints. Even though these models are efficient [104, 105], they do not consider an online approach to placing the requested SFCs in dynamic environments.

An analysis of network operational costs with VNF placement is presented by [106]. In their work, an ILP model is used to determine the number of VNFs required (and their placement) to optimize network operation costs and resource utilization while adhering to service level agreements. A dynamic programming-based heuristic, which models the SFC orchestration problem with multi-stage graphs and uses a Viterbi algorithm, is used to solve larger instances of the problem.

In [107], the authors define the service path selection as a grey system theory problem and introduce an algorithm that steers traffic among different services using SDN/NFV technologies. In this algorithm there is an SDN controller responsible for calculating service paths and then inserting forwarding rules into the data plane switches to steer flows across the required VNFs. The feasibility and functionality of the proposed framework were demonstrated through implementation in a prototype.

In [108], the authors address the problem of VNF placements in a data center as a multi-layer bin packing problem. They formulate the placement problem as an ILP and provide two greedy algorithms (Multi-layer WorstFit and Multi-layer BestFit). Results of Multi-layer WorstFit reduced bandwidth consumption by 15%, but the number of servers used was increased by 1% compared to the traditional BestFit algorithm. Furthermore, these strategies are only applicable in tree-like topologies and are incompatible with environments without dedicated control networks.

3.4 Open Issues

Although discovering the network topology is an essential service for SDN-managed networks, no official standard defines the topology discovery mechanism in this scenario [43, 74, 75, 84, 85]. Therefore, most of current SDN controllers (e.g. NOX [83], Beacon [109], OpenDaylight [110] and ONOS [111], among others) implement conventional topology discovery mechanisms based on the southbound protocol OpenFlow. However, OpenFlow-based topology discovery mechanisms require previous knowledge of network device characteristics, such as the number of active ports and MAC addresses [43]. This information is requested by SDN controllers after the establishment of an initial control connection with each device. Current SDN controllers cannot implement any conventional topology discovery mechanism without this information. Additionally, existing solutions require a preconfigured network identifier (i.e. IP address) on each forwarding device. Otherwise, SDN controllers will not be able to discover the forwarding devices in the network.

In addition, most of the current research efforts in the fault management area have been oriented towards proposing recovery mechanisms within the data plane of SDNs. However, a resilient control plane is a critical feature for current SDN deployments. This high-level goal is becoming extremely important due to the growing prevalence of SDN on large-scale and heterogeneous networks, for which the in-band mode is more practical and cost-efficient. Moreover, achieving robustness in the control plane should not be limited to recovering the system from failures. It should also ensure proper responsiveness regarding performance guarantees (such as control paths delay) once the network is recovered. To that end, exploiting the self-healing property of an ANM system, agreed upon as the next generation of management [112], represents a very suitable approach. Despite this, in our literature review we identified a lack of proposals that integrate cognitive and autonomic management schemes with SDNs.

Furthermore, it is worth emphasizing that while there have been important efforts regarding topology discovery and fault management in SDN, both features have been mostly treated in isolation. We argue that the research community would benefit from an integrated solution that jointly improves these critical services. Therefore, our study bridges a significant gap by providing an enhanced protocol design that combines both features for a more cognitive and resilient control plane in SDN environments.

After virtualizing the infrastructure through the NFV technology, one of the main technical challenges to be addressed is the location of VNFs on the virtualized network infrastructure. According to ETSI [65], the network functions can be placed either in a centralized fashion or at the service edge close to the end-user. However, it has been demonstrated that both approaches have certain limitations that can be addressed by a chained model [67]. In this context, the SFC approach requires advanced algorithms to efficiently orchestrate the available physical and virtual resources in the VNF placements. Moreover, if online scenarios are considered (i.e. with a set of already deployed SFCs where new incoming SFC requests are processed in a one-by-one modality), the proposed algorithm needs to be dynamic and have low-complexity properties in order to provide effective solutions and quick execution times.

Placing the VNFs into the software-defined and virtualized network infrastructure can be used to target different goals, resulting in different placement solutions. However, network operators need placement strategies that consolidate more VNF instances in order to reduce CAPEX and OPEX in the network. In relation to this, we study online algorithms that aim to achieve a strategic deployment of VNFs that efficiently uses existing physical resources and minimizes the number of links between deployed VNFs. As a consequence, network operators could also use fewer network resources (e.g. data links and bandwidth) to interconnect their supported network applications.

Finally, there is not a standard evaluation model to assess different VNF placement strategies [14]. Here, we have provided evaluation parameters that appraise the impact of network topology and system demands (i.e. requested SFCs) on the placement solutions.

Chapter 4

Network Topology Discovery in SDN

This chapter is based on:

- **L. Ochoa-Aday**, C. Cervelló-Pastor and A. Fernández-Fernández, "Current Trends of Topology Discovery in OpenFlow-based Software Defined Networks," *Scientific Report*, UPCommons, pp. 1–6, Sept. 2015.
- **L. Ochoa-Aday**, C. Cervelló-Pastor and A. Fernández-Fernández, "A Distributed Algorithm for Topology Discovery in Software-Defined Networks," in *Advances in Intelligent Systems and Computing. Trends in Practical Applications of Scalable Multi-Agent Systems, the PAAMS Collection.* Springer, vol. 473, pp. 363–367, June 2016.
- **L. Ochoa-Aday**, C. Cervelló-Pastor and A. Fernández-Fernández, "Discovering the Network Topology: An Efficient Approach for SDN," *Advances in Distributed Computing and Artificial Intelligence Journal (ADCAIJ)*, vol. 5, no. 2, pp. 101–108, Nov. 2016.

This chapter pushes forward the state-of-the-art of conventional topology discovery mechanisms in programmable networks. In this regard, we deliver the design and evaluation of an optimized topology discovery protocol that extends the boundaries of existing approaches (e.g. OpenFlow-based mechanisms) with distributed layer 2 discovery and minimal communication overhead.

4.1 Introduction

An efficient and straightforward mechanism for topology discovery in large-scale SDNs could be achieved by dividing the entire process into phases and distributing the discovery functions hierarchically between the network nodes. This would allow for obtaining the network graph as quickly as possible without incurring scalability issues.

In this chapter, we design a novel topology discovery protocol called Enhanced Topology Discovery Protocol (eTDP). Different from previous work, this network protocol is implemented in each switch through a basic software agent that executes simple decisions. Thus, this approach provides a distributed solution, as the nodes that support the network protocol perform the topology discovery process. In essence, this contribution is designed to obtain an abstract view in large-scale networks. It minimizes both the time and the number of required packets, while simultaneously decreasing the overload in controller performance to reinforce the current topology discovery service in SDN.

The remainder of this chapter is organized as follows. In Section 4.2, we first explain the primary considerations of our approach. Then, in Section 4.3, the proposed protocol is fully described. In Section 4.4, we present the simulations conducted and analyze the achieved results. Finally, in Section 4.5 the main conclusions of this work are outlined.

4.2 Problem Statement

Although SDN provides a flexible architecture by centralizing network intelligence, the controller intervention in those control tasks that only require local switch knowledge is not always ideal. The execution of such tasks (e.g. neighbor discovery) can be delegated to the forwarding devices, which can gather the corresponding information and send it to the controller. In this way, the controller remains responsible for performing those tasks that require a global network view and centralized control.

4.2.1 Programmable Network Infrastructure

The proposed solution can be implemented in a network system following the recent design proposed by the SDN paradigm [35]. The overall system architecture for the proposed solution is shown in Fig. 4.1. This network system embraces network control decoupled from forwarding devices and leverages SDN controllers to provide an abstract view of the entire network.

This proposal can be deployed in a network domain with multiple SDN controllers through the use of a software agent (e.g. eTDP client) running in each network device. As a result, based on the topology information sent by switches each SDN controller discovers and maintains an accurate network view in the topology database. The stored information is critical for the

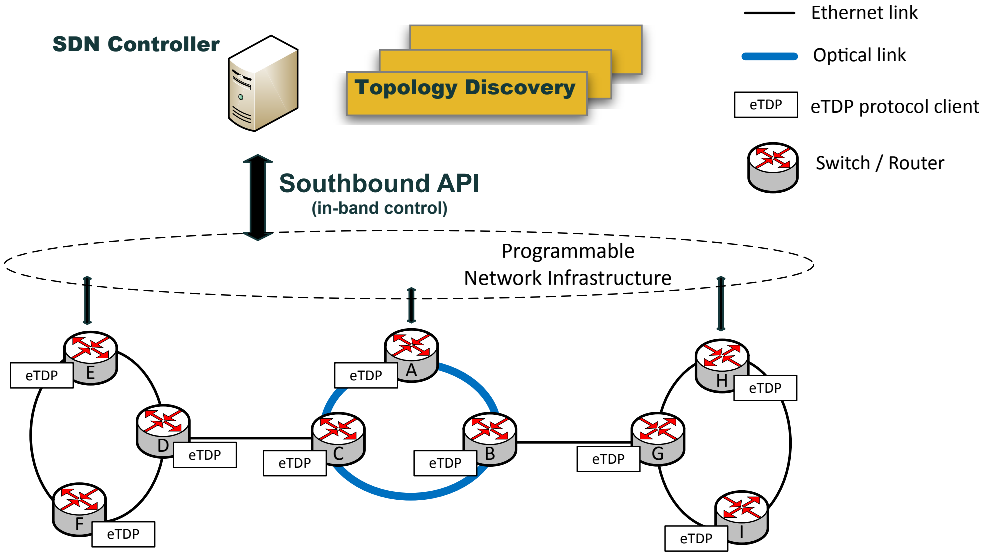


Fig. 4.1: Overall System Architecture for the Proposed Solution

proper operation of other controller services and supported network applications (e.g. traffic engineering, network telemetry and attack detection, among many others).

As shown, the control plane can be interconnected with a plurality of network devices (e.g. traditional routers, virtual network devices, programmable switches, etc.) through different transmission media (e.g. fiber optic links, electrical links, wireless links or logic connections). Although in Fig. 4.1 SDN controllers use an in-band control scheme, this enhanced approach can also be implemented using out-of-band connections.

4.2.2 Forwarding Network Device

The network devices may be any hardware-based (i.e. switch or router) or software-based (logical or virtualized) device configured to perform data forwarding functions according to the routes specified by the SDN controller. Fig. 4.2 presents a schematic diagram of a network device.

The Topology Discovery Protocol Agent is the component responsible for performing the proposed eTDP at each node, which can be implemented using an agent-oriented approach. These agents perform a local partial function of the entire discovery process while interacting autonomously. This capability of distributed operation allows the global topology discovery task to evolve in a scalable and effective way, without overburdening the SDN controller. Topology information retrieved by each node during the eTDP operation is temporarily stored in the device memory and periodically sent to the controller of the corresponding SDN domain.

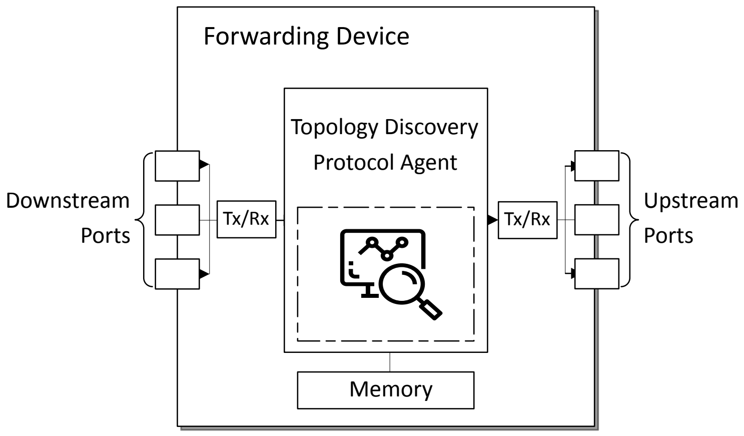


Fig. 4.2: Schematic Diagram of a Forwarding Network Device

4.3 Enhanced Topology Discovery Protocol

The global view of the network is supported by the SDN controllers after running the eTDP. Unlike conventional mechanisms [73, 75], the proposed protocol uses layer 2 messages to create a control tree before the establishment of an initial control connection. Once the control tree is created, each network device can establish a secure control channel with the SDN controllers.

In this section, we begin by describing the data frame formats of each message used by the eTDP mechanism. Afterward, a detailed description of the protocol operations is provided.

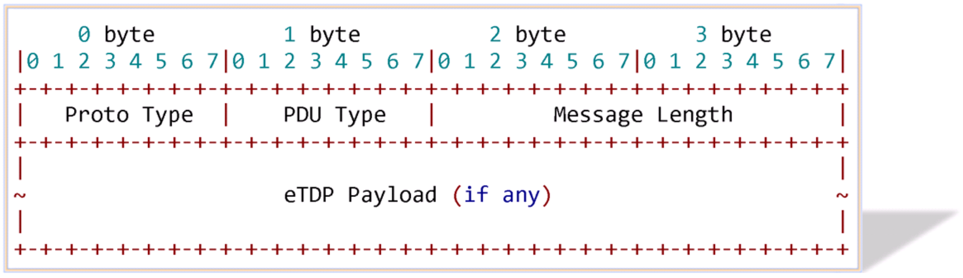
4.3.1 Data Frames Description

The eTDP communications are carried out using a standard network frame format for all data related to the protocol. This feature allows us to develop future extensions of the protocol while maintaining compatibility with previous versions. Moreover, the packets are encapsulated with their corresponding headers (i.e. MAC or IP) for transmissions over the network.

4.3.1.1 Message Header

Every message sent by the eTDP is encapsulated according to the same header structure shown in Fig. 4.3. Note that each tick mark represents a one-bit position in the frames and that the fields are transmitted from left to right.

These messages share a common header format, which allows a node to be able to accept or relay (if applicable) messages of different types. This feature supports fine-grained message forwarding using the powerful "match + action" abstraction of SDNs. The definitions of each



```
     0 byte           1 byte           2 byte           3 byte
|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Proto Type  |   PDU Type    |         Message Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                     eTDP Payload (if any)                     ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 4.3: General Structure of eTDP Messages

field included in the message headers are further described below:

1. *Proto Type*: Protocol type (8 bits). This field uses a specific hexadecimal number to denote the protocol type so that any switch that supports this protocol can easily identify eTDP messages in the network data frame.

2. *Protocol Data Unit (PDU) Type*: Packet data unit type (8 bits). This field contains a value that specifies the type of message in the payload. For example, type 0x01 denotes a topoRequest frame, type 0x02 indicates an echoReply frame and type 0x03 corresponds to a topoReply frame.

3. *Message Length*: Message size (16 bits). This field indicates the message end in the byte stream, starting from the first byte of the header.

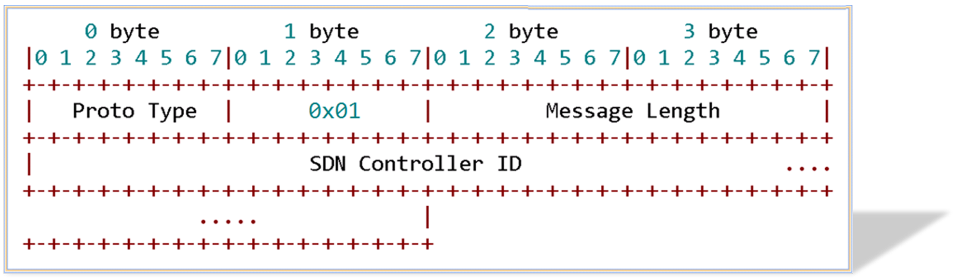
As illustrated in Fig. 4.3, the overall header size is 32 bits (i.e. 4 octets). Some field values (e.g. *PDU Type* and *Message Length*) used in this fixed structure depend on the kind of eTDP messages sent by the network nodes.

4.3.1.2 topoRequest

The topoRequest message is used by the SDN controller to initiate the topology discovery process in the network. Fig. 4.4 presents the message format of a topoRequest.

Besides the header, this simple message only carries the Identifier (ID) corresponding to the SDN controller that sends the topoRequest message. This is the manner in which each SDN controller announces its presence to every forwarding device active in the network.

1. *SDN Controller ID*: Controller identifier (48 bits). The node identifier used in the messages is the MAC address. If the network controller has more than one interface, it must choose



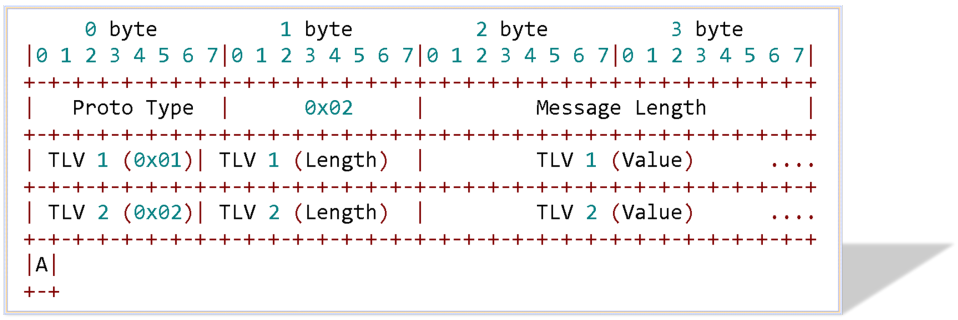
```
     0 byte          1 byte          2 byte          3 byte
|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Proto Type  |      0x01     |         Message Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        SDN Controller ID                  ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
              .....             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 4.4: topoRequest Message Format

the MAC address from one of its active interfaces. This field has the same value for every topoRequest message sent by the SDN controller.

4.3.1.3 echoReply

After receiving the topoRequest message, each network node should automatically reply with an echoReply message. This one-hop reply enables the exchange of local topology information between neighbors and the establishment of a hierarchical control tree rooted in the SDN controllers. In Fig. 4.5 the frame format of an echoReply message is presented. As shown, the value in the message header (i.e. *PDU Type*), has changed to type 0x02 for indicating the echoReply message.



```
     0 byte          1 byte          2 byte          3 byte
|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Proto Type  |      0x02     |         Message Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| TLV 1 (0x01)| TLV 1 (Length)  |         TLV 1 (Value)     ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| TLV 2 (0x02)| TLV 2 (Length)  |         TLV 2 (Value)     ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|A|
+-+
```

Fig. 4.5: echoReply Message Format

This message format was inspired by the use of Type-Length-Value (TLV) structures for the exchange of local neighbor information. Type-Length-Value structures have been widely exploited by several existing standardized protocols such as LLDP [46], Intermediate System to Intermediate System (IS-IS) [113] and Remote Authentication Dial-In User Service (RADIUS) [114], among others.

A TLV structure is a generic representation of an attribute that can be correctly parsed without requiring the parser to understand the attribute. Based on this, we utilized TLV as an efficient method for transmitting different kinds of topology data inside the message body. This encoding offers a reasonable balance between compactness and flexibility, which makes parsing faster and the data smaller. Moreover, using TLV for the data structure makes the proposed protocol extendable. Additionally, TLV elements can be placed in any order inside the message, which provides great flexibility in the design of the protocol.

While the TLV type and length fields occupy the first two octet of the TLV format, the value field may have a fixed or variable size. In addition, it may include different type of information, containing either binary or alpha-numeric data, which is specified using the associated subtype identifiers (e.g. port component, MAC or IP address, interface name, locally assigned identifiers, etc.).

Table 4.1 describes the standard TLVs supported by this protocol. Complementary TLVs can also be defined to enable protocol extensions. Specifically, in the echoReply message, the TLV Node ID and the TLV Node Port ID are included in order to share the node and port identifiers with another directly connected device. The remaining TLV types are used in the message format explained below.

Table 4.1: Summary of TLV Supported by eTDP

Type	**TLV Name**	**Brief Description**
0x01	Node ID	Identifies the node that sends the message
0x02	Node Port ID	Provides the node's port identifier
0x03	Neigh ID	Includes the neighbor's unique identifier
0x04	Neigh Port ID	Provides the neighbor's port identifier
0x05	Link Delay	Carries the Round-Trip-Time (RTT) delay to the neighbor

In addition, the echoReply message is used by forwarding devices as an acknowledgment to confirm or deny the association in the control tree. In essence, each switch in the network sends an echoReply message not only to announce its topology information but to indicate its association with a specific neighbor (i.e. other switch or SDN controller). To do this, an additional association bit is included in this protocol frame. A description of this association bit is given below.

1. *A*: Association Indicator (1 bit). This one-bit field is used by a network device to announce

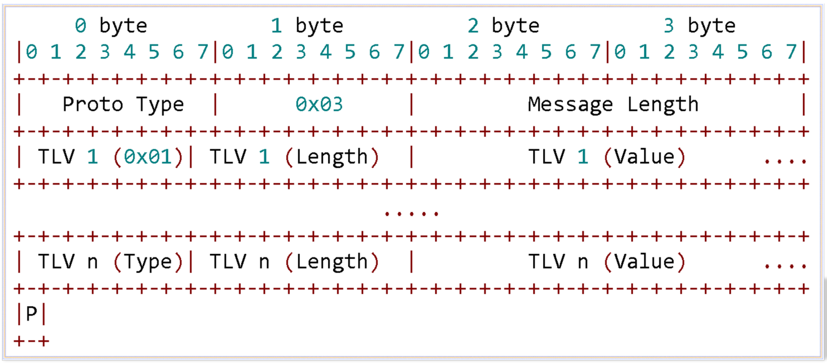
to its neighbors whether it has joined one of them in the control tree. This bit can turn the echoReply into a join message.

As a complement, this message enables eTDP nodes to measure RTT latencies in the network. This feature is crucial to support delay-constrained applications or services executed by the control plane properly.

4.3.1.4 topoReply

The principal function of the topoReply message is to guarantee the proper transmission of the topology network state from the forwarding devices to the SDN controllers. To achieve this, this message format is also based on the use of TLV structures.

Fig. 4.6 shows a brief description of the topoReply message following the basic TLV format. This message may contain the five TLV types supported by the eTDP and presented in Table 4.1. The first of these (i.e. TLV Node ID), which is mandatory for every topoReply message, identifies the node that sends this message, while the others are used to provide information related to the connectivity with the node's neighbors.



```
    0 byte          1 byte          2 byte          3 byte
|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Proto Type  |      0x03     |         Message Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| TLV 1 (0x01)| TLV 1 (Length)  |          TLV 1 (Value)     ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                              .....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| TLV n (Type)| TLV n (Length)  |          TLV n (Value)     ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|P|
+-+
```

Fig. 4.6: topoReply Message Format

Finally, we have also defined a pruning indicator in the topoReply messages. This pruning indicator is used to notify nodes regarding whether any of their neighbors are connected to the network through only them. A description of this pruning bit is given below.

1. *P*: Pruning Indicator (1 bit). This one-bit field enables eTDP nodes to announce to their neighbors whether they cannot provide an alternative path to the SDN controllers.

Unlike the two previous messages, the number of fields in the topoReply format is not fixed and depends on the sender position in the resulting control tree.

4.3.2 Protocol Operation

The presented topology discovery mechanism is initialized by each SDN controller sending a topoRequest message. The propagation of this multicast message creates a control tree topology rooted in the SDN controllers for collecting network state data. Moreover, this control tree also distributes the management of the physical infrastructure among several SDN controllers.

With the exception of the SDN controller, nodes have one of three roles, i.e. leaf, v-leaf or core, according to their position in the network topology. Leaf nodes are the nodes in the network that have only one neighbor. A node is v-leaf when it has more than one neighbor but only one of them can provide a path to the SDN controllers. The remaining nodes are denoted as core nodes.

Additionally, each active port takes one of four states related to the control tree: standby, parent, child or pruned. Fig. 4.7 shows the port states for a given network device.

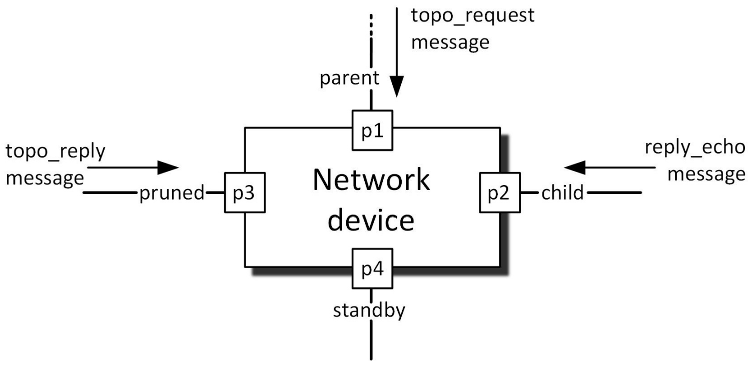


Fig. 4.7: Port States for a Given Network Device

- A standby port is an active port in the node that is not used in the control tree.
- A parent port is an upstream port in the control tree that has first received the topoRequest message. Thus, each node has only one parent port.
- A child port is a downstream port of the control tree that has received an echoReply message with the association bit set.
- A pruned port is a child port that has received a topoReply message indicating that it is connected to a leaf or v-leaf node.

Each port has a state machine that tracks and governs its current state during the control tree creation. This operation is described in the state transition diagram shown in Fig. 4.8.

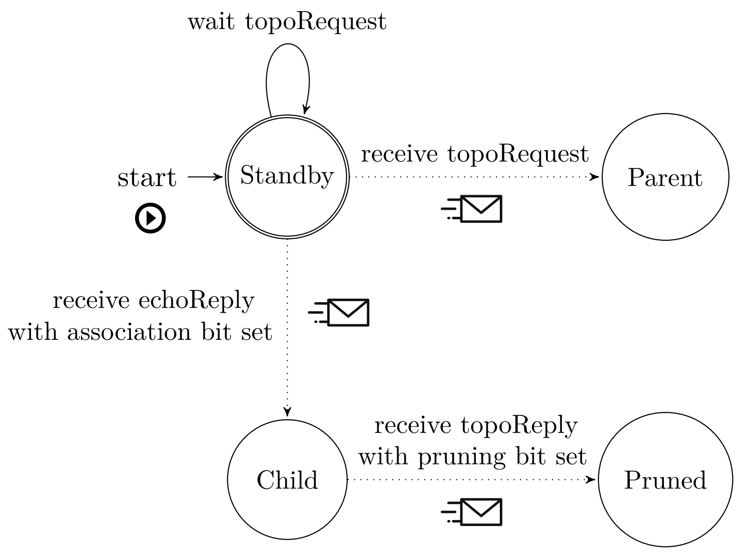


Fig. 4.8: Machine State for Ports in eTDP

The state diagram represented in Fig. 4.8 shows all possible port states drawn as circles. The arcs represent transitions from one state to another and are labeled with the condition that changes the state.

Initially, each node in the network is in a non-discovered mode, with all its ports in the standby state, waiting for a topoRequest message from an SDN controller or another node. After receiving their first topoRequest message, they become discovered nodes through the proposed eTDP. Algorithm 1 shows the forwarding mechanism for a given node v, after receiving the topoRequest message.

Algorithm 1 topoRequest Message Forwarding

```
1: Node $v$ receives *topoRequest* from node $u$ by port $p$
2: **if** node $v$ is *non-discovered* **then**
3:     Send *echoReply* to node $u$                    ▷ association bit set
4:     StateMachine ($p$)                              ▷ $p.state = Parent$
5:     Send *topoRequest* for all ports except $p$
6: **else**
7:     Send *echoReply* to node $u$                    ▷ association bit clear
8:     Discard *topoRequest*
9: **end if**
```

When node v receives the topoRequest from a node u, it sends a one-hop echoReply message to node u. This automatic reply enables the exchange of node and port identifiers between these neighbors as well as the measurement of the RTT in this network link. Moreover, it contains an association bit, which is used by node v as a joining confirmation, to announce to its neighbor node u whether or not they are attached in the control tree.

If when node v receives the topoRequest it is still in the non-discovered state. It confirms the association in the echoReply, sets the incoming port p to the parent state and forwards the topoRequest for all the remaining ports (with the exception of the incoming port). By contrast, if the topoRequest arrives at an already discovered node (i.e. a node that already has a parent port), it denies the association in the echoReply and discards the message. Thus, node v has an implicit mechanism that detects and prevents loops.

As the tree is being created, each node periodically sends its neighborhood data through the parent port using a topoReply message. The leaf nodes asynchronously start this cyclic process after receiving the topoRequest message. In this case, a one-bit field is added to the topoReply message to change neighboring ports to the pruned state. Meanwhile, core nodes aggregate the topology data from their child ports. Once they have received information through all their child ports, they complete their topoReply message and send it to the SDN controllers. As a result, topoReply messages are gathered by the SDN controllers, which receive at most an aggregated message from each of their active interfaces.

Once the network is discovered the SDN controllers use the resulting control paths to instruct leaf nodes about the retransmission period that must be used. This period must be carefully determined, taking into account the type of network to be discovered. In large-scale geographically distributed networks (i.e. networks that can be considered topologically static), the period value may reach maximum values. By contrast, in cloud or virtualized network deployments (i.e. networks that can be considered dynamic) this period should be adequately analyzed due to existing trade-offs between the number of topology messages forwarded across the network and the required accuracy of the network topology. In addition, given the holistic knowledge of the SDN controllers about the network state, the selected retransmission time can be dynamically adapted according to the network occupation, the applications requirements and the controller's load.

Fig. 4.9 presents an example of the eTDP basic operation. For this sample topology, we consider two SDN controllers connected to eight switches through links that have the same unitary delay. This multi-SDN controller platform can be deployed in the cloud using a resilient, federated architecture [102, 115].

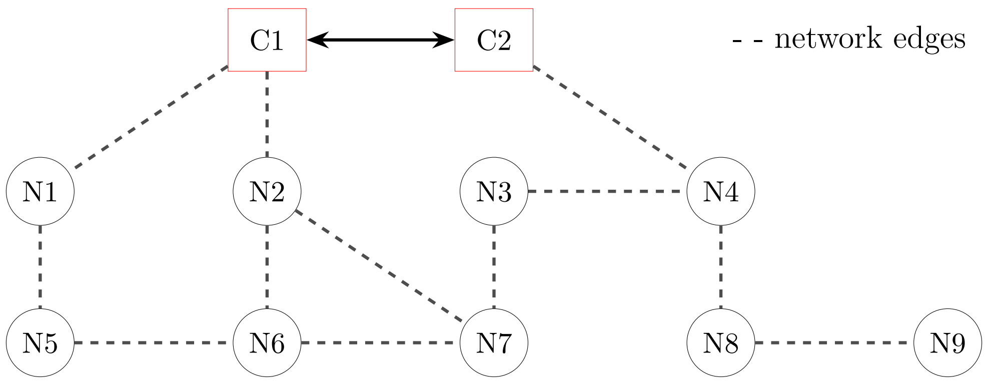


Fig. 4.9: Example Topology with Two Controllers and Eight Switches

After both SDN controllers run the proposed protocol, the control tree topology and port states are generated, as shown in Fig. 4.10. The distribution of switches among SDN controllers is depicted in this figure through the use of colors and shapes. A more detailed explanation of how the protocol aggregates the topology data contained in each topoReply message to the SDN controller C1 is provided below.

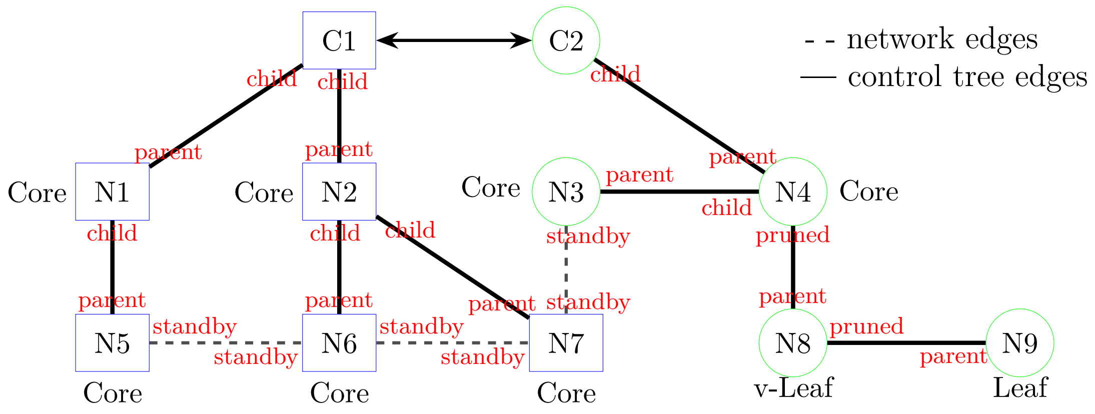


Fig. 4.10: Hierarchical Control Topology of eTDP Solution

It can be seen in Fig. 4.11 the topoReply messages that are sent from downstream nodes (i.e. N5, N6 and N7) to upstream nodes (i.e. N1 and N2) respectively, and then from these upstream nodes to C1. In general, each node will form its topoReply message putting its locally known topology information first, which includes the node and links to neighbors (i.e. involved ports and delays) from which it has previously received an echoReply. Therefore, the information associated with its parent port (i.e. upstream neighbor and the link between them) will not be included since the node does not have received topology information from its upstream neighbor (Neigh ID and Neigh Port ID). However, this information is provided by the upstream neighbor.

The local topology information provided by each eTDP node is organized in the topoReply

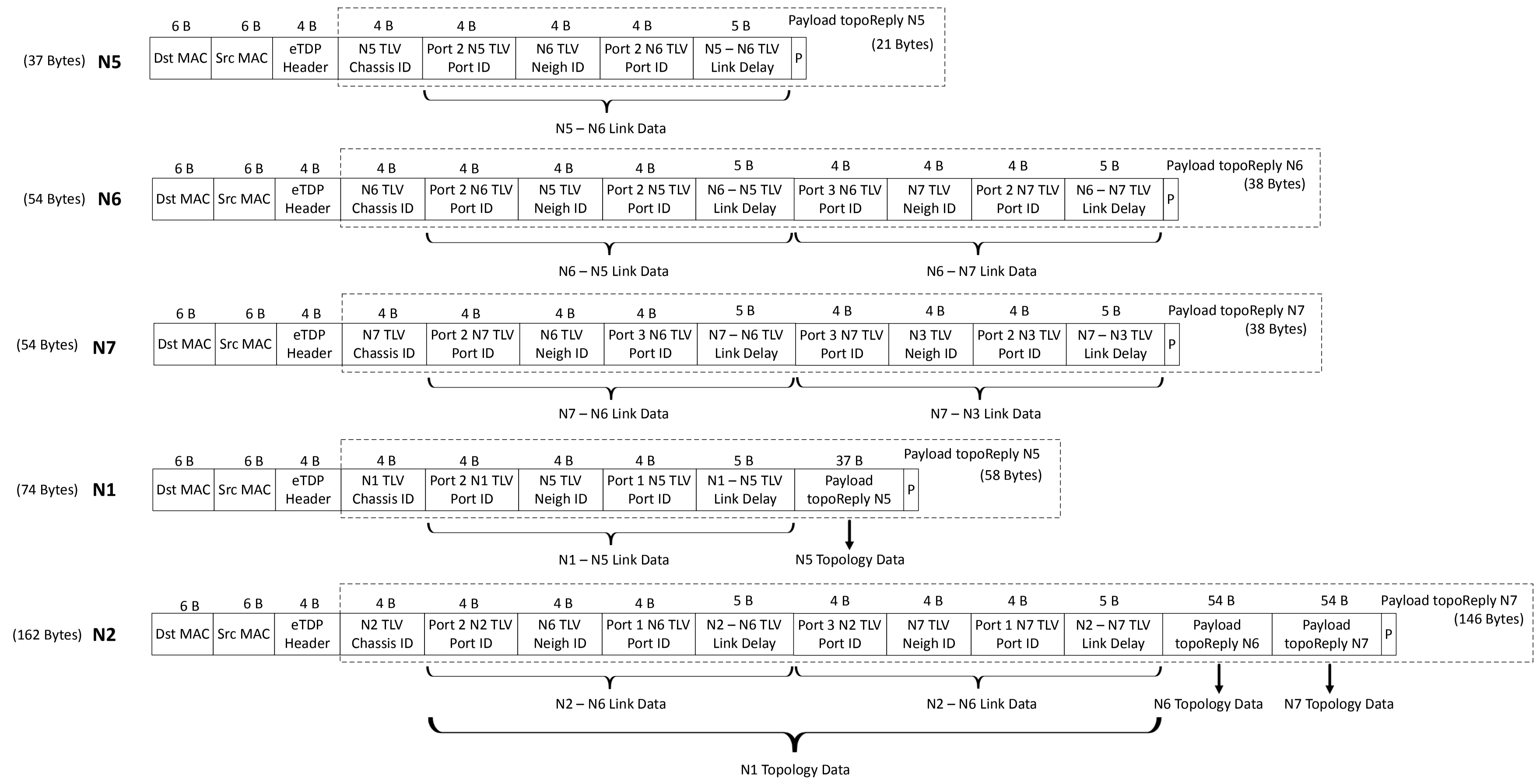

(37 Bytes) N5
6 B 6 B 4 B 4 B 4 B 4 B 4 B 5 B
Dst MAC | Src MAC | eTDP Header | N5 TLV Chassis ID | Port 2 N5 TLV Port ID | N6 TLV Neigh ID | Port 2 N6 TLV Port ID | N5 – N6 TLV Link Delay | P
Payload topoReply N5 (21 Bytes)
N5 – N6 Link Data
(54 Bytes) N6
6 B 6 B 4 B 4 B 4 B 4 B 4 B 5 B 4 B 4 B 4 B 5 B
Dst MAC | Src MAC | eTDP Header | N6 TLV Chassis ID | Port 2 N6 TLV Port ID | N5 TLV Neigh ID | Port 2 N5 TLV Port ID | N6 – N5 TLV Link Delay | Port 3 N6 TLV Port ID | N7 TLV Neigh ID | Port 2 N7 TLV Port ID | N6 – N7 TLV Link Delay | P
Payload topoReply N6 (38 Bytes)
N6 – N5 Link Data
N6 – N7 Link Data
(54 Bytes) N7
6 B 6 B 4 B 4 B 4 B 4 B 4 B 5 B 4 B 4 B 4 B 5 B
Dst MAC | Src MAC | eTDP Header | N7 TLV Chassis ID | Port 2 N7 TLV Port ID | N6 TLV Neigh ID | Port 3 N6 TLV Port ID | N7 – N6 TLV Link Delay | Port 3 N7 TLV Port ID | N3 TLV Neigh ID | Port 2 N3 TLV Port ID | N7 – N3 TLV Link Delay | P
Payload topoReply N7 (38 Bytes)
N7 – N6 Link Data
N7 – N3 Link Data
(74 Bytes) N1
6 B 6 B 4 B 4 B 4 B 4 B 4 B 5 B 37 B
Dst MAC | Src MAC | eTDP Header | N1 TLV Chassis ID | Port 2 N1 TLV Port ID | N5 TLV Neigh ID | Port 1 N5 TLV Port ID | N1 – N5 TLV Link Delay | Payload topoReply N5 | P
Payload topoReply N5 (58 Bytes)
N1 – N5 Link Data
N5 Topology Data
(162 Bytes) N2
6 B 6 B 4 B 4 B 4 B 4 B 4 B 5 B 4 B 4 B 4 B 5 B 54 B 54 B
Dst MAC | Src MAC | eTDP Header | N2 TLV Chassis ID | Port 2 N2 TLV Port ID | N6 TLV Neigh ID | Port 1 N6 TLV Port ID | N2 – N6 TLV Link Delay | Port 3 N2 TLV Port ID | N7 TLV Neigh ID | Port 1 N7 TLV Port ID | N2 – N7 TLV Link Delay | Payload topoReply N6 | Payload topoReply N7 | P
Payload topoReply N7 (146 Bytes)
N2 – N6 Link Data
N2 – N6 Link Data
N6 Topology Data
N7 Topology Data
N1 Topology Data

Fig. 4.11: Operation Example of the Proposed eTDP

message using the following ordered sequence of TLVs: Node ID, Node Port ID, Neigh ID, Neigh Port ID and Link Delay. The last four TLVs are then repeated for every neighbor of the node sending the topoReply message. Then, the node completes its topoReply by aggregating the payloads contained in the topoReply messages received from each of its child ports (if it has such child ports). It should be noted that each additional payload is headed by the use of a new TLV Node ID in the message; a topoReply from a leaf node will only contain the TLV Node ID.

Using the topoReply received from N1 in Fig. 4.11, the controller C1 can discover this part of the network topology and determine the resulting control branch. Specifically, the controller will read the ordered sequence of TLVs contained in this message from left to right. In doing this, it will notice that its neighbor, N1, is also connected by its port 2 to port 1 of node N5. It will also discover the delay between these two nodes them. It can also be determined that N5 is connected to the control tree through N1 since the topoReply from N1 includes the payload of the topoReply from N5 (identified by the use of a new TLV Node ID in the message). Lastly, from this payload the controller will also become aware of the connection between N5 and N6. In addition, as the payload from N6 is not aggregated in the topoReply of N5, the controller will know that N6 is not included in the control branch of which N1 and N5 are a part. Similarly, the topology information concerning N2, N6 and N7 is collected from the topoReply message sent to this controller by N2.

The use of the pruning indicator can also be observed in the topoReply messages shown in Fig. 4.11. These prune bits, all of which are set to zero in the presented example, are sent only one hop back toward the upstream node. Therefore, the upstream node extracts this information from the received topoReply and records it in memory.

4.3.2.1 Protocol Complexity

The complexity of eTDP is defined in terms of the time and number of messages required to collect the topology information at the SDN controllers and create the hierarchical control tree. Regarding time, the protocol complexity is $O(DT)$, where D is the depth of the control tree and T is the maximum edge delay, which is comprised of the link propagation latency, the transmission delay and the switch processing time. Considering the number of nodes N, the number of controllers C, and the highest number of neighbors per node A, the total number of discovery messages propagated across the network is equivalent to $2[AC + (N - C)(A - 1)] + (N - C)$, given that an equal number of topoRequest and echoReply messages are generated and that only one topoReply message is sent by each forwarding device. Therefore, the protocol complexity in terms of messages can be expressed as $O(AN)$.

4.4 Evaluation and Results Discussion

In this section we evaluate the performance of the proposed topology discovery mechanism. To do this, we use essential metrics to assess the operation of the designed protocol in SDN. Furthermore, we use these metrics to compare the control tree formed by the proposed eTDP against other approaches.

4.4.1 Simulation Environment

To provide insightful results over a wide range of simulated but realistic scenarios, we have implemented the eTDP mechanism from scratch in the Objective Modular Network Testbed in C++ (OMNeT++) [116]. Fig. 4.12 shows a sample simulation instance with a European network in the OMNeT++ simulator.

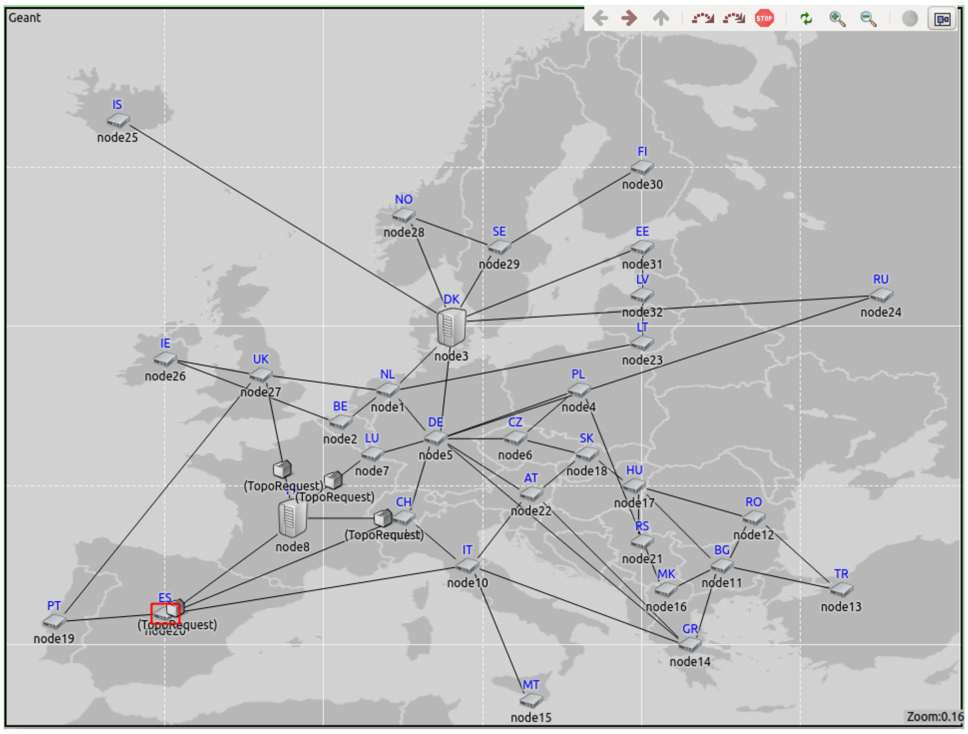


Fig. 4.12: Simulation Environment in OMNET++

Experimental simulations in OMNeT++ have proven to be a reliable approach for supporting studies in large-scale networks [117] that are not hardware dependent and therefore easy to scale and analyze. The role of experimental simulations using the OMNeT++ network simulator is quite significant in the performance evaluation of novel mechanisms in scientific literature [118]. In our study, it enables us to research the behavior and performance of the proposed eTDP in many realistic scenarios of geographically distributed real-world networks.

We generated three sets of networks to evaluate the performance of our solution across varying connectivity degrees. Each network family was generated using an underlying topology from the available online dataset Survivable fixed telecommunication Network Design (SNDlib) [119]. Specifically, we selected three network graphs representative of different scales, namely Atlanta (15 nodes, 22 links), Sun (27 nodes, 51 links) and Pioro (40 nodes, 89 links). Other significant network parameters of these topologies are presented in Table 4.2.

Table 4.2: Network Parameters of the Topologies Used in the Simulations

Topology	Nodes	Links	Average Degree	Diameter (ms)
Atlanta	15	22	2.93	3.1
Sun	27	51	3.78	3.7
Pioro	40	89	4.45	4.2

Topologies that belong to each family set have been constructed as scale-free networks using a power-law node degree distribution with the same degree exponent as that of the original network. This was a result of using the static Barabási-Albert (BA) model [120] and maintaining the original number of nodes and links. Each family size was determined after restricting the margin of error of the indicated average values to less than 6% in each simulation instance. In particular, each topology set is composed of 500 generated networks. All simulation results include their respective 95% Confidence Intervals (CI) in the plots based on Student-t distribution.

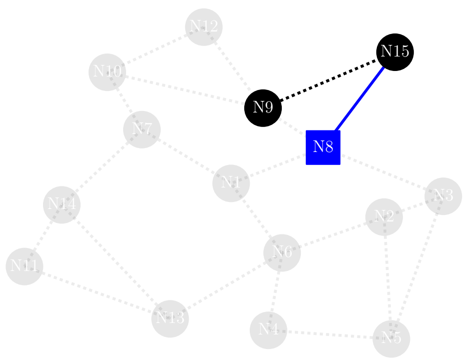
Different link latencies for each network family were randomly generated, considering the mean and standard deviation values of the original topology used as the master. For SDN controller placements, we used the most central nodes in each topology based on closeness centrality. In addition, for computations of the presented time values, we considered the propagation latencies among nodes in the network and the switch processing delays. The later was determined as given in [121] for NetFPGA implementations.

4.4.2 Protocol Performance

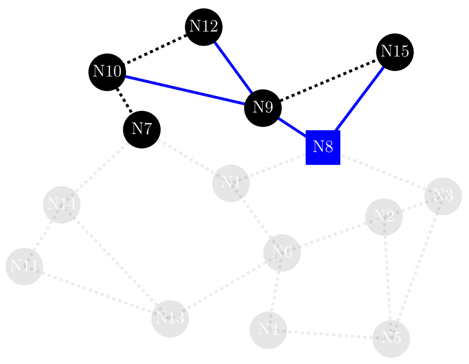
In this subsection we present the performance evaluation of the eTDP solution for different key metrics and analyze the obtained results.

4.4.2.1 eTDP Control Tree Generation

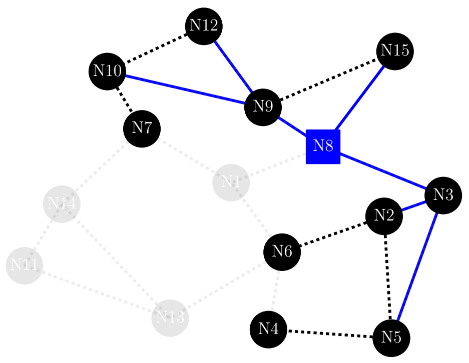
As the eTDP messages propagate across the network, a control tree connecting every forwarding device and rooted in the SDN controller is created. The topology information from each forwarding device is forwarded to the controllers using the generated control tree in order to provide them with a complete network abstract view. To clearly illustrate the obtained knowledge about the network view and the generated control tree, in Fig. 4.13 we present a step-by-step explanation of the information received by the controller in the Atlanta topology with a centralized controller as an example.



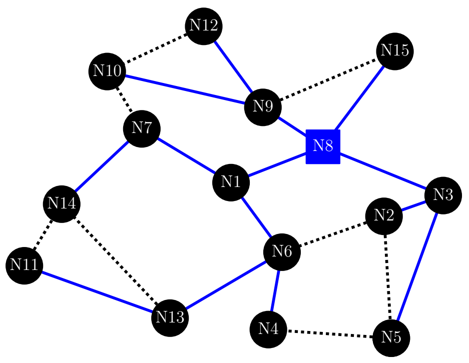
(a) One Control Branch Discovered



(b) Two Control Branches Discovered



(c) Three Control Branches Discovered



(d) Four Control Branches Discovered

Fig. 4.13: Generation of the Control Tree in Atlanta Topology

For this evaluation, we placed the controller in the node denoted as N8 and identified with a unique shape (square) and color (blue). The other switches in the network are depicted as black circles. Regarding the links, solid blue lines represent the control connectivity established by eTDP between the network nodes in the resulting tree. Meanwhile, the remaining network links not included in the control tree are rendered as dotted black lines. It should be noted that partially transparent nodes and links represent undiscovered elements of the original topology in the example.

Fig. 4.13 illustrates the reception order of topoReply messages at the controller. In general, at each step the discovered segments of the entire topology are shown from a holistic point of view in the SDN controller. For instance, Fig. 4.13(a) reveals the constructed network view after receiving the first topoReply with the corresponding topology data. Specifically, this segment corresponds to the information provided by the topoReply message received from the node denoted as N15, where its connection with N9 is also stated. However, the topology information about N9 is not fully discovered until receiving the second topoReply message with information about the network segment shown in Fig. 4.13(b). Finally, after receiving the topoReply messages from the other two neighbors (i.e. N3 and N1) the controller's knowledge of the entire network topology is complete, as depicted in Fig. 4.13(d).

4.4.2.2 Discovery Time

For the simulations, we have defined the eTDP discovery time as the overall amount of time required by SDN controllers to discover the underlying network topology. This metric measures the period elapsed from the moment when the first topoRequest message is sent to the instant when an SDN controller receives the last topoReply.

In Fig. 4.14, we measure the average discovery time required by a number of SDN controllers to discover the overall network topology. The proposed discovery mechanism reveals a significant reduction in the average discovery time as the number of SDN controllers increases from 1 to 5. Moreover, this decreasing behavior remains consistent across the three considered topologies with different connectivity degrees. Specifically, reductions of 57%, 43% and 36% are obtained in Atlanta, Sun and Pioro-based families, respectively. This result is expected given that in our protocol, in order to discover the network the SDN controllers must receive an aggregated topoReply message from each of their interfaces with child state in the control tree (i.e. those

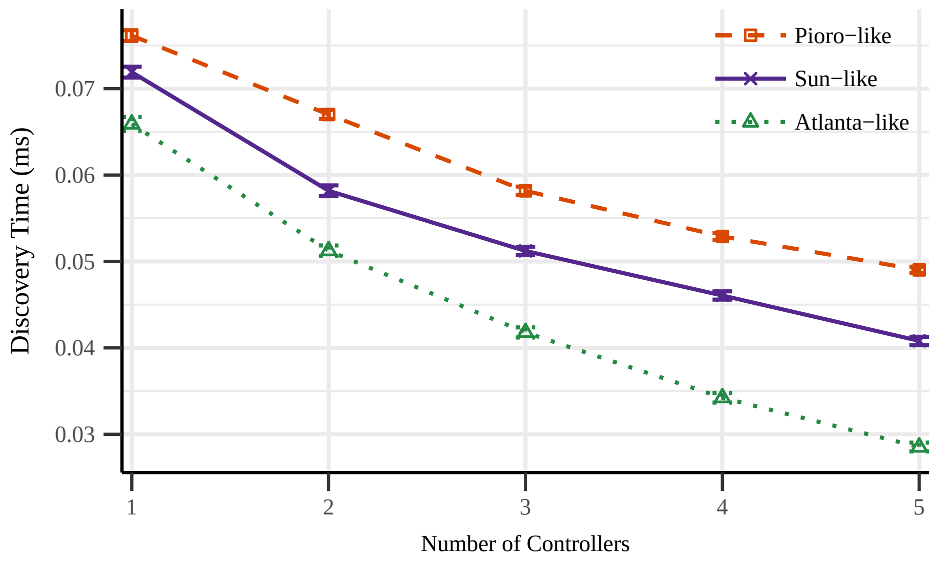


Fig. 4.14: Topology Discovery Time of eTDP

that received an echoReply with the association bit set). Therefore, with the increase of network controllers a smaller number of nodes are associated with each controller (i.e. fewer nodes are included in its control tree), which means that less time is required to obtain the corresponding topology information.

Additionally, in Fig. 4.14 we can corroborate the relationship between the discovery time and the network diameter (in terms of delay). In essence, network topologies with smaller diameters are more likely to require shorter discovery times, since in our approach control trees are formed using the shortest paths from each node to its controller. Therefore, the associated discovery messages will travel, at the most, the length of the network diameter in their propagation across the network.

4.4.2.3 Controller Overhead

Measuring the overhead imposed by the topology discovery service on the network controller is of paramount importance in SDN. In this section, we evaluate the number of topology packets the controller sends or receives under different topology discovery mechanisms.

As previously discussed, the topology discovery mechanism implemented by most OpenFlow controllers is called OFDP [73]. This de-facto standard uses the LLDP frame format to discover links between the OpenFlow switches. The controller load due to OFDP is determined by the number of packet-out and packet-in messages that SDN controllers must process.

Given an OpenFlow-based network of N switches interconnected by L active links, we use p_i to denote the number of ports in a switch $i \in N$. The maximum number of messages sent by the controller for discovering all existing links is defined in Eq. (4.1), while the number of packet-in messages received by the controller is described in Eq. (4.2).

$$\text{OFDP}_{\text{ Packet-out}} = \sum_{i=1}^{N} p_i \tag{4.1}$$

$$\text{OFDP}_{\text{ Packet-in}} = 2 \cdot L \tag{4.2}$$

In order to achieve a reduction in the number of packet-out messages sent in the OpenFlow-based topology discovery mechanism, an improved version, called OFDPv2, is proposed in [75]. In this second approach, the SDN controller only sends one message per OpenFlow switch, as described in Eq. (4.3).

$$\text{OFDPv2}_{\text{ Packet-out}} = N \tag{4.3}$$

The reduction of packet-out messages provided by OFDPv2 can be achieved due to the ability of OpenFlow switches to rewrite packet headers. Similar to the previous approach, OFDPv2 takes advantage of the OpenFlow connection establishment between switches and controllers to collect the required topology information. As a result, this improved version consistently performs better than OFDP regarding the number of messages that the controller is required to process. Moreover, OFDPv2 provides a more suitable approach for networks with a higher number of total ports (i.e. networks with higher average switch port density).

In Fig. 4.15 we present the reduction in the average number of packets managed by the SDN controllers considering two existing approaches as baselines (i.e. OFDP and OFDPv2). This metric is derived from Eq. (4.4), where $NumPkt$ denotes the average number of messages handled (i.e. sent and received) per SDN controller.

$$\text{Packet/Controller Reduction Ratio} = \frac{NumPkt_{baseline} - NumPkt_{eTDP}}{NumPkt_{baseline}} \tag{4.4}$$

As shown, in all cases our approach outperforms the two other topology discovery protocols with significant reductions in the number of packets handled per controller that can be over 50%. In general, eTDP achieves noticeable improvements with respect to both baselines, but as expected, larger reductions are achieved in comparison to the use of OFDP. In contrast to OFDP

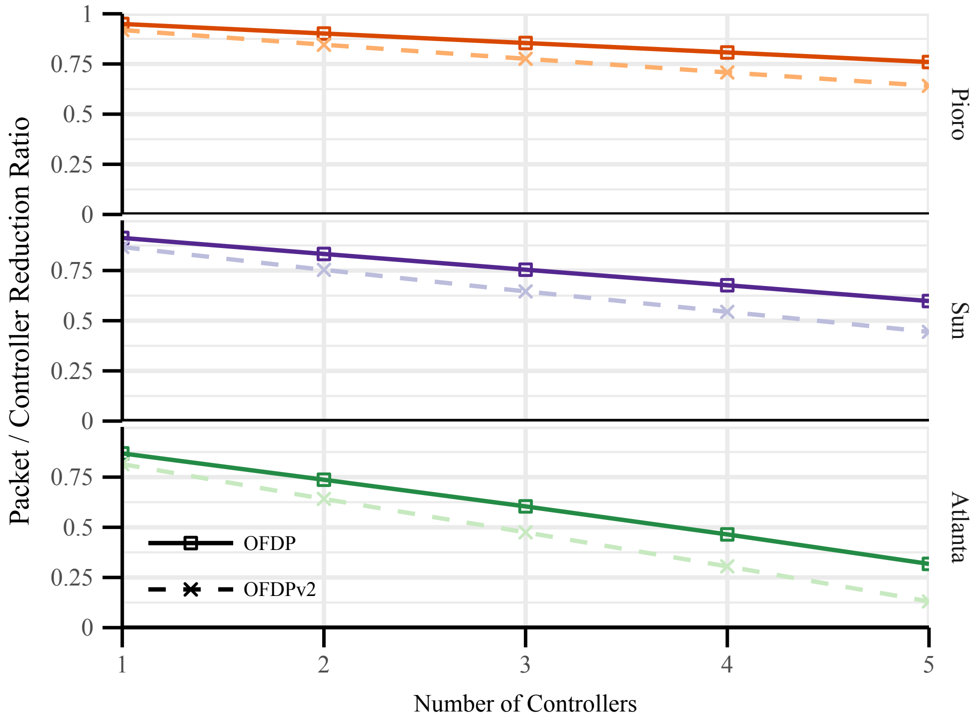


Fig. 4.15: Comparison between eTDP and OpenFlow-based Protocols

and OFDPv2, in eTDP each controller sends only one packet (i.e. a topoRequest message) and receives at most two packets (i.e. one echoReply and one topoReply) from each of its active interfaces. Consequently, our approach allows decreasing the burden on SDN controllers.

Fig. 4.15 also illustrates that the improvements with respect to OFDP and OFDPv2 decrease as the number of controllers increases. This behavior is due to the reduction in the number of switches as a result of increasing the size of the controller set. In this case, fewer packets will be required by the two baselines, while in eTDP this number will remain almost constant or will increase (if the new controllers have higher connectivity degrees).

As there is no study that shows OFDP operation for several SDN controllers, in this analysis we assume the switch distribution among SDN controllers obtained from the proposed eTDP. The average number of switches per controller as determined by eTDP is depicted in Fig. 4.16 for the three considered topology sets.

As expected, the number of switches per controller decreases as the number of controllers grows. Moreover, in the three cases, switches are nearly evenly distributed between controllers at each step along the x-axis. This behavior is the result of the protocol operation in conjunction with the controller's placement assumed in the simulations (i.e. the most central nodes based on the closeness centrality). Under eTDP, switches associate with the controller that is closest to

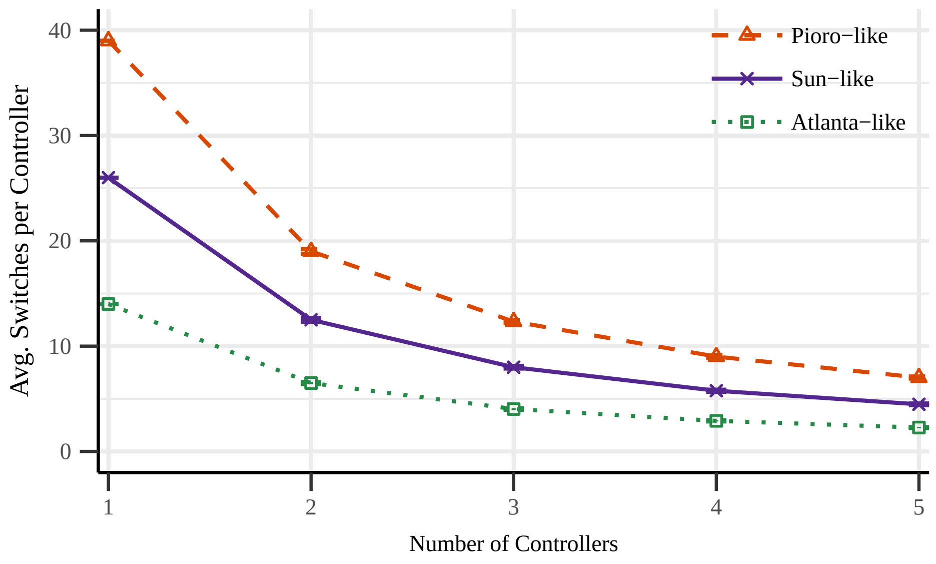


Fig. 4.16: eTDP Distribution of Switches among Controllers

them, forming control trees of shortest paths rooted in the controllers. This reasoning, coupled with the use of controllers placed at the geographic center of the network, results in a balanced load of forwarding devices among the SDN controllers.

4.4.2.4 Discovery Packets per Switch

The average number of packets generated per switch to discover the complete network view is shown in Fig. 4.17. As can be seen, this metric is related to the average network degree.

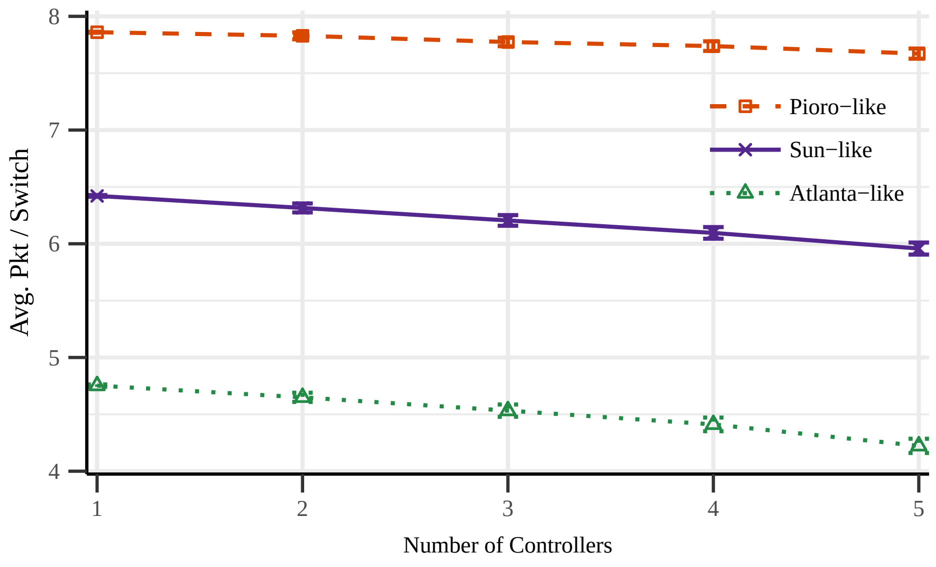


Fig. 4.17: Topology Discovery Packets of eTDP

During the execution of the proposed topology discovery mechanism, eTDP nodes use three main messages (i.e. topoRequest, echoReply and topoReply). However, as each switch always generates only one topoReply message, the average number of packets forwarded in the network is primarily influenced by the topoRequest and echoReply messages. On one hand, the number of topoRequest packets required for discovering the topology corresponds with the number of node neighbors. On the other hand, a switch generates one echoReply per received topoReply. This is equivalent to sending echoReply packets by the parent and standby ports in the forwarding device. As a result, nodes with higher connectivity degrees are likely to generate more topology packets, meaning the resulting average value is therefore increased. The network topologies selected for this study exhibit different connectivity degrees, with the highest values obtained in the Pioro-based topologies.

Furthermore, in all the considered topologies the number of packets handled per switch does not increase in tandem with the number of SDN controllers. Thus, it may be inferred that the eTDP mechanism is scalable in terms of the required number of packets with respect to the number of SDN controllers –a characteristic that is crucial in practical applications.

Table 4.3 presents the average number of packets generated per forwarding device while the number of SDN controllers is increased in the selected network topologies. These values are also classified according to the types of eTDP data frames.

Under eTDP, echoReply messages exchanged in the network are equivalent to the overall number of generated topoRequest packets, since one echoReply must be generated per received topoReply. However, as shown, more echoReply than topoRequest messages are generated by the switches. This is because the number of topoRequest messages sent by the SDN controllers is not included in this table. Therefore, while an equivalent number of topoRequest and echoReply messages are exchanged between switches, an additional echoReply is generated by those forwarding devices directly connected to the controllers. Evidently, this difference increases with the number of controllers and the number of switches connected to them. Additionally, we can also confirm that the number of topoReply messages generated per switch remains constant and equal to 1, irrespective of the network topology.

Table 4.3: Average Number of Generated Pkt/Switch Classified by Type

eTDP	**SDN**	**Network Topology**		
Data Frame	**Controllers**	Atlanta	Sun	Pioro
topoRequest	1	1,77	2,60	3,34
	2	1,62	2,46	3,25
	3	1,51	2,35	3,17
	4	1,44	2,26	3,10
	5	1,38	2,17	3,04
echoReply	1	1,99	2,82	3,52
	2	2,03	2,85	3,58
	3	2,04	2,86	3,61
	4	2,06	2,84	3,63
	5	2,06	2,81	3,64
topoReply	1-5	1,00	1,00	1,00
Total	1	4,75	6,42	7,86
	2	4,65	6,32	7,83
	3	4,55	6,21	7,77
	4	4,51	6,10	7,74
	5	4,43	5,98	7,67

4.4.2.5 eTDP Control Traffic in the Network

The eTDP operation is conceived as a cyclical mechanism to be periodically initiated by the leaf nodes. Once the leaf nodes receive a topoRequest message, as previously explained, they send a topoReply message with their topology data through the parent ports. Leaf nodes should continuously initiate this process in order to maintain an accurate global network view in the SDN controller.

In this final evaluation, we capture the overall traffic received by the SDN controller as a result of the eTDP operation after discovering the network topology. To achieve this, we set the retransmission period to 5 s, similar to the OFDP implementation in current POX controllers [76]. In addition, we used the TLV configuration shown in Table 4.4 to define the length of the required messages.

The different identifiers relating to nodes and ports (i.e. Node ID, Node Port ID, Neigh ID

Table 4.4: Summary of TLV Configuration Used in the Simulations

TLV Name	Total Length (B)	TLV Description
Node ID	4	Header (2B) + Subtype (1B) + Information (1B)
Node Port ID	4	Header (2B) + Subtype (1B) + Information (1B)
Neigh ID	4	Header (2B) + Subtype (1B) + Information (1B)
Neigh Port ID	4	Header (2B) + Subtype (1B) + Information (1B)
Link Delay	5	Header (2B) + Subtype (1B) + Information (2B)

and Neigh Port ID) are represented using alpha-numeric strings of 1 B length. These values are specified in the subtype field as "locally assigned." The value field of the TLV Link Delay is composed of two parts: 1 B of subtype, used in this case to identify the corresponding unit of measure (i.e. s, ms, etc.), and 2 B of information, which contains an integer value between 0 and 65535.

In Fig. 4.18 we present the traffic periodically received by the centralized controller after discovering the network topology with respect to the three original network topologies.

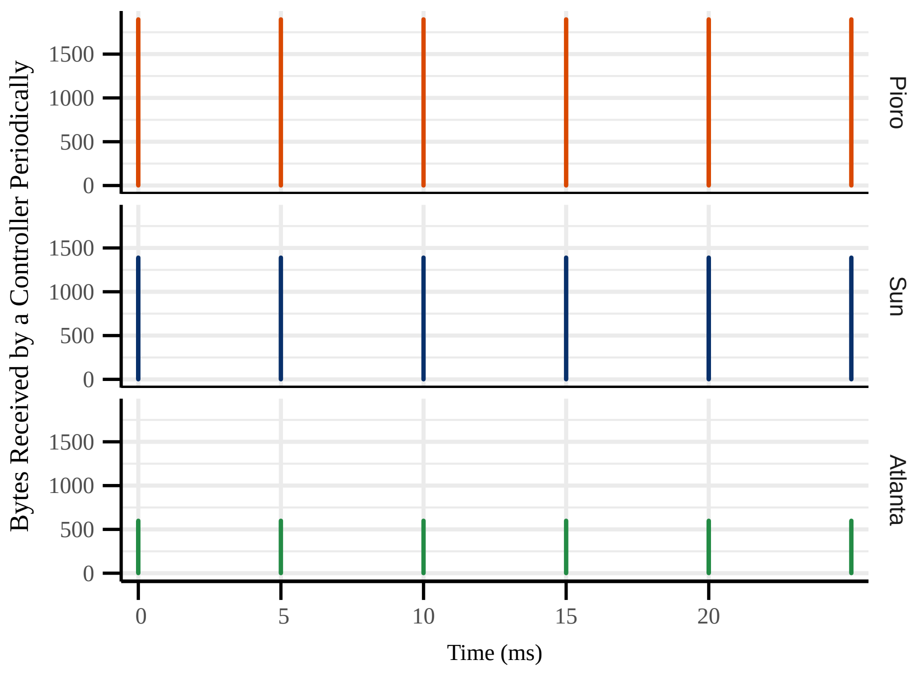


Fig. 4.18: Control Traffic Received at One SDN Controller

The purpose of this evaluation is to analyze and compare the control traffic overhead of the controller across the different topologies. Obviously, the volume of received traffic increases as the size of the network grows (in terms of nodes and links) given that more information must be sent to the controller in order to maintain a complete and accurate network view.

Due to the message aggregation strategy employed in the proposed scheme, the topology information periodically sent to the controller is the result of receiving one topoReply message from each controller neighbor. To better illustrate this, Fig. 4.19 shows the number of bytes received by the controller with the temporal granularity of the plot divided into smaller units.

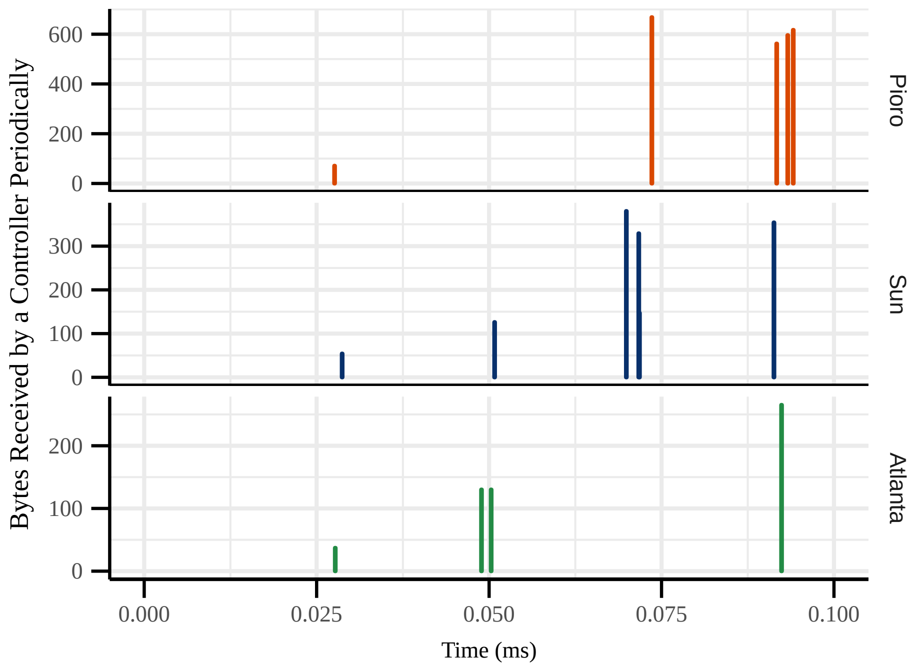


Fig. 4.19: eTDP Traffic Received During One Discovery Period

In this figure it can be seen how several topoReply messages of different sizes successively arrive at the controller. The volume of traffic carried by each message is proportional to the network segment description contained in the data packet unit, and thus to the size of the corresponding branch in the resulting control tree.

4.5 Conclusions

In this chapter we proposed a novel mechanism for discovering layer 2 infrastructures in large-scale SDN topologies. To that end, the proposed eTDP hierarchically distributes the discovery functions among switches supporting this protocol. Unlike existing approaches, this solution enables automatic discovery of the network without requiring previous IP configurations or controller knowledge of the network. By using this mechanism, the SDN controller is able to discover the network topology and construct a holistic network view without incurring scalability issues while taking advantage of the shortest control paths to each switch. Through experimental simulations with real-world topologies, we have demonstrated that eTDP provides a suitable approach for discovering the network topology with discovery times of under 0.08 ms in the three considered networks. The obtained results also show that the overall number of packets generated per switch is not affected by increasing the number of SDN controllers. Moreover, eTDP achieves noticeable improvements with respect to OpenFlow-based approaches, with the most significant reductions seen in comparison to the current OFDP.

Chapter 5

Self-healing and SDN: Bridging the Gap

This chapter is based on:

- **L. Ochoa-Aday**, C. Cervelló-Pastor and A. Fernández-Fernández, "Self-healing Topology Discovery Protocol for Software Defined Networks," *IEEE Communications Letters*, vol. 22, no. 5, pp. 1070–1073, May 2018.

Achieving an efficient topology discovery is only one part of the system necessary to guarantee an up-to-date view of the network infrastructure at all times. The expected solution must be tuned in order to support and perform well in the current dynamic environment in which available network connectivity may change continuously — an aspect for which today's network stacks are poorly optimized.

5.1 Introduction

After discovering the network topology, the control plane needs a survivability strategy to guarantee its reliability at all times. In the last chapter, we proposed an efficient approach to discovering the network topology in SDN-managed networks automatically. However, maintaining accurate knowledge of the network view in real time is a critical issue for proper network operation and one of the most important challenges that must be overcome. Therefore, in this chapter the proposed solution is extended to provide autonomic fault recovery for the control plane.

In this chapter, we shape the design of a novel Self-Healing Protocol (SHP) to boost the control plane resilience in SDN-managed environments. By using the previously proposed eTDP as a basis, both features (i.e. topology discovery and fault recovery) are integrated into an enhanced mechanism. To the best of our knowledge, the presented solution is the first to propose a unified approach for discovering physical topology and providing autonomous fault recovery in the control plane of programmable networks [26].

The remainder of this chapter is organized as follows. In Section 5.2 we define the considered architecture and present the proposed framework. Then, in Section 5.3 we describe in detail the autonomic fault recovery mechanism. The performance of the proposed solution assessed using several evaluation metrics is analyzed in Section 5.4 through experimental simulations in OMNeT++. Finally, we draw some conclusions in Section 5.5.

5.2 Problem Statement

The question of how much control intelligence should remain in SDN switches remains an issue of ongoing debate [43]. Although our solution embraces the idea of centralized network control decoupled from forwarding devices, we envisage an autonomic SDN environment where distributed forwarding devices also contribute to providing services like topology discovery and fault recovery.

5.2.1 Autonomous System Architecture

By definition, autonomic networks are comprised of two major entities: the managed components and the autonomic manager [14]. The overall system architecture for the proposed solution is shown below in Fig. 5.1.

Both elements are identified as follows:

- *Managed Components*: These components are represented by the set of forwarding devices that support the proposed SHP. Each managed component includes sensors for monitoring the state of neighboring links and effectors for modifying local parameters in its network.
- *Autonomic Manager*: This manager is coupled within each SDN controller, has centralized network knowledge and can, therefore, better diagnose problems. This component is responsible for making tactical decisions and optimizing network performance in order to accomplish high-level objectives (e.g. inherent control plane robustness and minimum-latency control paths).

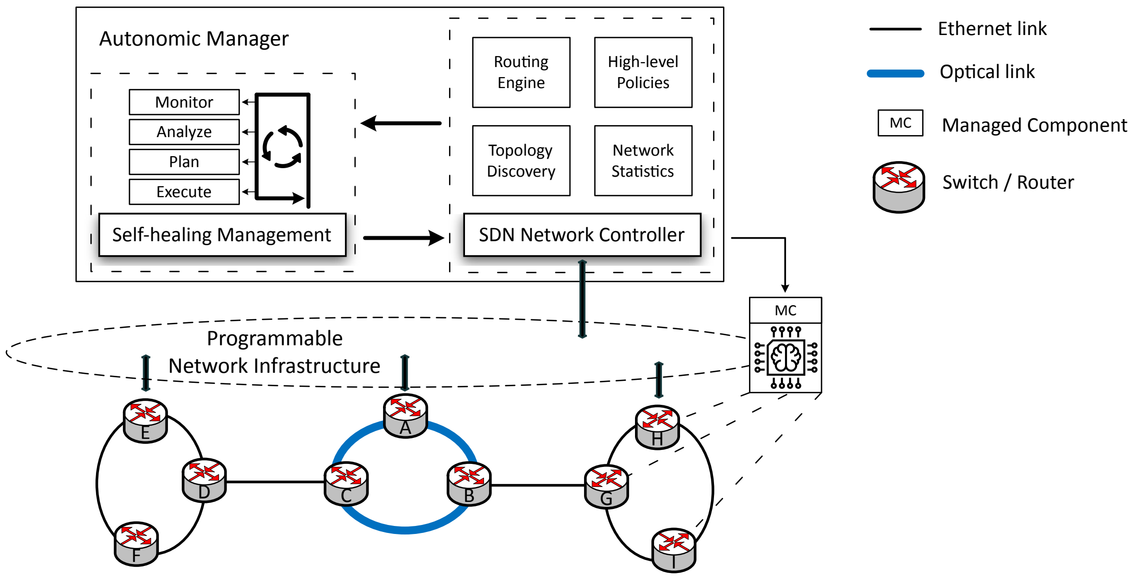


Fig. 5.1: Overall System Architecture for the Proposed Solution

5.2.2 Framework Description

In order to solve the scalability issues of traditional autonomic systems and reduce the time required to recover the control plane in the event of failures, in this approach the managed components also perform some of the MAPE (i.e. Monitor, Analyze, Plan and Execute) functionalities. In essence, each forwarding device is equipped with a Self-healing Protocol Agent composed of several modules as illustrated in Fig. 5.2.

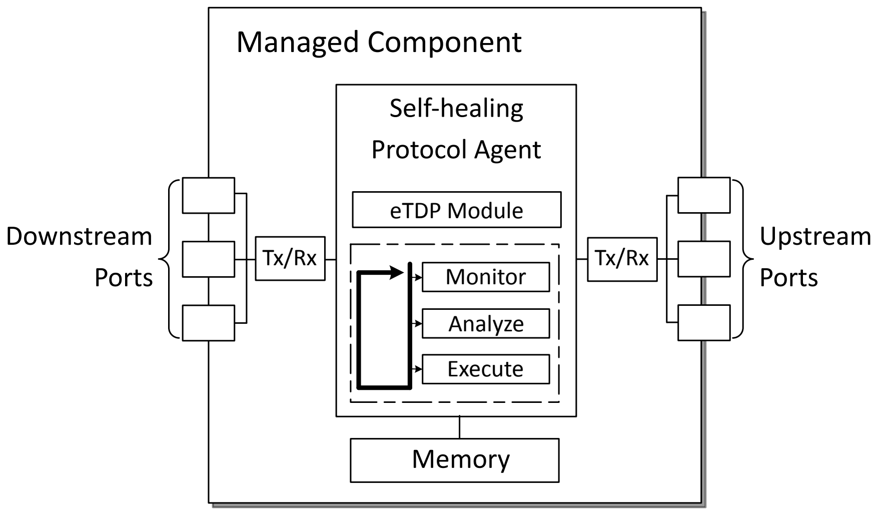


Fig. 5.2: Schematic Diagram of an Autonomous Forwarding Device

The SHP agent allows the forwarding devices to monitor their port interfaces, analyze the collected data and execute the required actions. As a result, forwarding devices are capable of autonomously and quickly resolving link or node failure without the intervention of the controller.

However, as connectivity is recovered from the broken state by only taking local actions, the network can be in a "good" but possibly degraded global state. After the control plane connectivity is recovered and the topology information of the SDN controllers is updated, the control paths can be centrally optimized. To achieve this, the autonomic managers, aware of the entire network view, may change the overlay control topology in order to improve performance (e.g. by finding the control paths with minimum delay).

A summary of the functionalities performed in the proposed solution by both entities (i.e. managed components and autonomic manager) and classified according to the MAPE tasks, is presented in Table 5.1.

Table 5.1: Functionalities Performed by each Entity in the Autonomic Framework

Task	**Managed Components**	**Autonomic Manager**
Monitor	Receive measurement data about the neighboring link state from sensors residing on the devices.	Collects and consolidates the data obtained from its managed components at different locations.
Analyze	Interpret collected data into a state description according to the incoming port statuses.	Isolates the failure from the wider topology and anticipates further implications using the system knowledge.
Plan	Interact with the neighboring nodes through message exchange to discover alternative control paths.	Optimizes the hierarchical control tree to enhance network performance and achieve a set of higher-level goals.
Execute	Perform local control tree adaptations in order to find an immediate solution to the network failure.	Installs new flow configuration rules in the forwarding devices along optimized control routes.

5.3 Autonomic Self-healing Protocol

In order to restore the control plane connectivity and maintain an accurate network view in the controller, the proposed fault recovery mechanism is autonomously performed by the SHP components. In particular, this proposal is conceived to provide a quick control plane restoration by only taking local actions while notifying the controller about the network disruption. The controller can then perform an optimized route computation.

In this section, we first describe the data frame structure of each message used by the SHP components. Afterward, a detailed description of the protocol operations is provided, including the mathematical formulation used in the control path optimization.

5.3.1 Data Frames Description

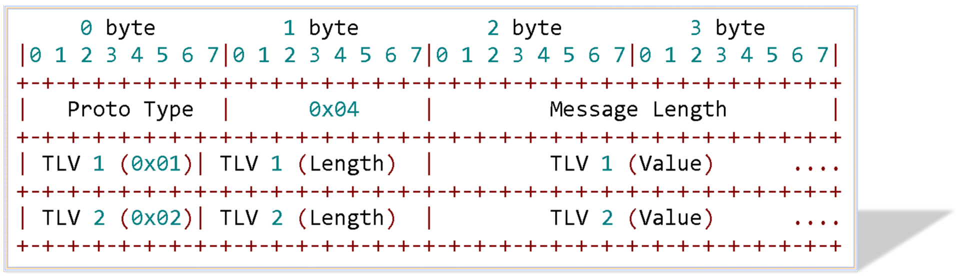
The SHP communications follow the frame encapsulation previously proposed for the eTDP. Accordingly, data frames defined in this proposal use the same header format described in Section 4.3.1. However, new PDU types for two new messages must be defined, as shown in 5.2. Both messages are explained in the following subsections.

Table 5.2: New PDU Types Supported by SHP

PDU Type	Message Name
0x04	topoUpdate
0x05	replyUpdate

5.3.1.1 topoUpdate

The topoUpdate message is used by the forwarding devices to announce that a network failure is affecting the connectivity established in the control plane. In Fig. 5.3 we provide the topoUpdate message format. Besides the header, this message carries two TLVs in the payload, namely TLV Node ID and TLV Node Port ID, previously defined in Table 4.1. The former is used to identify the node detecting the failure, while the latter specifies the involved port.



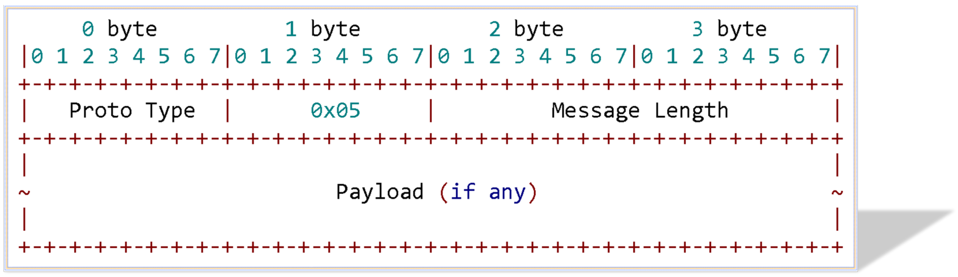
```
     0 byte          1 byte          2 byte          3 byte
|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Proto Type  |      0x04     |         Message Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| TLV 1 (0x01)| TLV 1 (Length)  |         TLV 1 (Value)      ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| TLV 2 (0x02)| TLV 2 (Length)  |         TLV 2 (Value)      ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 5.3: topoUpdate Message Format

This recovery message is first sent when a node detects a network failure in its parent port. This message is then forwarded through child ports to the downstream nodes along the compromised control branches. In addition, this message is also sent through the standby ports of each affected node as possible alternatives to recover the control path to an active SDN controller in the network. In this way, forwarding devices announce the network failure and simultaneously try to recover the control path toward an active SDN controller. By contrast, affected leaf and v-leaf nodes do not receive such a message, since no alternate path can be identified through them.

5.3.1.2 replyUpdate

When a forwarding device that has an active parent port receives a topoUpdate, rather than forwarding it, the node discards this message and responds with a replyUpdate. The data frame format of a replyUpdate is shown in Fig. 5.4.



```
    0 byte          1 byte          2 byte          3 byte
|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Proto Type  |      0x05     |         Message Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                       Payload (if any)                        ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 5.4: replyUpdate Message Format

This message is critical in the process of recovering the broken control plane connectivity since its functionality is twofold. The presence of a payload in the replyUpdate message is optional, and its inclusion depends on the function performed by the particular instance of the message. If the payload is required, the information contained within it is either directly encapsulated by the node detecting the failure or taken from a previously received topoUpdate (i.e. TLV Node ID and TLV Node Port ID).

First, this message is used to provide affected nodes with alternate control paths, enabling the reestablishment of the control connectivity in the hierarchical control tree. For this purpose, the replyUpdate message only carries its header information, so as to perform quick restoration and reduce the communication overhead. In particular, non-affected nodes, which become aware of the network failure after receiving a topoUpdate from a neighbor, send a short replyUpdate message to advertise themselves as possible points of recovery for the hierarchical control tree. In the same way, affected nodes also forward the first replyUpdate they receive across the disconnected branches, with the exception of their pruned ports. Similar to the previous message, affected leaf and v-leaf nodes do not receive this message because they are not able to provide a different route to reach the SDN controllers. In this way, forwarding additional messages to nodes that cannot be used to recover the control tree topology is avoided.

The second task performed by the replyUpdate message is related to the notification of the network failure to the controllers. In this regard, nodes with their control path active that receive a topoUpdate from a neighbor also generate a second replyUpdate, which is sent through their parent port to the corresponding SDN controllers. The replyUpdate payload is reserved for this function, since in this case the information identifying the network failure (received in the topoUpdate) is included as part of the message. Therefore, the remainder of the nodes receiving this extended replyUpdate (i.e. those upstream nodes along the control path) also forward this message to the controller.

5.3.2 Mechanism Operation

The forwarding devices initiate the proposed autonomic mechanism through the SHP. When a network device detects a port failure (i.e. when a neighbor's connectivity fails), the managed component executes specific actions depending on the state of its disrupted port.

In addition to the recovery messages previously presented (i.e. topoUpdate and replyUpdate), a new port state is defined in SHP, called "recovering." This temporal port state identifies a forwarding port of an affected node that is connected to some disrupted network element (node or link). In particular, a disconnected node assigns the recovering state to those ports that are either part of the affected control branch or are receiving a topoUpdate from another affected neighbor.

Failures detected on standby, pruned or child ports are automatically reported to SDN controllers with no changes to the upstream control tree. To do this, the notification of the failure is sent through the established control branch to the corresponding SDN controller using an extended replyUpdate message. The failure is specified in this message, as are the respective identifiers of the node and port detecting the fault.

However, if the network failure is detected through a parent port, the affected node must autonomously recover its control plane connectivity by making local decisions with no SDN controller intervention. First, the node informs its neighbors about the failure and forwards a topoUpdate message with its own Node ID and involved Node Port ID through all the remaining ports except those that are pruned. Given their topological nature, leaf and v-leaf nodes cannot provide an alternative control path to the SDN controllers. Hence, unnecessary topoUpdate and replyUpdate messages are not forwarded to them in the control tree. This feature is critical to achieving minimal communication overhead in the proposed SHP. The remainder of the process after receiving a topoUpdate message is described in Algorithm 2.

Algorithm 2 topoUpdate Message Forwarding

```
 1: Node $v$ receives a $topoUpdate$ from node $u$ by port $p$
 2: if $p.state = Parent$ or node $v$ does not have $Parent$ port then
 3:     Set $Recovering$ state to port $p$ and $Child$ ports
 4:     if same $topoUpdate$ had not been previously received then
 5:         Forward $topoUpdate$ for all ports except $p$ or $Pruned$ ports
 6:     else
 7:         Discard $topoUpdate$ from node $u$                ▷ to avoid propagation loops
 8:     end if
 9: else
10:     Send $replyUpdate$ to node $u$ by port $p$                  ▷ without payload
11:     if same $topoUpdate$ had not been previously received then
12:         Send $replyUpdate$ for $Parent$ port toward the controller    ▷ with payload
13:     end if
14:     Discard $topoUpdate$ from node $u$                    ▷ to avoid propagation loops
15: end if
```

In essence, nodes receiving a topoUpdate message from their parent ports (or those that already have their parent ports disconnected), set the incoming port p, as well as all their child ports, to the recovering state (line 3). In this way, nodes identify themselves as affected (i.e. in case the incoming port is the parent port) and mark the ports connected to neighbors that also require an alternate path to controllers. In addition, they propagate the received topoUpdate through all their ports, except the pruned ones, in order to notify their neighbors about the failure and identify a new path to the SDN controllers (line 5). The same sequence of actions is also performed by the node that initially detected the failure from its parent port.

Nodes receiving a topoUpdate while their control paths are active (lines 9 to 15) discard this packet and answer it sending a short replyUpdate to the affected neighbor. Next, these nodes notify the controller about the failure by sending an extended replyUpdate with the node and port identifiers received in the topoUpdate as a payload. It is important to note that, although a node may receive the same topoUpdate several times from different neighbors, the controller is informed only once about each particular failure.

As the announcement of the network failure is performed through the topoUpdate forwarding process, alternative control paths are advertised using the replyUpdate message. Algorithm 3 describes the steps after receiving a replyUpdate.

Algorithm 3 replyUpdate Message Forwarding

```
 1: Node $v$ receives a $replyUpdate$ from node $u$ by port $p$
 2: if node $v$ does not have $Parent$ port then
 3:     Set $p.state = Parent$          ▷ control plane connection of node $v$ is recovered
 4:     Forward $replyUpdate$ for all the $Recovering$ ports
 5:     Set $Standby$ state to all the $Recovering$ ports
 6:     Send a $topoReply$ to node $u$ by port $p$          ▷ with updated topology information
 7: else if received $replyUpdate$ carries the failure identifiers then
 8:     Forward $replyUpdate$ for the parent port toward the controller          ▷ with payload
 9: else
10:     Discard $replyUpdate$ from node $u$          ▷ to avoid propagation loops
11: end if
```

Once a disconnected node receives a replyUpdate, the neighbor sending this message becomes its point of recovery. This means that in order to provide a quick recovery strategy, each affected node will join with the neighbor from which it first receives the notification of an alternate control route (i.e. a short replyUpdate). Thus, the incoming port p is set to the parent state, indicating that node v has recovered its connection to the controller through this port (line 3). Then, the received replyUpdate is forwarded by all the ports in the recovering state to notify other affected neighbors about this new possibility of reaching the controller (line 4). Afterward, these recovering ports are changed to the standby state (line 5).

Furthermore, a topoReply message is sent by the affected node to its point of recovery in order to share its topology information and confirm this new association (line 6). Therefore, under the SHP, receiving a topoReply from a standby port after sending this port a replyUpdate is equivalent to receiving an echoReply with its association bit set. Accordingly, the neighbor node, acting as a point of recovery, changes the port status from standby to child and collects this topology information to be sent to the SDN controller as stated by the eTDP module.

When a node receives a replyUpdate with the network failure specified in the payload (i.e. for fault notification purposes), the incoming message is forwarded through the parent port to the controller (lines 7 and 8). Finally, although several replyUpdate messages can be sent to disconnected nodes from different neighbors, only the first is selected, meaning the replyUpdate messages received after the node is recovered are discarded (line 10).

Taking the SDN topology with two controllers and eight switches discussed in the previous chapter as an example, we can describe the basic operation of SHP after a node failure occurs in the network. Specifically, in Fig. 5.5 we redraw the considered control tree topology, illustrating a sample disruption of the core node N2. In the explanation of this example, we are assuming that the closest active nodes to N6 and N7 are N5 and N3, respectively.

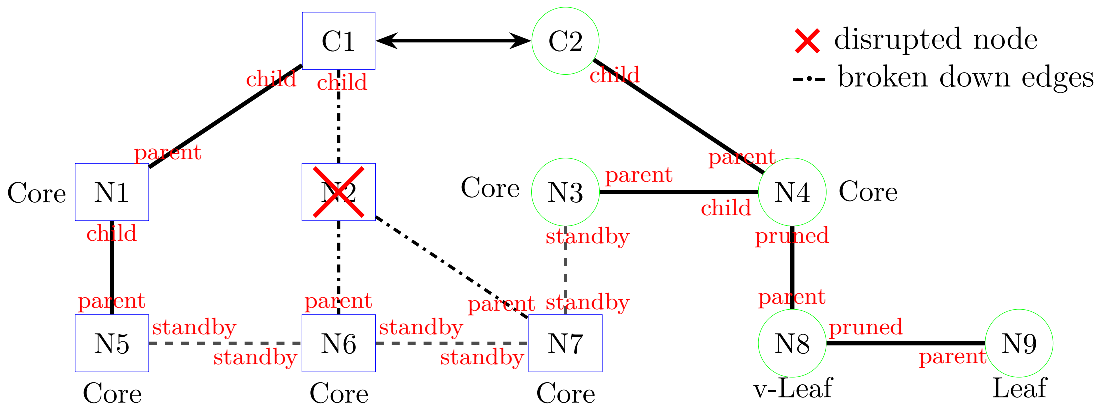


Fig. 5.5: Sample Control Tree Topology with Node Failure

As shown in Fig. 5.5, when N2 fails, N6 and N7, which are connected to the disrupted node through their parent ports, lose their paths to the SDN controllers. Hence, both nodes send a topoUpdate message for all their active ports in order to announce the network failure and identify new control paths to the SDN controllers. Thus, two topoUpdate messages are propagated between neighbors, indicating the disrupted port of N6 in one and the disrupted port of N7 in the other.

When N6 receives the topoUpdate generated by N7, it changes the state of the incoming port of this message from standby to the recovering state. The same change is triggered in N7 after receiving the topoUpdate corresponding to N6. Once N5 and N3 become aware of the network disruption, they respond to the received topoUpdate packets, sending back two short replyUpdate messages to N6 and N7.

In addition, each of these points of recovery (i.e. N5 and N3) notifies its controller about the network disruption using replyUpdate messages. In this instance the replyUpdate messages are extended with the received TLVs specifying the failures.

After receiving the first replyUpdate, the two affected nodes assign the incoming ports to the parent state and forward these reply messages using their recovering ports. After doing this, ports in the temporary recovering state of both nodes are reset to the standby state. Additionally, these nodes send a topoReply to their respective points of recovery. This topoReply message contains updated topology information, which is also forwarded to the SDN controllers following the recovered control paths. To avoid propagation loops, replyUpdate messages exchanged between the considered nodes (i.e. N6 and N7) and received after the control plane connection is recovered are discarded. The recovered control tree that results after the completion of this procedure is illustrated in Fig. 5.6.

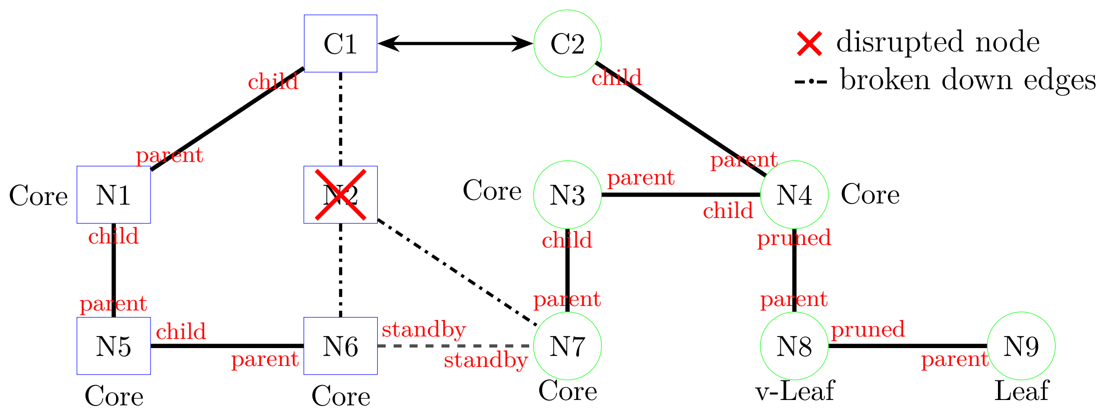


Fig. 5.6: Control Topology Recovered by SHP Operation

To more clearly illustrate this process, Fig. 5.7 shows the message sequence for the proposed fault recovery mechanism when N6 detects that its parent port is nonfunctional. For the sake of simplicity, the exchange of recovery messages between N6 and N7 is not included in this figure.

As explained above, N6 sends one topoUpdate message containing its Node ID and the Node Port ID of the port connected to N2, which has failed, to N5 and N7 simultaneously. This topoUpdate message is also forwarded from N7 to N3. Instead of forwarding the topoUpdate, nodes that have their parent port active (i.e. N5 and N3) respond to this request by sending

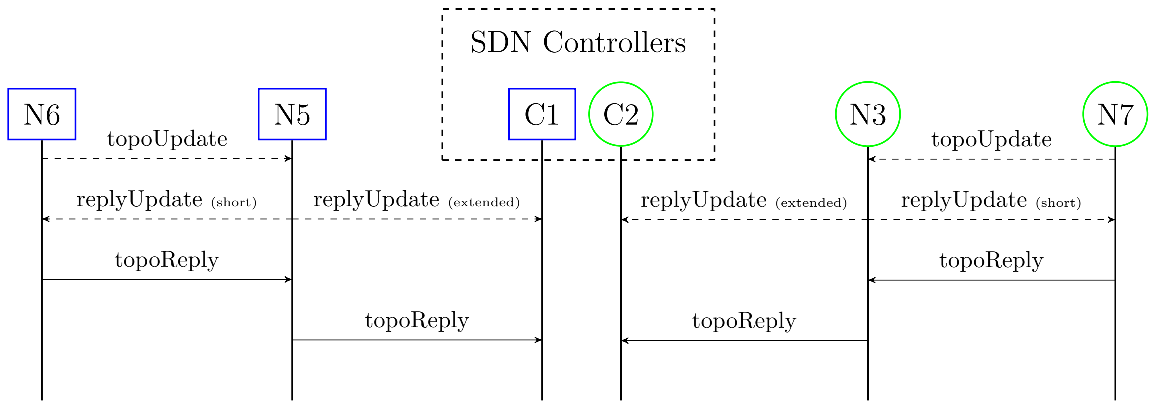


Fig. 5.7: Message Flows for the Autonomic Fault Recovery Mechanism

back a short replyUpdate message. This replyUpdate is sent through the path followed by the received topoUpdate message, creating a new way for N6 to reach the SDN controllers. At the same time, the received failure identifiers are sent by N5 to C1, using an extended replyUpdate.

Upon receiving the first replyUpdate sent by N5, N6 changes the state of the incoming port to parent and automatically sends an updated topoReply message to N5. Afterward, any subsequent replyUpdate messages will be discarded (e.g. the one coming from N3 and forwarded by N7). The updated topoReply message is also used by N6 to announce to N5 that they are now joined in the recovered control tree topology. Hence, N5 should update its port status as well as inform its SDN controller (i.e. C1) of the updated topology data.

5.3.2.1 Centralized Optimization

Once the control plane connectivity is recovered the resulting tree can be optimized by the SDN controllers since they have complete knowledge of the network topology and state. In this regard, several optimization criteria may be considered in order to meet the requirements of the supported network applications and high-level objectives. In particular, due to the separation of network control from the forwarding devices, it is critical to establish control paths with minimum delay. To that end, minimizing propagation latency in control paths is fundamental to being able to respond to events in real time, and this has become one of the most significant design metrics for large-scale SDN.

The computation for minimizing propagation latency can be modeled using an optimal ILP. Designed to run as a network application on the SDN controller, this simple model computes a loop-free topology with optimal-delay control paths based on the network information previously collected by the controller. The goal is to identify the tree with the lowest path-delay between each node and the controller. Consequently, the topology information of the network could be sent to the controller using the shortest control paths (in terms of delay), allowing the topology data and statistic information of the forwarding plane to reach the controller with the shortest path-delay possible.

To describe the considered SDN, we model the network as a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. Each link (i, j) has its own associated non-negative delay $d_{i,j}$. We denote C as the controller location in a specific node of the network graph and U as the set of forwarding devices (i.e. $U = V \setminus C$). The goal of this model is to find the subset of control paths (P_C) that form a minimum-delay tree rooted in the SDN controller. To do this, the decision variable for the ILP model is defined as follows:

$p^u_{i,j}$: describes the selection of an edge (i, j) in the control path from a node $u \in U$ to the controller.

$$p^u_{i,j} = \begin{cases} 1, & \text{if edge } (i,j) \text{ is selected in the path,} \\ 0, & \text{otherwise.} \end{cases}$$

Using this notation, the objective function can be defined as follows:

$$\text{minimize} \qquad \sum_{u \in U} \sum_{(i,j) \in E} p^u_{i,j} \cdot d_{i,j} \tag{5.1}$$

subject to:

$$\sum_{j \in N | (i,j) \in E} p^u_{i,j} - \sum_{j \in N | (j,i) \in E} p^u_{j,i} = \begin{cases} 1 & \text{if } i = u, \\ -1 & \text{if } i = C, \\ 0 & \text{otherwise,} \end{cases}$$
$$\forall i \in N, \forall u \in U \tag{5.2}$$

Eq. (5.1) minimizes the delay in all control paths from each node to the controller. This objective function ensures an optimal delay spanning tree with the shortest control paths between each node and the SDN controller. Flow conservation constraints in Eq. (5.2) require that the control path of each node $u \in U$ is formed by the sequence of links at which $p^u_{i,j} = 1$. The overall control path delay is the summation of the corresponding selected link delays. Based on this simple formulation we are able to find the optimal delay set of control paths from each node to the controller.

5.4 Evaluation and Results Discussion

In this section we first describe the simulation setup used to evaluate the proposed recovery mechanism. Then, we discuss the different tests conducted and results achieved in order to analyze the impact of SHP on several network parameters.

5.4.1 Simulation Environment

Similar to the previous chapter, we used the hlDiscrete Event Simulator OMNeT++ [116] to implement the proposed solution because of its suitability for studying realistic large-scale scenarios and because of the lack of suitable tools for researching SDN from a layer 2 perspective [117,118].

Likewise, for the conducted simulations we continued to work with Atlanta (15 nodes, 22 links), Sun (27 nodes, 51 links) and Pioro (40 nodes, 89 links) from SNDlib [119] as network graphs representative of different scales (see Table 4.2). In order to evaluate the performance of our solution across varying connectivity degrees, we also used the three family sets generated using these networks as seed, taken from Subsection 4.4.1. For the sake of brevity, a short description of the parameters used in the generation of the family sets is summarized in Table 5.3.

Table 5.3: Parameters Used in the Generation of the Network Sets

Parameter	Description
Nodes Degree	Obtained using the BA model with the original number of nodes and links.
Family Size	Composed of 500 generated networks.
Links Latency	Randomly generated using the mean and standard deviation from the original topology.
SDN Controllers	Placed on the most central nodes based on the closeness centrality.

As previously explained, topologies that belong to each family set were constructed as scale-free networks using a power-law node degree distribution with the same degree exponent as the original network. In addition, for the computation of the presented time values, we considered the propagation latencies among nodes in the network and the switch processing delays. The latter was determined as given in [121] for NetFPGA implementations.

5.4.1.1 BFD Mechanism Design

An essential element that was necessary to determine in our simulations in order to accurately assess the performance of the proposed fault recovery mechanism was the required latency to detect a link failure. Given that Ethernet was not designed with high requirements for failure detection, traditional techniques such as Loss of Signal (LOS) or layer 2 heartbeats cannot meet the 50 ms requirement of carrier-grade networks [122]. Therefore, in our simulations we have used a protocol-agnostic mechanism called BFD [94], for failure detection.

The goal of BFD is to provide low-overhead, short-duration detection of failures in links or paths between two end-point systems. This mechanism operates on top of any data protocol (e.g. network layer, link layer, tunnels, etc.) and is always executed in a unicast, point-to-point mode [94]. We have configured BFD sessions to detect link failures (between neighboring forwarding devices in SDN) within the required 50 ms.

In Eq. (5.3) we derived the (worst-case) failure detection time T_{detec} using the BFD method.

$$T_{detec} = (M + 1) \cdot T_{interv} \tag{5.3}$$

As shown, in Eq. (5.3) the failure detection time of BFD strongly depends on the transmit interval T_{interv} (i.e. the periodicity of the control messages) and the detection time multiplier M. This parameter identifies when a session end-point is considered unreachable in terms of lost control packets. For the simulations, we utilized a multiplier of $M = 3$ to prevent small packet loss from triggering false positives.

Moreover, we focused on detecting link loss instead of path failures. Thus, only one BFD session was set per switch interface. This approach not only reduces detection time significantly but also decreases message complexity and overhead in the network. In Eq. (5.4) we derived the minimal transmit interval T_{min_interv} by implementing a BFD scheme that detects link losses instead of path failures, as previously explained. Link monitoring exhibits great improvement compared to per-path monitoring in terms of failure detection times. This method was also adopted by [93].

$$T_{min\_interv} = 1.25 \cdot \beta \cdot T_{RTT} \tag{5.4}$$

The transmit interval time is lower bounded by the RTT of a link in the network. On highly loaded links, this RTT measure can fluctuate greatly and might result in false positives [123]. Therefore, the retransmission interval of lost packets can be computed using $\beta \cdot T_{RTT}$, where β is the variation of inter-arrival times. For our simulations, we selected a fixed and conservative value of $\beta = 2$, as identified in [124]. In addition, we validated the detection times measured in our experimental simulations using the analytical model presented above.

5.4.2 Protocol Performance

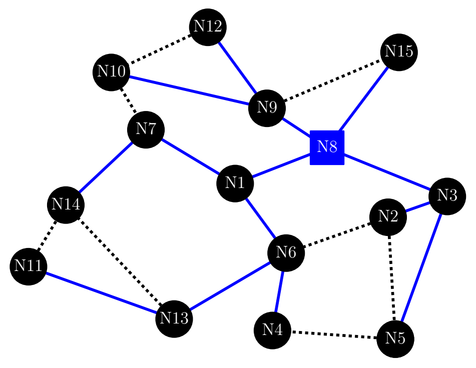
In this subsection, we present the performance evaluation of the SHP solution for different key metrics and analyze the obtained results. Specifically, we assess the proposed SHP considering various metrics such as recovery time, number of generated packets and percentage of nodes involved in the recovery process.

5.4.2.1 SHP Control Tree Recovering

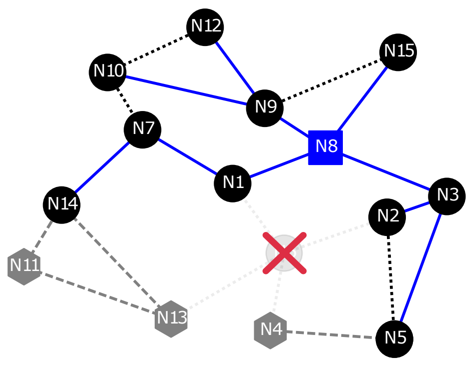
After a failure occurs in the network, the proposed mechanism attempts to recover the connectivity of the hierarchical control tree. To more clearly illustrate the operation of SHP in the event of failures, we begin this evaluation section by presenting a basic recovering example using the Atlanta topology with a centralized controller (see Fig. 5.8).

For this evaluation we placed the controller in the node denoted as N8, identified with a blue square. In Fig. 5.8(a) we first redrew the hierarchical control tree generated by eTDP in this scenario, depicting the forwarding devices in the network as black circles. We used solid blue lines to represent the control paths established between the switches and the SDN controller in the resulting tree. The remaining network links, not included in the control tree, are drawn using dotted black lines.

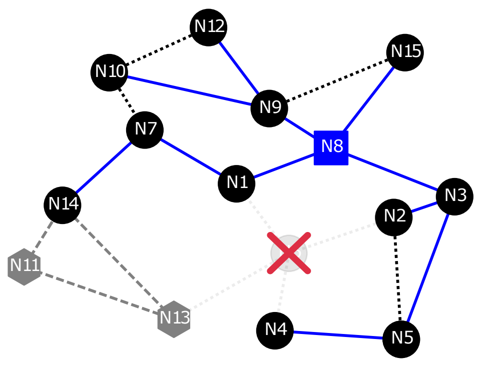
Next, in Fig. 5.8(b) we modified the previous graph to illustrate the occurrence of a node failure. Explicitly, the node denoted as N6 and its links are represented as partially transparent to identify this sample disruption. Additionally, nodes that lose their control connection due to the failure (i.e. nodes N4, N11 and N13), are depicted in a different shape (hexagon) and



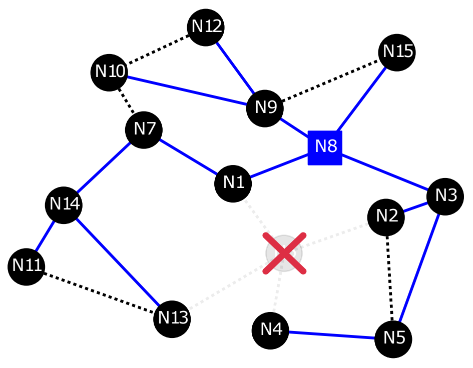
(a) Control Tree Topology



(b) Broken Control Tree Topology



(c) Recovered Control Branch for N4



(d) Recovered Control Branches for N11 and N13

Fig. 5.8: Recovering of the Control Tree in Atlanta Topology

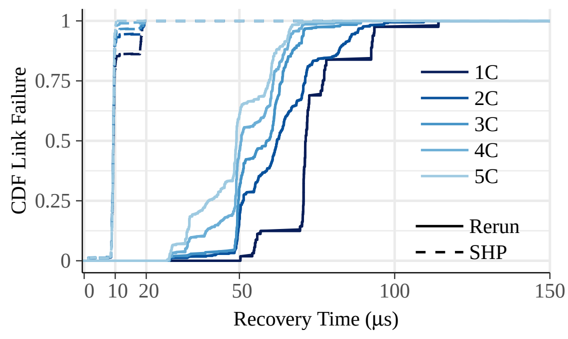
color (gray). Concerning the edges, the set of candidate links that can reestablish the affected control paths are identified using dashed gray lines. The exchange of topoUpdate and short replyUpdate messages performed by SHP occurs over these links.

Fig. 5.8(c) and Fig. 5.8(d) depict the recovering solution adopted for each of the two disconnected branches of the original control tree. In Fig. 5.8(c), we can see that the recovered control path of node N4 now goes through N5, which in this case was the only neighbor that sent a short replyUpdate to N4 with the notification of an alternate control route. In Fig. 5.8(d), however, we see that both nodes N11 and N13 received the first replyUpdate message from the same point of recovery, namely N14. Therefore, two blue lines are drawn between these nodes to indicate the establishment of these new control paths. Meanwhile, the interfaces connecting the nodes N11 and N13 are now in the standby state.

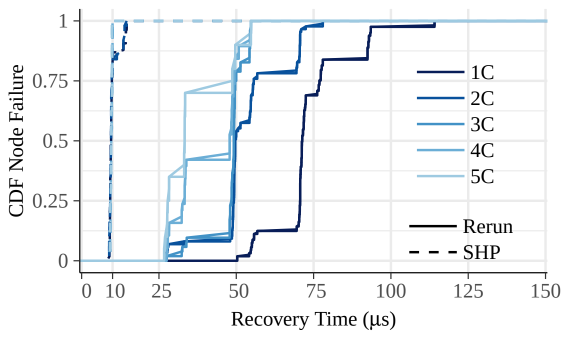
5.4.2.2 Recovery Time

To assess the performance of the proposed SHP mechanism, we start by analyzing the recovery time. We have defined the SHP recovery time as the overall amount of time required to recover a disrupted control path. In more detail, this metric measures the period elapsed from the moment the failure occurs until the establishment of the new association between the disconnected node and one of its neighbors in the recovered control tree. Therefore, the recovery time is composed of the detection latency (obtained using the BFD strategy) and the time needed to complete the required SHP message exchange (i.e. from the sending of the first topoUpdate with the failure announcement to the reception of the topoReply message with the updated topology information by the node acting as point of recovery).

In Fig. 5.9, Fig. 5.10 and Fig. 5.11 we analyze single failure assumption for both links and nodes in the three selected topologies.

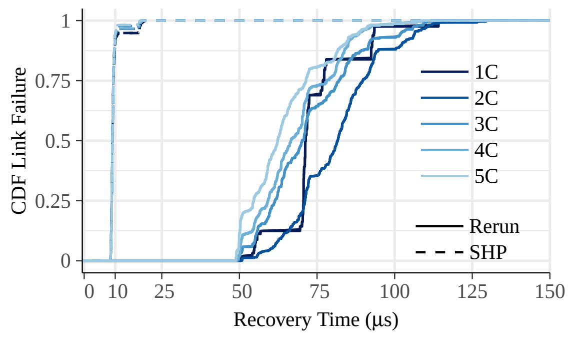


(a) Single Link Failure

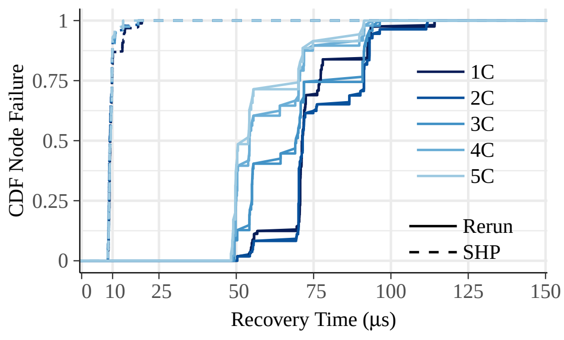


(b) Single Node Failure

Fig. 5.9: Fault Recovery Time in Atlanta

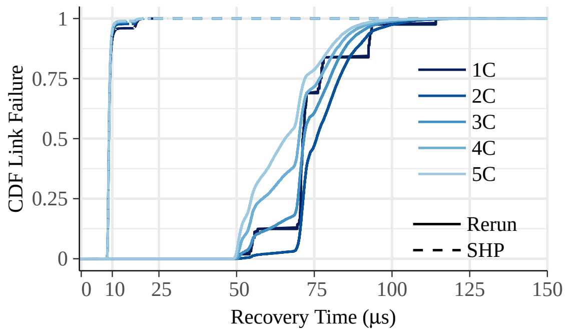


(a) Single Link Failure

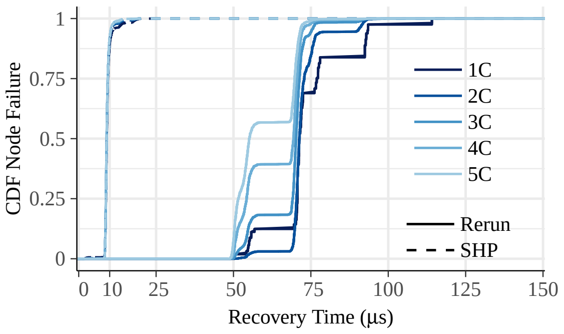


(b) Single Node Failure

Fig. 5.10: Fault Recovery Time in Sun



(a) Single Link Failure



(b) Single Node Failure

Fig. 5.11: Fault Recovery Time in Pioro

Note that a node failure corresponds to the occurrence of a multi-link failure. For this more complex scenario, the reported latencies reveal the entire period required to recover each of the individual link failures that comprise the network event.

In addition, in our simulations we only consider the failure of links and nodes that do not affect the network connectivity, meaning the resulting graph remains strongly connected. In this way, we ensure that recovered control paths can always be established after the occurrence of the network failure.

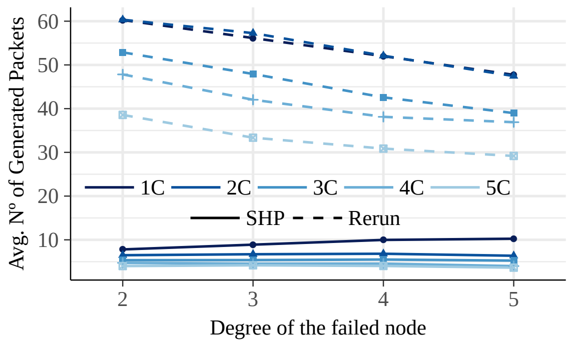
To get a better sense of the achieved recovery times we also include a Rerun approach in this analysis. This baseline approach refers to applying the previously explained eTDP mechanism after the SDN controllers are spontaneously notified of the network failure from the nodes that detect it. In this case, the recovery time was computed by considering the detection time, the time required to inform the controllers about the failure (using the corresponding shortest paths) and the eTDP discovery time.

As expected, in all cases the recovery mechanism outperforms the Rerun approach in terms of required time to reestablish the connectivity of the hierarchical control tree. Specifically, for all the generated topologies, fault recovery times are always below 20 μs for both considered cases (i.e. link and node failures). Therefore, the suitability of the recovery mechanism for application in carrier-grade networks, which require less than the 50 ms, is confirmed. In addition, this behavior is not influenced by the increase of SDN controllers, validating the good scalability of this proposal.

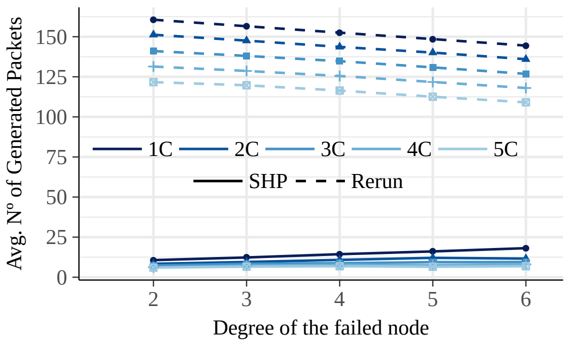
5.4.2.3 Recovery Packets Overhead

Next, we evaluate the impact of the proposed recovery strategy in terms of generated packets for various sizes of multi-link failures. In this analysis, we restricted the scope of the multiple failures to links connected to the same node because simultaneous wider-scope link failures are probably not realistic. In other words, we assume that simultaneous failures of multiple links are due to a failure of a node with a given connectivity degree. It should be noted that the failure of leaf and v-leaf nodes (nodes with a single way of reaching the controllers) are not included in this analysis since their control paths cannot be recovered.

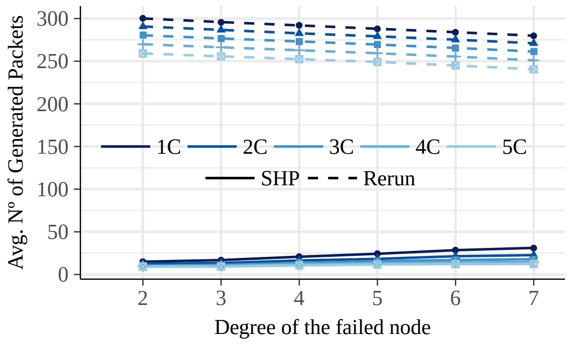
Fig. 5.12 shows the average number of generated packets in comparison to the baseline Rerun strategy for the three considered topologies and varying the number of controllers. As previously mentioned, the degree of the failed node indicates the number of affected links. The number of packets reported in the plots represents the overall average of generated messages considering the failure of each node with a given connectivity degree for the 500 instances of a network.



(a) Atlanta



(b) Sun



(c) Pioro

Fig. 5.12: Average Number of Generated Packets for Recovering Multi-link Failures

From the results it can be seen that under SHP, smaller failures require the propagation of fewer messages, but this metric increases as the size of the failure (i.e. the number of affected links) increases. This result is expected given the generation of topoUpdate and replyUpdate messages defined by SHP. In particular, under this strategy affected nodes, trying to recover their control paths as quickly as possible, forward a topoUpdate message through each of their interfaces. Likewise, a short replyUpdate message is also received by every port of the affected node. Therefore, the degree of the failed node directly determines the resulting packet overhead. We can also observe from the figure that the increase in the number of generated packets corresponds to the network size. This result is due to the propagation of extended replyUpdate messages to notify the SDN controllers of the failure through the shortest path, the length of which (in terms of the number of hops) corresponds to the number of network nodes.

Furthermore, the same trend exists for the different number of controllers. For small failures, the average number of messages is low and approximately the same for all controller values (around 5.67 in Atlanta, 7.69 in Sun and 11.32 in Pioro). However, a slight decrease in the number of generated packets can be observed as the number of controllers increases, and this difference becomes more noticeable when considering the failure of nodes with a higher degree. The reason for this is the reduction in the number of extended replyUpdate messages that are sent to announce the failure, as an increase in the number of controllers reduces the distance between them and the network nodes. In other words, when the number of controllers grows, fewer hops are likely needed to connect them with the neighbors of the affected node, which means that fewer replyUpdate messages are generated along these paths.

Inversely, the behavior exhibited in Fig. 5.12 by the Rerun strategy is in accordance with the eTDP performance. As previously stated, under this topology discovery protocol the number of packets generated in the network corresponds with the average number of the node's neighbors. Therefore, given the reduction in the number of switches with higher connectivity degrees as a result of the considered node failure, the number of packets required for rediscovering the topology is decreased. We can see in this figure that, in all cases, the proposed recovery mechanism significantly outperforms the default Rerun strategy in terms of generated messages with percents of difference that are above 79%, 87% and 89% respectively.

5.4.2.4 Number of Involved Nodes

In this last evaluation, we analyze the number of nodes involved in the operation of the proposed recovery mechanism. The term "involved nodes" takes into account the set of nodes performing different roles in the operation of SHP. Specifically, this set includes the disconnected nodes, their unaffected neighbors (i.e. those that can act as points of recovery since they have an active parent port) and the upstream nodes of those neighbors with relation to the SDN controllers. We use this metric to evaluate the impact of the SHP mechanism on the number of nodes with additional workload as a result of the autonomous operation of this protocol.

Fig. 5.13, Fig. 5.14 and Fig. 5.15 show the number of involved nodes for a different number of controllers and varying degrees of the affected node.

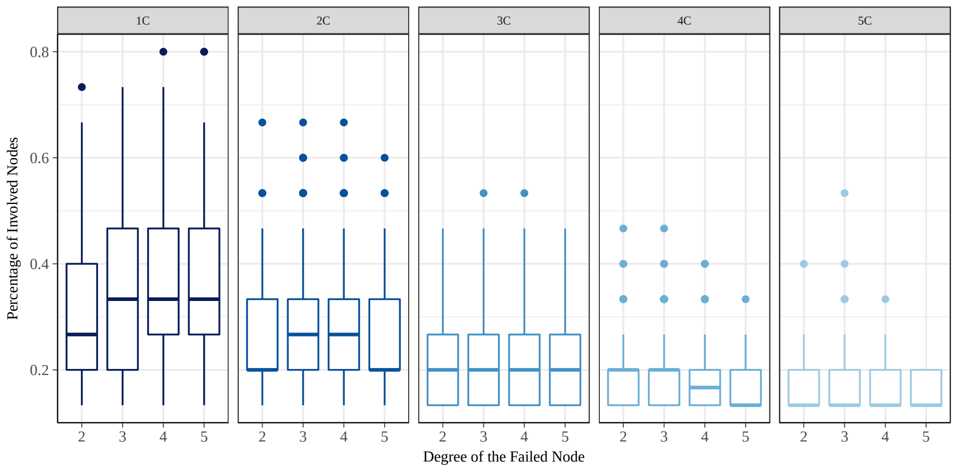


Fig. 5.13: Nodes Involved in the SHP Operation in Atlanta

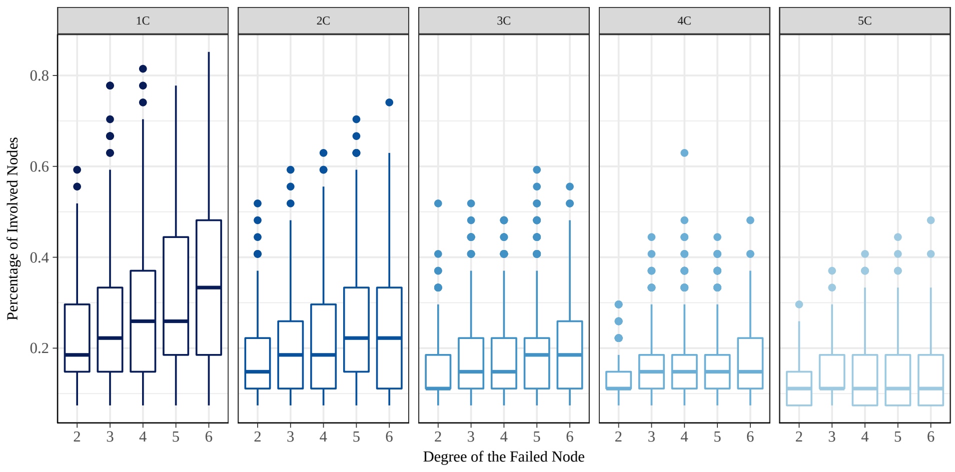


Fig. 5.14: Nodes Involved in the SHP Operation in Sun

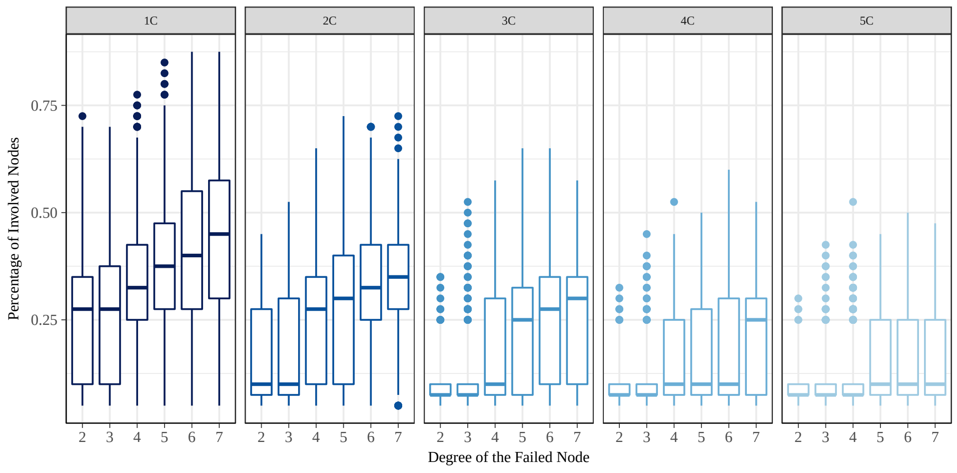


Fig. 5.15: Nodes Involved in the SHP Operation in Pioro

In this case, the Rerun strategy is not included in the plots because the operation of the eTDP mechanism requires the implication of the entire network, increasing the workload of every switch. In contrast, SHP reduces this impact by limiting the scope of the recovery functions to the neighborhood of nodes whose control plane connectivity has failed.

As shown, in the majority of cases for a given number of controllers, the number of switches involved in the recovery strategy increases while the degree of the affected node grows. This behavior is expected given that in our approach the neighbors of the disconnected node, that still have active control paths, are responsible for restoring the control plane connectivity.

Results also show that when the number of SDN controllers is increased, the number of involved switches decreases for a given failed node degree. As previously discussed, increasing the number of controllers reduces the length of branches in the control tree. As a result, a smaller number of nodes is required to send the failure notification to the network controllers.

In summary, in all the cases depicted in Fig. 5.13, Fig. 5.14 and Fig. 5.15, the average values of involved switches are always below 50% of the network nodes. This result reveals a significant merit of the SHP performance –it is able to achieve a fairly reasonable trade-off between reducing the recovery time with autonomic principles and keeping the associated impact in terms of increased workload on the network devices to a minimum.

5.5 Conclusion

In this chapter we proposed a novel SHP mechanism to enhance control plane reliability in SDN-managed networks. To achieve this, we leveraged the self-healing attribute of the ANM paradigm to guarantee the survivability of control connectivity as long as at least one SDN controller remains reachable within the network. The benefits of adopting the SHP are manifold. First, the mechanism uses the fewest possible number of messages (i.e. it has minimal communication overhead) and each message is small in size. Thus, it is easy to implement and yet efficient. Second, network devices can autonomously and stably recover the network from "broken" states with no intervention of an SDN controller. In this way, not only is the workload of the SDN controllers minimized, but the recovery times, packet loss probability and the memory requirements of forwarding devices are also reduced. In addition, once connectivity is recovered throughout the control tree topology, SDN controllers can optimize the recovered control plane by evaluating the requirements of the supported network applications. In addition, the results obtained in the experimental simulation reflect the time efficiency and scalability of the proposed solution across various key network metrics. Specifically, the recovery of the control connectivity was assured with recovery times below $20\mu s$ for all the performed simulations. This result confirmed the suitability of the recovery mechanism for application in carrier-grade networks, which require recovery times of less than 50 ms.

Chapter 6

Dynamic VNF Placement in SDN/NFV

This chapter is based on:

- **L. Ochoa-Aday**, C. Cervelló-Pastor, A. Fernández-Fernández and P. Grosso, "An Online Algorithm for Dynamic NFV Placement in Cloud-Based Autonomous Response Networks," *Symmetry*, vol. 10, no. 5, pp. 163:1–163:18, May 2018.

This chapter addresses the dynamic placement of virtual functions in SDN/NFV environments. To this end, we leverage the traditional bin packing scheme (i.e. low-complexity algorithms from the online optimization theory) in order to develop a heuristic approach that efficiently deploys VNFs on top of virtualized network infrastructures.

6.1 Introduction

Having discussed efficient approaches for discovering and maintaining an accurate network topology view for the SDN controller in the previous chapters, we now use this information to address the problem of VNF allocation in software-defined and virtualized networks.

This chapter evaluates the suitability of different VNF placement strategies using the SFC concept. In this study, we consider both static (offline) and dynamic (online) scenarios. The offline scenario assumes a system aware of the full set of SFC requests. In contrast, the online scenario considers a system with a set of already deployed SFCs, where new incoming SFC requests are processed in a one-by-one modality. As a complement, we also derive analytical expressions for two evaluation metrics to assess the network performance of the identified placement solutions.

Differing from previous works, this contribution aims to provide a low-complexity solution for the online VNF placement problem in combination with two useful metrics to assess the proposal's suitability with respect to different network topologies and SFC characteristics.

Specifically, the contributions of this chapter are summarized as follows:

- Formulation of the network function placement and chaining problem as an ILP for the offline approach.
- Heuristic algorithm based on the WorstFit approach (i.e. an optimal solution of the bin packing problem) to efficiently place the VNFs in an online scenario.
- Evaluation parameters denoted as consolidation and aggregation to assess the impact of network topology and system demands (i.e. requested SFCs) on the placement solutions.

The remainder of this chapter is organized as follows. After describing the system model in Subsection 6.2.1, we formulate the resource allocation problem in Subsection 6.2.2. Afterward, we define the proposed online heuristic algorithm in Section 6.3. We then introduce two evaluation metrics and present the considered simulation setup in Section 6.4. We discuss some experimental results in Section 6.5. Lastly, we summarize our findings in Section 6.6.

6.2 Problem Statement

In this section we formalize the system model and provide an ILP model for the network function placement and chaining problem. The model aims at strategic deployment of VNFs that uses existing physical resources efficiently. Simultaneously, it also attempts to minimize the number of physical links used between deployed VNFs, thereby decreasing the associated network costs.

6.2.1 System Model

The following system model defines three major components, namely network infrastructure (i.e. the cloud virtualized infrastructure in which VNF chains are placed), the VNFs and service function chains. We study an SDN/NFV system in which network functions are offered as a service.

Fig. 6.1 presents the basic network scenario for the VNF placement. It primarily consists of forwarding network devices (i.e. programmable switches or IP routers) and NFV compliant nodes

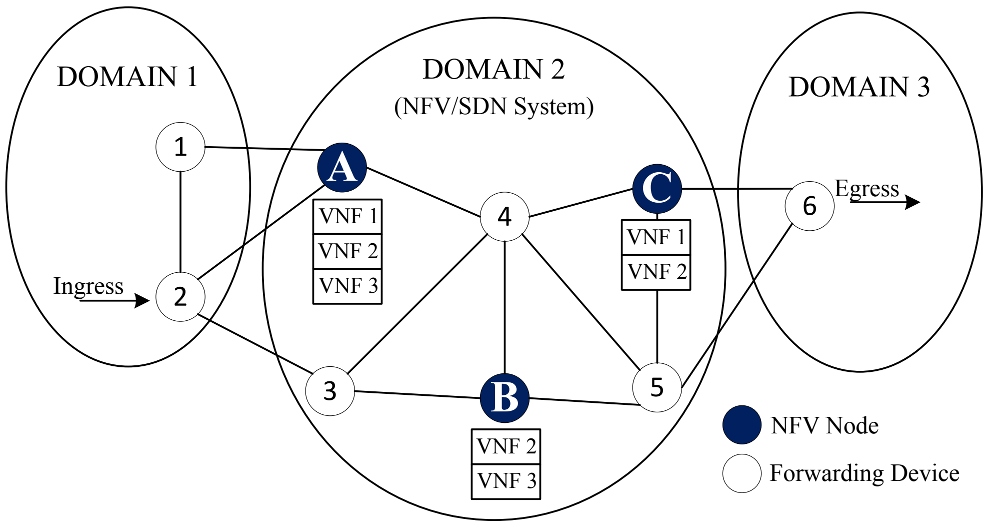


Fig. 6.1: Basic Network Scenario in VNF Placements

that can host a number of VNFs according to their assigned resources. In the presented model, several VNFs can run on the same NFV node through isolated containers or VMs (e.g. three specific VNFs on node A). Due to the limitations of assigned resources in NFV nodes, different instances of one VNF may be distributed between two or more NFV nodes (e.g. instances of VNF 2 are placed in nodes A, B and C). Controllers that are SDN-based can manage traffic among virtual functions according to requested resources in the SFCs (i.e. maximum delay, capacity, etc.).

6.2.1.1 Network Infrastructure

In this work, we have modeled the network topology as a connected directed graph, $G = (N, E)$, where N and E denote the sets of nodes and links, respectively, in the existing infrastructure. The aforementioned network elements have limited resources to support the requirements of a certain number of SFCs.

Regarding resources on nodes, we use CPU to represent overall resource capacity. However, this approach can be easily extended to take into account several types of resources (such as memory, disk, etc.). We represent the CPU capacity of each network node $n \in N$, as c_n. Moreover, we extend the model adding the delay associated to each network link $(i, j) \in E$, represented as $d_{i,j}$. Additionally, we have defined different network domains in our model as shown in Fig. 6.1.

6.2.1.2 Virtual Network Functions

Virtual network functions refer to the virtualized network entities (e.g. firewall, load balancer, etc.) that may be instantiated by the operator on top of a cloud infrastructure. In this model, the set of available VNFs is denoted as F. For each VNF $f \in F$, the maximum number of instances is U_f, due to the number of licenses that the operator owns for the VNF.

Virtual network functions can be implemented in containers or VMs running on the network infrastructure. In such cases, they require a certain volume of resources according to the requirements of their traffic demands. Additionally, we have denoted d_f as the processing delay associated with a network function $f \in F$.

6.2.1.3 Service Function Chains

A service function chain is composed of the sequence of VNFs that a given service must traverse. In this model, we denoted Q as the set of incoming SFCs, and we formalized each SFC using the context-free language described in [103].

Following this specification, each service chain $q \in Q$ provides its ingress and egress node, denoted as s_q and t_q, respectively. Additionally, the order of requested VNFs $f \in F_q$ between both endpoints is also specified, where $F_q \subset F$ is the subset of VNFs requested for this SFC q. They also contain the required CPU capacity (denoted as $R_{q,f}$) for each requested VNF. Moreover, the maximum admissible end-to-end delay between the ingress and egress points, denoted as D_q, is also included. We have defined E'_q as the set of virtual links (k, l) that connect the requested VNFs between the ingress and egress points.

In Table 6.1 we show a basic example of a described SFC.

Table 6.1: Basic Example of Service Function Chain Request

Parameter Name	**Example Value**
Ingress Point	node 2
Requested VNFs	VNF1, VNF2, VNF3
CPU Requirements	[10, 10, 10] (units)
Delay Requirement	50 ms (end-to-end)
Egress Point	node 6

6.2.2 ILP Model (exact approach)

In the following section, we define the network function placement and chaining problem as an ILP model. First, the decision variables and the objective function are introduced, followed by the set of constraints.

6.2.2.1 Variables of the Model

$$y_f^n = \begin{cases} 1, & \text{if an instance of the VNF } f \in F \text{ is located in node } n \in N, \\ 0, & \text{otherwise.} \end{cases}$$

$$x_{q,f}^n = \begin{cases} 1, & \text{if VNF } f \in F \text{ required by SFC } q \in Q \text{ is serviced in node } n \in N, \\ 0, & \text{otherwise.} \end{cases}$$

$$v_{q,k,l}^{i,j} = \begin{cases} 1, & \text{if virtual link } (k,l) \in E'_q \text{ required by the SFC } q \in Q \text{ is hosted on the edge } (i,j), \\ 0, & \text{otherwise.} \end{cases}$$

6.2.2.2 Objective Function

The objective function presents a general formulation that jointly optimizes the placement of the network functions and virtual links in the network. The first term of this function seeks to reduce the number of VNF instances placed across the network, while the second term is intended to map the virtual links, decreasing the delay among deployed VNFs.

$$min \left\{ norm(\sum_{n \in N} \sum_{f \in F} y_f^n) \; + \; norm(\sum_{(i,j) \in E} \sum_{q \in Q} \sum_{(k,l) \in E'_q} v_{q,k,l}^{i,j} \cdot d_{i,j}) \right\} \tag{6.1}$$

To avoid dominant effect in Eq. (6.1), the possible values of the different terms are normalized into the interval (0, 1) using the $norm(x)$ function.

6.2.2.3 Demand Satisfaction Constraints

$$\sum_{n \in N} x_{q,f}^n = 1 \qquad \forall q \in Q, \; f \in F_q \tag{6.2}$$

These constrains ensure that every required SFC and its respective network functions are mapped to the network infrastructure.

$$\sum_{f \in F} y_f^n \; \geq \; x_{q,f}^n \qquad \forall n \in N, q \in Q, f \in F_q \tag{6.3}$$

These constraints ensure that if a network function requested by an SFC is assigned to node n, then at least one instance of the requested network function will be placed on that node.

6.2.2.4 Capacity Constraints

$$\sum_{q \in Q} \sum_{f \in F_q} x_{q,f}^n \cdot R_{q,f} \; \leq \; c_n \qquad \forall n \in N \tag{6.4}$$

The capacity constraints guarantee that the sum of CPU capacities required by placed VNF instances does not exceed the available resources in each network node.

6.2.2.5 Path Creation Constraints

$$\sum_{j \in N} v_{q,k,l}^{i,j} \; - \; \sum_{j \in N} v_{q,k,l}^{j,i} \; = \; x_{q,k}^i \; - \; x_{q,l}^i \qquad \forall q \in Q,\; i \in N | (i,j) \in E,\; (k,l) \in E'_q \tag{6.5}$$

These equations are derived from the flow conservation constraints and ensure the proper building of the virtual paths between the required endpoints.

6.2.2.6 Latency Constraints

$$\sum_{(i,j) \in E} \sum_{(k,l) \in E'_q} v_{q,k,l}^{i,j} \cdot d_{i,j} \; + \; \sum_{n \in N} \sum_{f \in F_q} x_{q,f}^n \cdot d_f \; \leq \; D_q \qquad \forall q \in Q \tag{6.6}$$

These constraints ensure that end-to-end latency requirements on mapped SFC requests will be met. Accordingly, Eq. (6.6) includes a sum of the delay incurred by end-to-end latencies between mapped endpoints and the second part defines the delay incurred by packet processing on VNFs.

These constraints ensure that end-to-end latency requirements on mapped SFC requests will be met. Accordingly, Eq. (6.6) is composed by a sum of the delay incurred by end-to-end latencies between mapped endpoints and packet processing on virtual network functions.

6.2.2.7 Other Constraints

$$x^{s_q}_{q,s_q} = 1 \qquad \forall q \in Q \qquad (6.7)$$

$$x^{t_q}_{q,t_q} = 1 \qquad \forall q \in Q \qquad (6.8)$$

These constraints ensure that the required ingress and egress points, respectively, are mapped to devices in the requested physical locations.

6.2.3 Model Considerations

Although the model above computes the optimal NFV placements for a given network, it becomes challenging to solve in large and even medium-scale topologies. This is because the NFV resource allocation problem is known to be NP-Hard [69], meaning the consumption of resources and time complexity grow exponentially with the network size. The ILP model also requires awareness of the full set of expected SFC requests, which might be impractical for the online planning of network resources. To overcome these challenges, in the next section we propose an online heuristic strategy, which is more suitable for dynamic cloud-based environments.

6.3 Online Placement of Virtual Network Functions

In this section we present an online algorithm to address the VNF placement problem. We consider a scenario with previously allocated VNFs and with new, incoming SFC requests that need to be provisioned while minimizing the usage of physical resources. In the following subsections we explain the operation of the Topology Aware Placement of Virtual Network Functions (TAP-VNF) algorithm and analyze its computational complexity.

6.3.1 The TAP-VNF Heuristic (Heuristic Approach)

We have designed an online heuristic scheme that determines the placement of the VNFs and chains them together to reduce the overall volume of resources used in the network. For such purposes, we have observed that efficient algorithms for solving the bin packing problem are especially suitable for approaching the online VNF placement problem in SDN/NFV environments [108]. After being provided a set of items (i.e. requested NFV instances) to be inserted into the bins (i.e. set of VNF nodes in the network), algorithms based on fit strategies such as

FirstFit, BestFit and WorstFit process one item at a time in an arbitrary order and attempt to place them in a bin according to a certain strategy. If no bin is found, they open a new bin and place the item there.

In this work we specifically leverage the suitability of the WorstFit algorithm for the problem described in Section 6.2.2 to develop the TAP-VNF algorithm. This algorithm takes into consideration the limited resources of the underlying network infrastructure and the requirements of the requested SFCs. Therefore, it efficiently allocates the requested VNF instances while attempting to minimize server and link costs. The pseudocode of the proposed VNF placement strategy for online scenarios is shown in Algorithm 4.

Algorithm 4 TAP-VNF Algorithm

```
Require: $G$, $q$, $s_q$ and $t_q$, $P$, $A$
Ensure: ($\Gamma$) placement and chaining of requested VNFs in $q$
 1: $\Gamma \leftarrow$ NULL
 2: for path $p \in P$ do
 3:   if total remaining capacity of $p < \sum_{f \in F_q} R_{q,f}$ or length($p$) $> D_q$ then
 4:     continue
 5:   end if
 6:   $C \leftarrow$ Array of node's remaining capacities ($c_n$) in $p$
 7:   $\Gamma \leftarrow$ MOD_WORSTFIT($q, C, p, A$)
 8:   if $\Gamma \neq$ NULL then
 9:     return $\Gamma$
10:   end if
11: end for
```

The main loop of the algorithm determines the placement and chaining solutions for the considered path $p \in P$ provided by the MOD_WORSTFIT function (i.e. a modified version of the WorstFit scheme). If the considered path allows a feasible solution, it is returned and the algorithm ends without further iterations. However, if no solution can be found using the current path, the algorithm iterates over the next shortest path, which is subsequently analyzed.

As inputs, the TAP-VNF algorithm requires the network topology G and the requested SFC q along with its ingress and egress nodes in the network. Additionally, the list of paths between ingress and egress nodes (sorted in increasing order of delay) P and the set of previously allocated VNFs on each node A, are passed to the algorithm. Note that the size of the set P can be limited by a maximum number of paths k, which may be required in network topologies with higher path redundancy.

Once a new SFC is requested, Algorithm 4 begins evaluating the shortest path between the ingress and egress nodes of SFC q. First, the algorithm determines if the considered path p is suitable for allocating the incoming SFC q. To do this, TAP-VNF determines if the total remaining capacity of the considered path p is sufficient to place all the required functions and if the length of the considered path is not larger than the maximum admissible delay. In this way, end-to-end latency requirements are satisfied by the algorithm.

For the considered path $p \in P$, the main loop of the algorithm determines the placement and chaining solutions provided by the Mod_WorstFit function (i.e. a modified version of the WorstFit scheme). If the considered path allows a feasible solution, this path is returned and the algorithm ends without further iterations. However, if no solution can be found using the current path, the algorithm generates the next shortest path and analyzes it.

Fig. 6.2 presents the sequence of actions of the modified Mod_WorstFit strategy in a flow chart for each function f requested by the incoming SFC q. This procedure evaluates each intermediate node n along the path p (except the ingress and egress nodes) as a possible candidate to allocate the considered function and computes the tentative node utilization (i.e. U'_n) as a result of the allocation of the function f with the demanded resources $R_{q,f}$.

After confirming that a node n has sufficient available CPU resources to allocate the required function, the Mod_WorstFit method determines if this function has been already allocated in n. In the affirmative case (i.e. some instance of f exists in n) the same node is used again to place this function, unlike the traditional WorstFit strategy.

Otherwise, after the considered allocation of f (i.e. $c_n - U'_n$), the available CPU resources in n are compared with the maximum remaining resources (max_V). If a new maximum is found, the considered n becomes the preferred node. Then, after considering all nodes, the preferred node is used to place the function. Additionally, the set of allocated functions and utilized nodes are updated.

6.3.2 Complexity Analysis

Although this heuristic approach does not provide optimal solutions, it outperforms the exact approach in computation times. In this subsection, we derive the complexity of the proposed TAP-VNF algorithm. The overall complexity of Algorithm 4 is primarily determined by the for

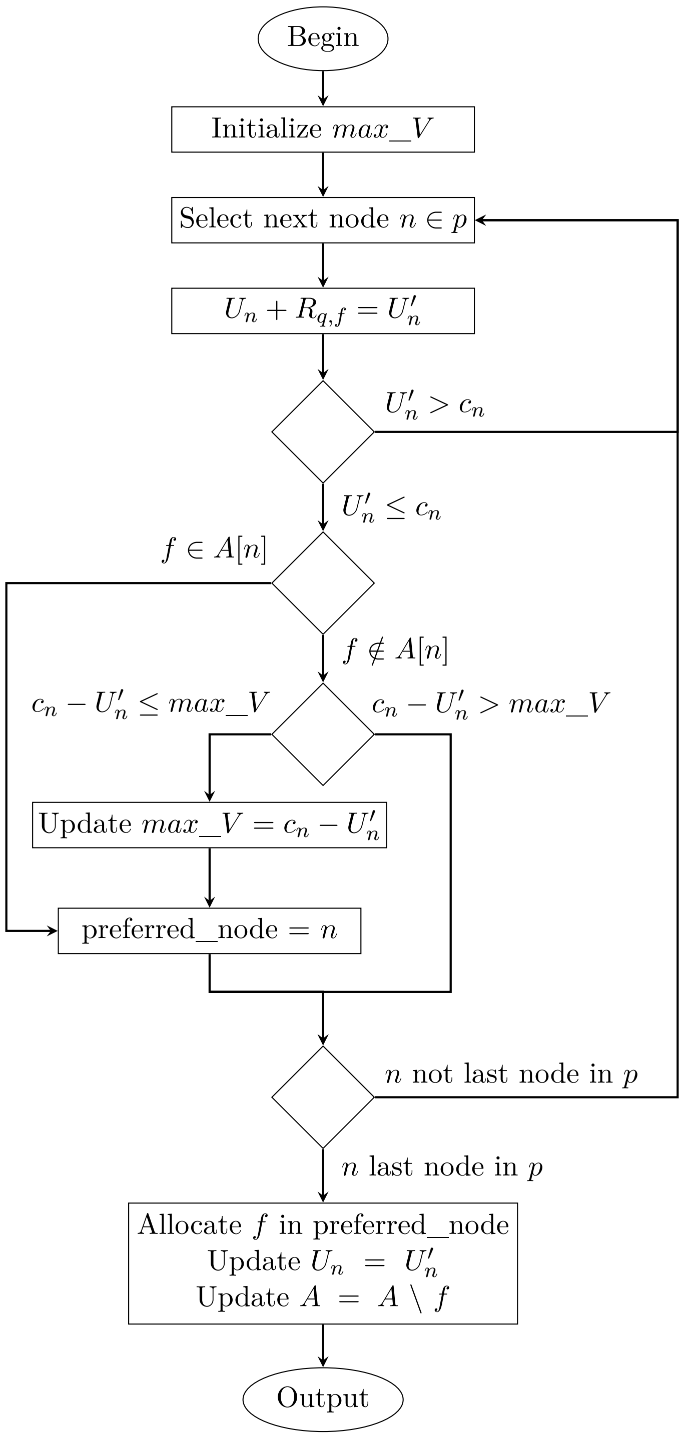


Fig. 6.2: Flow Chart of the MOD_WORSTFIT Scheme

loop in lines 2-11 and the MOD_WORSTFIT function.

Although this algorithm attempts to use the first admissible path (i.e. the one satisfying capacity and delay requirements), to provide the placement and chaining solution, in the worst case this for loop iterates k times over the sorted list of paths between the ingress and egress nodes. During each iteration, it applies the MOD_WORSTFIT scheme $|F|$ times in order to place each VNF instance of the requested SFC q. This operation is executed to select the node where the requested VNF will be instantiated.

In this work, an efficient implementation of the WorstFit algorithm is used to compute the placements. The code uses complex data structures to lower the running times to $\mathcal{O}(|N| \log |N|)$, where $|N|$ is the maximum number of nodes in a path p. It should be noted that modifications to the WorstFit approach do not include any loop structure. Therefore, the low-complexity property of this bin packing algorithm is preserved. As a result, the overall worst-case complexity of the proposed TAP-VNF algorithm is $\mathcal{O}(|k||F||N| \cdot \log |N|)$. This online heuristic approach could be implemented in high-performance SDN/NFV servers and be executed in a real-time fashion for such systems.

6.4 Evaluation Metrics and Simulation Setup

Currently, there is no standard evaluation model to assess different VNF placement strategies [103]. Therefore, we introduce two metrics towards the evaluation of VNF placements here, namely consolidation and aggregation, taking into consideration the parameters of the network infrastructure and characteristics of requested SFCs.

6.4.1 Consolidation Parameter

The consolidation metric represents the fraction of deployed VNF instances over the total sum of VNFs requested by the overall set of SFC demands.

In order to compute the consolidation parameter, we derived Eq. (6.9), which considers the number of placed VNF instances on the network infrastructure with respect to the sum of all requested VNF instances in the full set of demands.

$$consolidation = \frac{\sum\limits_{n \in N} \sum\limits_{f \in F} y_f^n}{\sum\limits_{q \in Q} \mid F_q \mid} \tag{6.9}$$

6.4.2 Aggregation Parameter

The aggregation metric represents the fraction of physical links $i, j \in E$ used to host the virtual links required by all SFC requests with respect to the full set of requested virtual links.

In order to calculate this parameter, we include the binary variable $Z_{i,j}$, which takes value 1 when the physical link $(i, j) \in E$ is used. Eq. (6.10) also provides a notion of how close the VNF instances are placed within the network infrastructure. Therefore, lower values of this ratio mean that network costs are also minimized.

$$aggregation = \frac{\sum\limits_{i,j \in E} Z_{i,j}}{\sum\limits_{q \in Q} \mid E'_q \mid} \tag{6.10}$$

6.4.3 Simulation Environment

We conducted extensive simulations to assess the performance of the TAP-VNF algorithm. The parameters considered in our environment setup are described below.

All experimental simulations were performed on the network graph Abilene (11 nodes, 28 links), taken from the SNDlib dataset [119]. The selected topology represents a typical topology of the Internet and Internet Service Provider (ISP) networks. The delay on each link is assumed to be the propagation latency calculated from the geographical distance between nodes in the topology. For all experimental simulations we used a fixed number of 100 CPU units for each node of the considered topology. As in [125], we assumed that all nodes in the topology can be used to deploy VNFs. In these simulations, we used five types of VNFs (i.e. VNF1, VNF2, VNF3, VNF4 and VNF5) requiring fixed or variable (random) CPU units. The resource requirements are given in Table 6.2.

Table 6.2: Resource Requirements for Network Functions

Number of	CPU Requirements (Units)	
VNFs	Fixed	Random
5	10	[1, 20]

To accurately assess the performance of the proposed solution, we carefully designed four evaluation profiles using sets of SFC requests with different parameters. A summary of considered profiles is presented in Table 6.3.

Table 6.3: Parameters of SFC Profiles

SFC Profile	VNF Requirement	Distribution	Mean	Var
Scenario I	fixed	uniform	0.2	0.0
Scenario II	random	uniform	0.2	0.0
Scenario III	fixed	non-uniform	0.2	0.043
Scenario IV	random	non-uniform	0.2	0.043

The SFC demands were generated with a random number of VNFs (from 2 to 5) and ingress/egress endpoints. The selection of these nodes was ensured to be at least two hops away. These evaluation profiles are used to provide important insights into the performance of the TAP-VNF algorithm.

The heuristic TAP-VNF was developed using the programming language Python. We implemented the ILP model using the linear programming solver Gurobi Optimizer [126]. Additionally, each simulation instance was tested 15 times, setting different random seeds on the selected topology. All computations were performed on a Laptop equipped with 3.30 GHz Intel Core i7 and 32 GB RAM.

6.5 Results and Discussion

In this section, we first provide an in-depth examination of the optimal results achieved by the ILP model. Afterward, we assess the proposed TAP-VNF algorithm and validate its results using the exact approach as well as traditional bin packing strategies.

The consolidation and aggregation parameters are used to discuss the performance of the different approaches. After this, we present computation times required by each solution. Finally, the performance of TAP-VNF on several real-world topologies is discussed.

6.5.1 Optimal Solution Assessment

For this experimental simulation the SFC requests were generated using scenario 1 from Table 6.3. In order to determine all of the optimal placements for each set of SFC requests, we independently performed each simulation with an increased number of SFC requests.

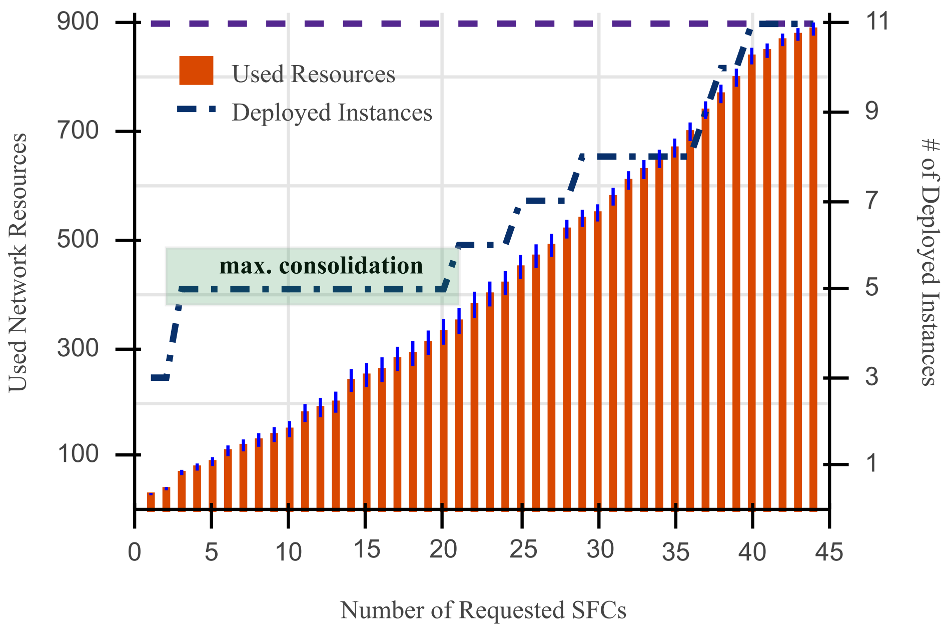
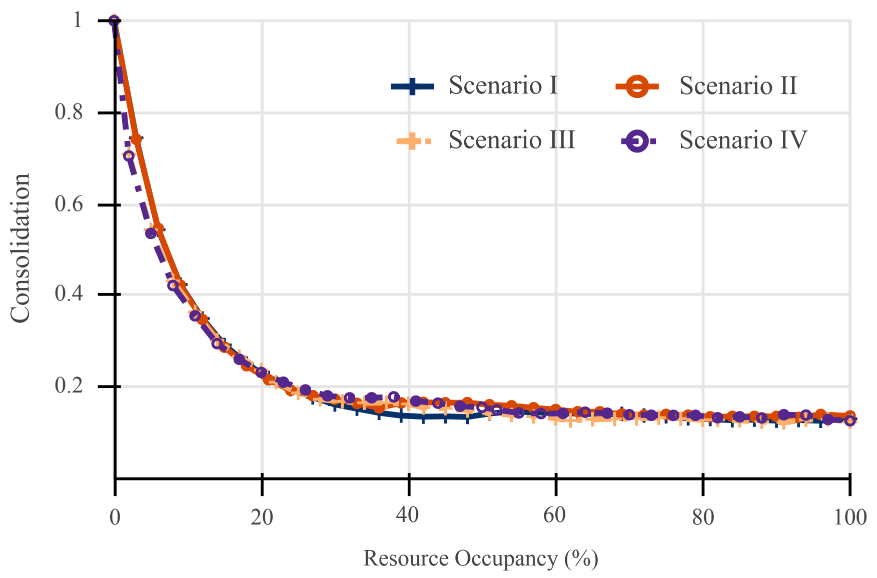


Fig. 6.3: Optimal Resource Allocation for the ILP Method

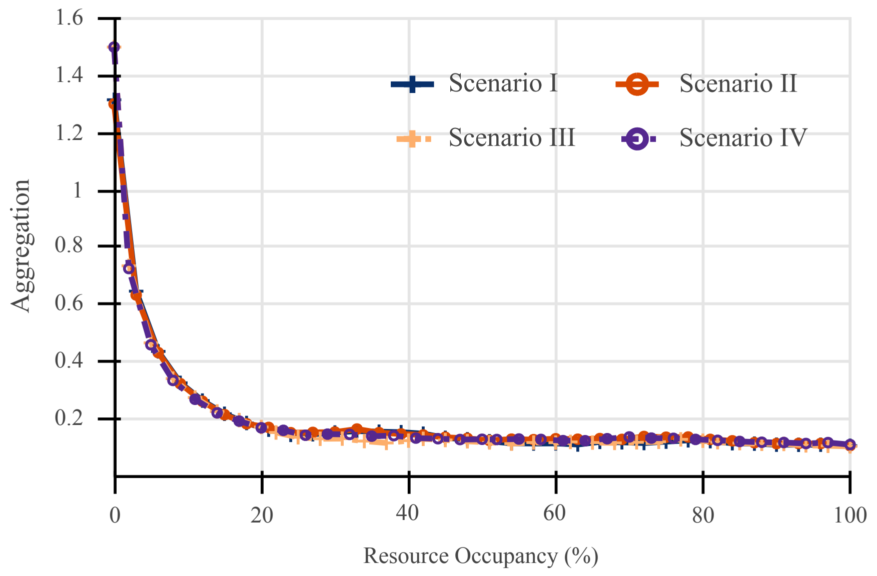
Fig. 6.3 presents the network resources used by the ILP model as well as the average number of deployed VNF instances. The x-axis represents the number of requested SFCs, which are increased until the existing network resources are fully utilized. Precisely, the overall volume of available resources in the presented topology is emphasized in the figure by the dashed line at the top (i.e. $y = 900$).

It can be observed that the exact approach accomplishes all the placements using 11 VNF instances. As expected, when the number of resources requested by SFCs increases, the number of deployed VNFs also grows. More interesting is the behavior manifested in the interval denoted in the figure as *max. consolidation*, where the increase in requested resources does not influence the number of deployed VNF instances. Specifically, although an additional 260 CPU units are requested by 20 SFCs with respect to 3 SFCs, the average number of deployed VNF instances remains the same at 5 instances. This desired behavior of optimal placements exhibits great scalability and decreases licensing costs substantially.

To obtain insights into the ILP model performance across all the evaluation profiles in Table 6.3, we used the evaluation metrics proposed in Section 6.4. Consequently, in Fig. 6.4 we display the performance evaluation for optimal VNF placements in terms of the consolidation and aggregation ratios as functions of the resource occupancy efficiency. These metrics are plotted against the percentage of resource occupancy, which is directly related to the number of



(a) Optimal Consolidation of Requested VNF Instances



(b) Optimal Aggregation of Requested Virtual Links

Fig. 6.4: Performance Evaluation of the ILP Model in Abilene Topology

requested SFCs in each scenario.

Fig. 6.4(a) shows the consolidation ratios of the exact approach. Given the nature of this metric, higher values are obtained when the number of requested SFCs is low. However, under more demanding workloads the consolidation effect becomes more appreciable. Specifically, this model can achieve values lower than 0.2 once the 20% resource occupancy is exceeded. For 100% resource occupancy, consolidation values lower than 0.14 are achieved in the four scenarios. This result means that the optimal solution only requires instantiates 14% of the requested VNFs to satisfy all the CPU demands in SFC requests. It is worth highlighting that similar results are attained by the offline approach in all the considered evaluation profiles.

Fig. 6.4(b) shows the attained aggregation ratios. As expected, lower aggregation is achieved when the number of incoming SFCs is low. This behavior occurs because the first incoming SFC is likely to have fewer virtual links than the number of physical links required to map them from the ingress to the egress nodes across the network topology. Consequently, values higher than 1 are obtained at the beginning of the simulations. Similarly, aggregation ratios lower than 0.2 are achieved by the exact model irrespective of the considered parameters of SFC profiles.

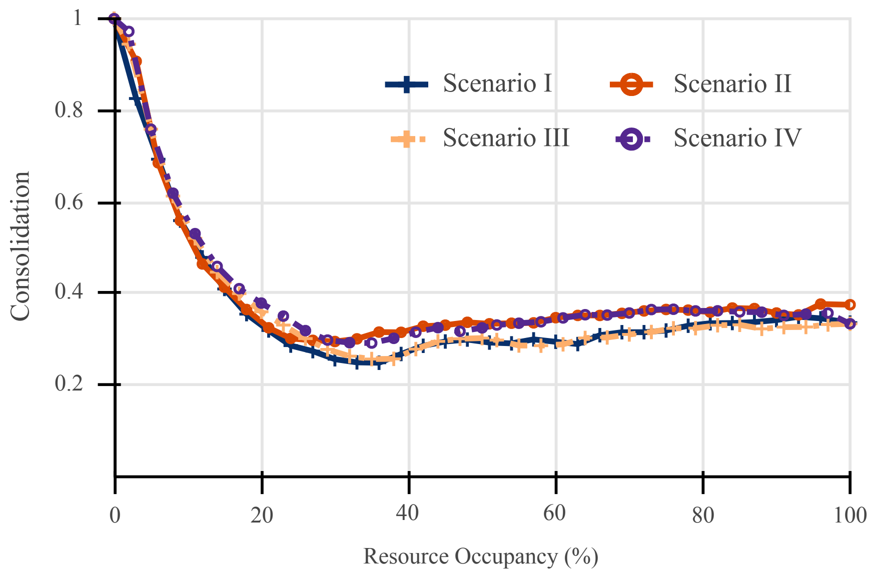
In this experimental simulation the lowest aggregation ratios achieved are approximately 0.1 (at 100% resource occupancy). This result means that each physical link of the network infrastructure can host around 10 virtual links of the full set of deployed SFC requests. Consequently, network operators can use fewer network resources (e.g. data links and bandwidth) to connect their supported network services.

6.5.2 Performance of TAP-VNF

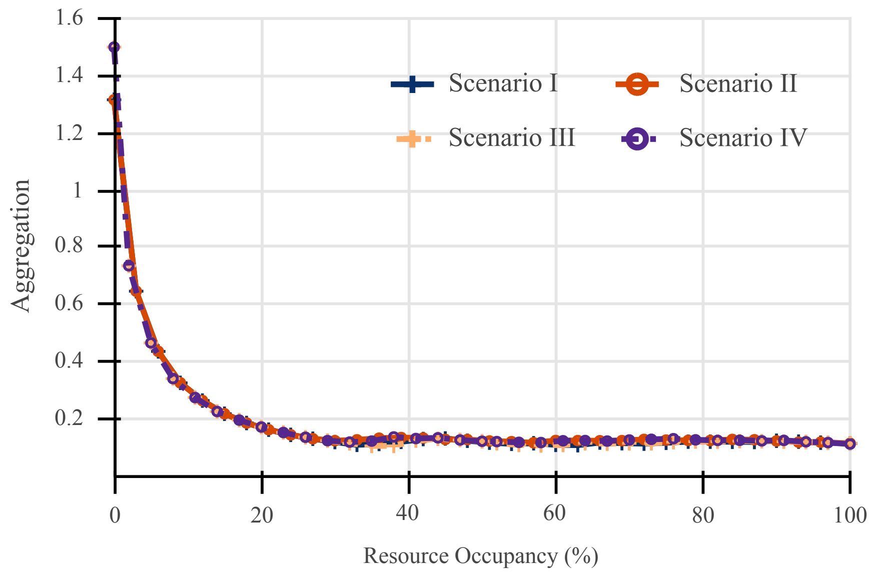
We also present the heuristic algorithm results for the consolidation and aggregation metrics using the same network topology and SFC profiles. The TAP-VNF algorithm exploits the key idea behind the consolidation parameter and attempts to scale up the previously deployed VNF instances in order to reduce operation costs.

Fig. 6.5 shows the performance evaluation after running the TAP-VNF algorithm on the Abilene topology. As can be seen in Fig. 6.5(a), the consolidation values of the online approach exhibit the same pattern as the offline model, which verifies that our proposed algorithm preserves the desired properties. In this case, consolidation ratios under 0.4 are achieved once resource occupancy exceeds 20%.

Aggregation ratios in Fig. 6.5(b) exhibit almost identical behavior to those in Fig. 6.4(b) compared to the offline approach regarding the mapping of virtual links over the physical infrastructure. Despite the high aggregation values at the beginning of the SFC requests, the more resources are required, the better the aggregation ratios achieved by this online approach. This figure confirms that the TAP-VNF algorithm can reach lower ratios of aggregation as the number of requested SFCs increases. For instance, at 100% resource utilization the online algorithm achieves a minimum aggregation ratio of 0.11, which is less than an 8% difference compared to



(a) Heuristic Consolidation of Requested VNF Instances



(b) Heuristic Aggregation of Requested Virtual Links

Fig. 6.5: Performance Evaluation of the Online NFV Placement Strategy

the optimal values for the same conditions.

As in the previous cases, similar results are achieved in the four considered scenarios. This result confirms the adequate performance of the proposed solution for VNF placements in online scenarios.

In order to further evaluate the effectiveness of the proposed algorithm, Fig. 6.6 and Fig. 6.7 present the performance comparison with the ILP model using both metrics when network resources are 100% utilized. Additionally, we also include three traditional bin packing schemes, namely BestFit, FirstFit and WorstFit. Specifically, these strategies are applied in the procedure described by the pseudocode of Algorithm 4 to provide the placement and chaining solutions instead of the MOD_WORSTFIT (line 7). In these figures, it can be seen that the TAP-VNF algorithm reaches close to optimal values. Although smaller optimality gaps are observed for the aggregation metric, the proposed online approach achieves acceptable consolidation ratios.

In Fig. 6.6 the proposed online algorithm clearly outperforms the traditional bin packing strategies in terms of consolidation ratios. This result occurs because the MOD_WORSTFIT function allocates requested VNF prioritizing nodes where instances of these functions are already placed.

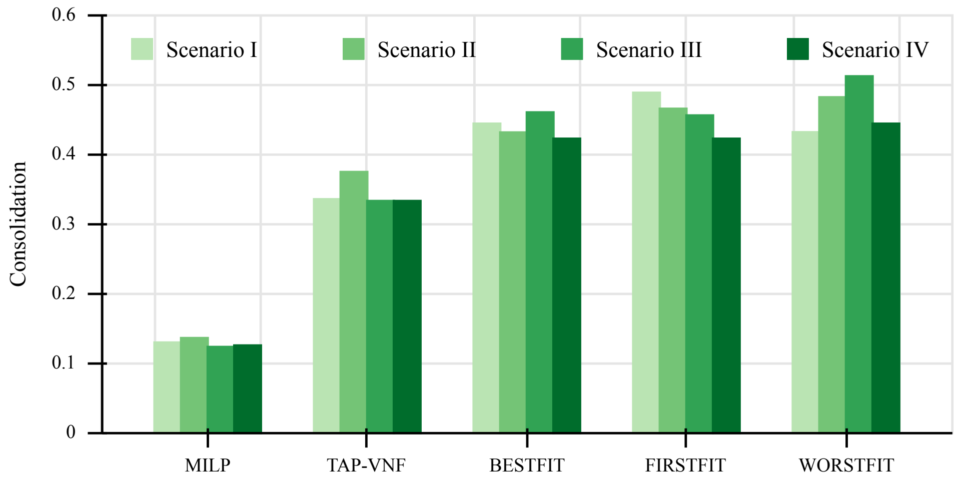


Fig. 6.6: Consolidation between Optimal and Heuristic Solutions

Regarding aggregation ratios, similar results are achieved by all of the evaluated approaches in Fig. 6.7.

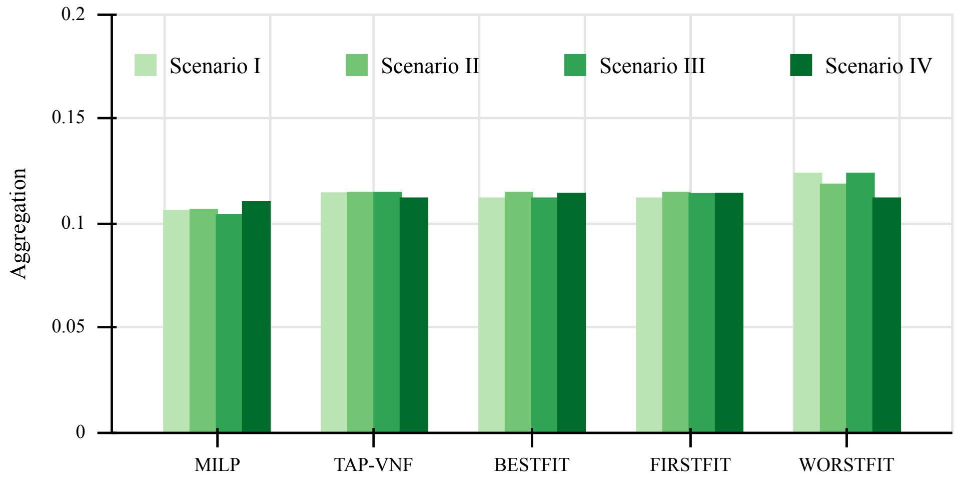


Fig. 6.7: Aggregation between Optimal and Heuristic Solutions

In this case, slightly better values can be attained by the BestFit and FirstFit strategies compared with the TAP-VNF approach. This result is expected due to the characteristics of these bin packing algorithms. While the first two strategies allocate as many requested VNFs as possible on the same node (without requiring any physical link), the WorstFit strategy attempts to place different requested VNFs on different nodes, resulting in the utilization of more physical links.

6.5.3 Execution Time

Another important aspect for consideration is the execution time required by the presented solutions. To evaluate this, Fig. 6.8 shows the average running time of the different placement strategies. In this figure, the requested SFCs were created following the parameters defined by Scenario I in Table 6.3. Similarly, we used the Abilene topology due to the long running times of the ILP model.

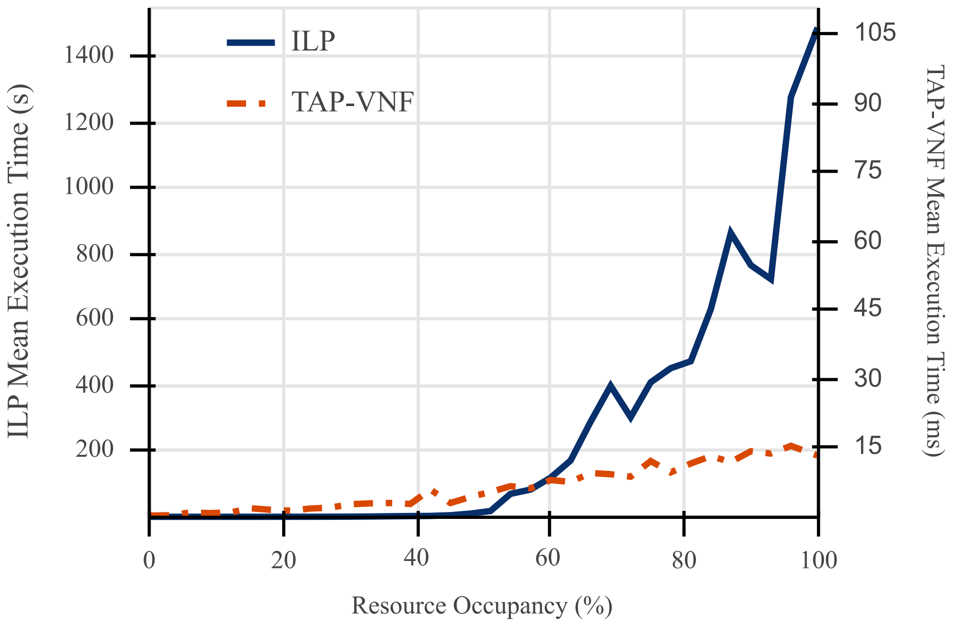


Fig. 6.8: Execution Time Performance across Proposed Placement Strategies

It should be highlighted that in Fig. 6.8, the TAP-VNF algorithm and the ILP model differ drastically in terms of computing time. Although the proposed strategy outperforms the optimal model in all cases, a greater improvement is achieved as the resource occupancy grows. For instance, when the resource occupancy exceeds 50 %, the processing time required by the optimal model increases dramatically, which is a significant limitation for current networking environments. Ultimately, the TAP-VNF algorithm accomplishes good consolidation and aggregation ratios extremely quickly (in less than 15 ms) while the ILP model can take up to 25 min to determine a solution. The TAP-VNF algorithm thus presents a fifth order of magnitude improvement over the ILP model. As far as scalability is concerned, the execution times indicate that TAP-VNF is more scalable —a characteristic that is crucial in practical applications. These considerations suggest that it would be reasonable to use TAP-VNF for dynamic resource allocation in software-defined and virtualized networks. Fig. 6.9 confirms that the TAP-VNF algorithm is of similar complexity to traditional bin packing schemes.

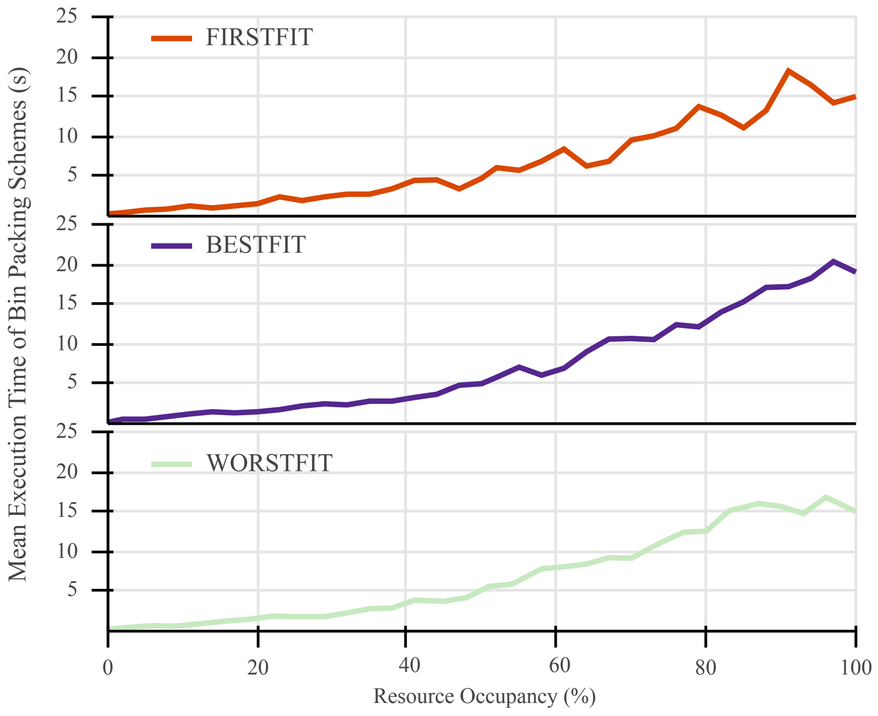
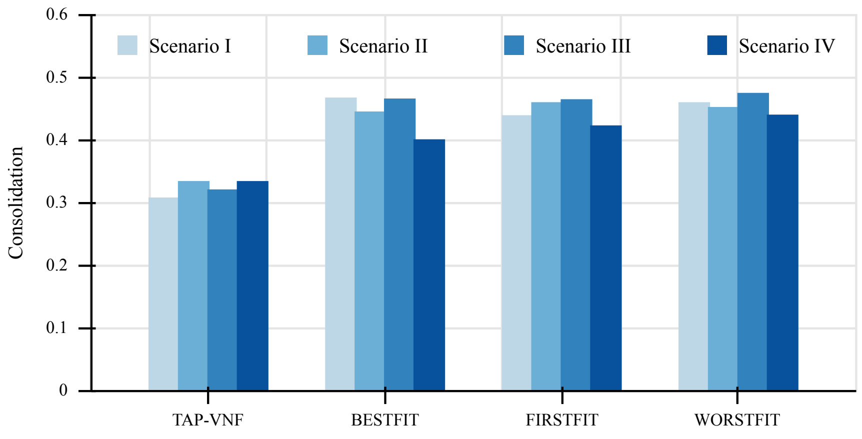


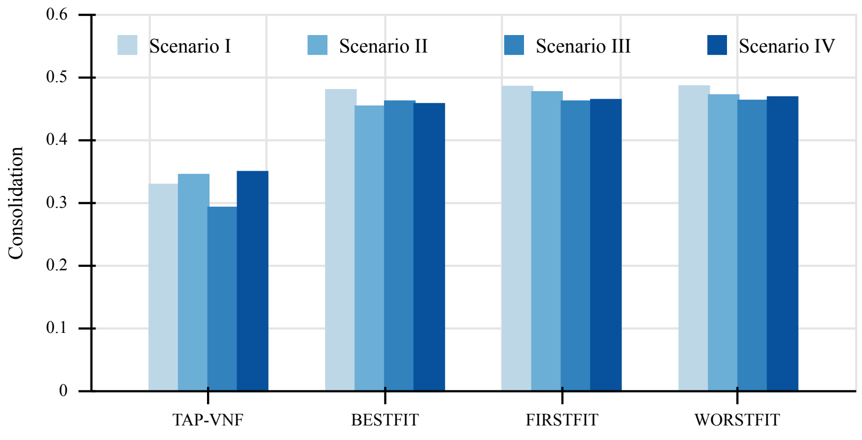
Fig. 6.9: Execution Time Performance across Bin Packing Schemes

6.5.4 Analysis of More Topologies

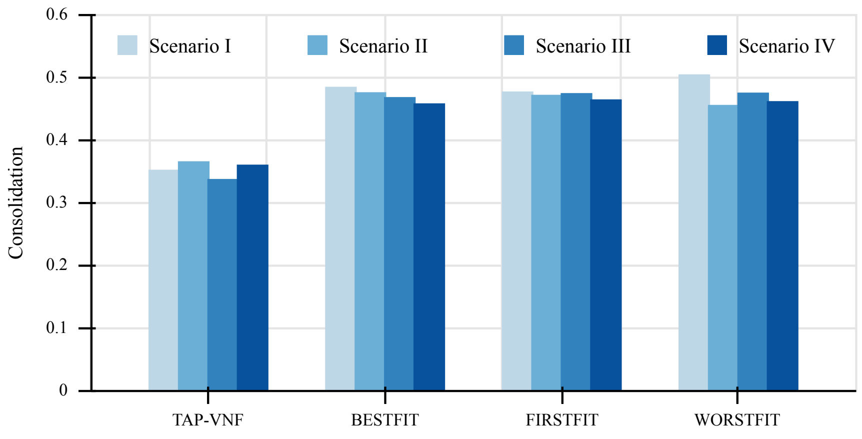
To properly understand the behavior of the presented placement strategies, we also expanded our performance analysis to several network topologies of the SNDlib [119]. This available online dataset has been used extensively to test the performance of novel algorithms and protocols in SDN/NFV research. Furthermore, it contains different classes of network graphs (regional, continental and global) with a diverse range of network sizes (from 10 to 161 nodes). As such, this dataset enables us to shed light on the performance of the proposed TAP-VNF algorithm



(a) Atlanta



(b) Janos_US

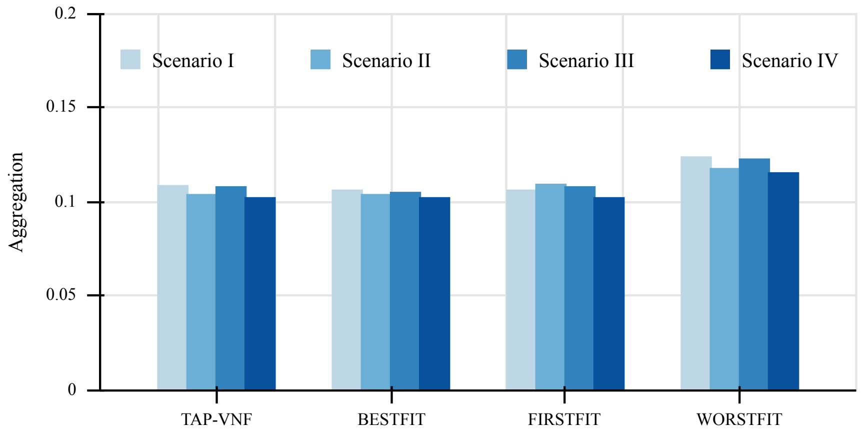


(c) Pioro

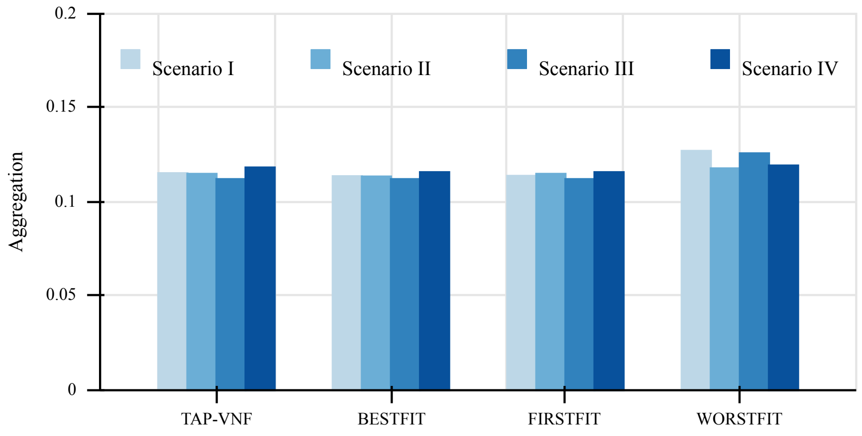
Fig. 6.10: Consolidation of Placement Strategies across Topologies of the SNDlib Dataset

in terms of the presented evaluation metrics across a wide range of network topologies.

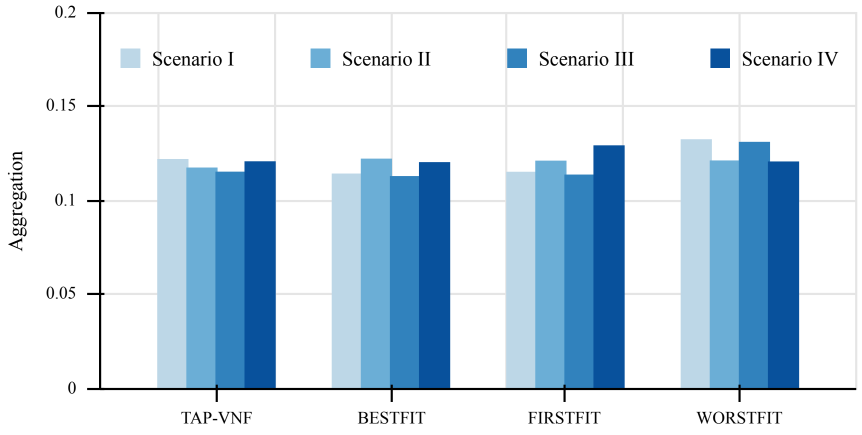
Fig. 6.10 and Fig. 6.11 present consolidation and aggregation ratios of the proposed placement strategies in different network topologies. For these evaluations we considered three topologies representative of different network scales and connectivity degrees, namely Atlanta (15 nodes,



(a) Atlanta



(b) Janos_US



(c) Pioro

Fig. 6.11: Aggregation of Placement Strategies across Topologies of the SNDlib Dataset

44 links), Janos_US (26 nodes, 84 links) and Pioro (40 nodes, 89 links). In the last case, we limited the maximum number of considered paths to $k = 500$. The consolidation and aggregation ratios were measured at 100% resource utilization and presented as a function of the SFC profiles for different allocation strategies.

Fig. 6.10 shows the consolidation ratios of the TAP-VNF algorithm while the algorithm attempts to allocate the requested VNF instances over the described topologies. The requested SFCs were generated according to all scenarios presented in Table 6.3. It can be observed that similar consolidation values are achieved for each of the proposed scenario across network topologies with different network sizes and connectivity degrees. This behavior reveals that consolidation of the proposed TAP-VNF algorithm is irrespective of the network topology. This result is promising because it enables network operators to design placement strategies that consolidate more requested VNF instances without significant variations regarding different network topologies.

Fig 6.11 shows the aggregation achieved by the online placement approach (i.e. the TAP-VNF algorithm) for the same three network topologies. Similar to the consolidation metric, the aggregation of proposed placement strategies is not strongly influenced by the considered topologies. For this reason, it may also be confirmed that the proposed TAP-VNF algorithm is not dependent on the network topology. This behavior is critical for resource allocation in cloud-based environments where the network infrastructure can be dynamically generated to ensure the requirements of the supported network applications are met.

6.6 Conclusions

In this chapter we addressed the VNF placement problem in software-defined and virtualized networks. This problem was investigated through different approaches. First, we presented a generic model capable of finding the optimal VNF placement for offline scenarios. Then, in order to address dynamic scenarios, we proposed a heuristic algorithm that is based on efficient solutions of the bin packing problem. To evaluate the performance of the presented solutions, we also identified two metrics to assess the resource occupancy efficiency of the proposed methods. Experimental simulations show that the heuristic approach achieves near-optimal solutions with smaller optimality gaps for the aggregation metric (with differences of under 8%). Moreover, obtained results confirm that the proposed online algorithm outperforms the traditional bin packing strategies (i.e. BestFit, FirstFit and WorstFit) in terms of consolidation ratios, making it a more suitable solution for the VNF placement problem. Additionally, significantly shorter computation times (compared to the ILP model) are achieved by the TAP-VNF algorithm.

Chapter 7

Conclusion

The drastic increase in Internet services, such as video on demand, big data, server virtualization and cloud services, is one of the trends driving the networking industry to change traditional network architectures. To overcome the ever-increasing demands of these new services, network operators require emerging solutions to effectively manage their network resources using more flexible and dynamic schemes.

Many challenges must be met and tasks accomplished in order to successfully deploy a flexible and configurable software-defined and virtualized network, including an automatic topology discovery protocol, a reliable self-healing framework and a dynamic VNF placement strategy. For these purposes, this thesis contributes to leveraging network programmability in three ways. First, it offers a novel protocol for topology discovery in SDN environments with multi-domains and in-band control that does not require prior controller knowledge of the network or specific network configurations from the forwarding devices. Second, it introduces a fault-tolerant mechanism for the maintenance of a global network view and control plane survivability through autonomic principles applied in self-healing SDN environments. Third, it provides a fast and efficient strategy for dynamically allocating chained VNFs, reducing the number of required instances and the delay between deployed functions.

In essence, throughout this thesis we have provided several tools to address current issues associated with topology discovery, self-healing and VNF placement in software-defined and virtualized networks. All of the contributions presented in this document are original proposals that were innovatively conceived and developed. This chapter summarizes the research work and contributions of this thesis in order to assess the fulfillment of research objectives. In addition, we outline some perspectives for future investigations.

7.1 Research Contributions

A key observation identified by this thesis is that existing topology discovery mechanisms in SDN require the establishment of previous communication between the controller and the switches in order to set specific network configurations (e.g. forwarding devices must know the controllers' IP addresses or the controllers must be aware of the forwarding devices' identifiers and capabilities), restricting their suitability and flexibility. Thus, as the first contribution in Chapter 4 this thesis presents an efficient layer 2 protocol for the topology discovery problem in SDN environments that is not bound by these restrictive assumptions. This approach, which is compatible with SDN scenarios with multiple domains and in-band control traffic, allows reducing the time and number of packets required to obtain the network graph.

In Chapter 5 we provide our second research contribution. Motivated by the benefits of the ANM paradigm, we focused on the use of self-healing properties to boost SDN robustness. For this purpose, we proposed a novel mechanism enabling fully autonomous real-time management of SDN-enabled networks in the event of failures. This approach, unlike traditional schemes, is not based on proactively configuring multiple (and memory-intensive) backup paths in each switch or performing a reactive and time-consuming routing computation at the controller level. Instead, the control paths are quickly recovered by local switch actions and subsequently optimized by global controller knowledge. By exploiting the use of this self-healing framework, an autonomous and optimized control plane recovery can be achieved, suitable for supporting the performance requirements of large-scale SDN deployments.

Finally, in Chapter 6 we addressed our final research problem, which was related to VNF placement and chaining over a software-defined and virtualized infrastructure. In this chapter we studied VNF placement based on SFC requirements, considering both static (offline) and dynamic (online) scenarios. The proposed heuristic approach makes use of the topology knowledge available at the SDN controller followed by bin packing techniques to provide a low-complexity strategy. It is designed to be suitable for real-time chained VNF allocation as well as effective in reducing the number of required instances and the delay between deployed functions. In addition, our proposal contributes to existing literature by providing two new metrics for assessing the performance of VNF placement solutions. By using these metrics, we were able to examine the implications of the network topology and requested VNFs on the VNF allocation in greater depth.

7.2 Room for Improvement

Although the proposed approaches have shown high suitability for addressing topology discovery, fault management and VNF allocation problems in software-defined and virtualized networks, there are a number of open problems that may be considered for further research.

Regarding the optimality in terms of delay, the control paths established by eTDP in Chapter 4 can be tuned in scenarios of multi-domain with several SDN controllers. Although optimal path delays between each network device and one SDN controller are guaranteed by eTDP, for several SDN controllers a federated control plane communication is required. This means that every SDN controller in the network must execute the proposed mechanism simultaneously. However, it is important to emphasize that optimal path delays are not critical for the proper operation of the eTDP mechanism.

In Chapter 5, the autonomic responses of the network devices can be enhanced using machine learning-based algorithms towards network anticipation. This functionality relates to the utilization of system knowledge to anticipate the future state of the network. In this way, forwarding devices not will only adopt a survivability strategy that guarantees optimal control path delays to the SDN controllers but will also leverage the SDN control plane to collect live load information and avoid congested links and overloaded switches. This approach improves the predictability in SDN while enabling the reestablishment of the control tree connectivity, taking into account the parameters of the current network state.

Finally, although our dynamic VNF placement algorithm addresses resource allocation as an online approach (which is more realistic in NFV-based network architectures) it does not consider the possibility of remapping or recomposing one or more SFC requests to improve allocation performance. It should be noted that SFC requests may change over time, i.e. new VNFs may be added or deleted from an SFC, meaning that recomposition, remapping and rescheduling may be necessary for such scenarios.

Publications

International Journals

- **L. Ochoa-Aday**, C. Cervelló-Pastor, A. Fernández-Fernández and P. Grosso, "An Online Algorithm for Dynamic NFV Placement in Cloud-Based Autonomous Response Networks," *Symmetry*, vol. 10, no. 5, pp. 163:1—163:18, May 2018.
- **L. Ochoa-Aday**, C. Cervelló-Pastor and A. Fernández-Fernández, "Self-healing Topology Discovery Protocol for Software Defined Networks," *IEEE Communications Letters*, vol. 22, no. 5, pp. 1070–1073, May 2018.
- **L. Ochoa-Aday**, C. Cervelló-Pastor and A. Fernández-Fernández, "Discovering the Network Topology: An Efficient Approach for SDN," *Advances in Distributed Computing and Artificial Intelligence Journal (ADCAIJ)*, vol. 5, no. 2, pp. 101—108, Nov. 2016.
- A. Fernández-Fernández, C. Cervelló-Pastor and **L. Ochoa-Aday**, "Energy Efficiency and Network Performance: A Reality Check in SDN-Based 5G Systems," *Energies*, vol. 10, no. 12, pp. 2132:1–2132:27, Dec. 2017.
- A. Fernández-Fernández, C. Cervelló-Pastor and **L. Ochoa-Aday**, "A Multi-Objective Routing Strategy for QoS and Energy Awareness in Software-Defined Networks," *IEEE Communications Letters*, vol. 21, no. 11, pp. 2416–2419, Nov. 2017.
- A. Fernández-Fernández, C. Cervelló-Pastor and **L. Ochoa-Aday**, "Energy-Aware Routing in Multiple Domains Software-Defined Networks," *Advances in Distributed Computing and Artificial Intelligence Journal (ADCAIJ)*, vol. 5, no. 3, pp. 13—19, Nov. 2016.

Technical Report

- **L. Ochoa-Aday**, C. Cervelló-Pastor and A. Fernández-Fernández, "Current Trends of Topology Discovery in OpenFlow-based Software Defined Networks," *UPCommons*, Sept. 2015. [Online]. Available: https://upcommons.upc.edu/handle/2117/77672.

International Conferences

- **L. Ochoa-Aday**, C. Cervelló-Pastor and A. Fernández-Fernández, "A Distributed Algorithm for Topology Discovery in Software-Defined Networks," in *Trends in Practical Applications of Scalable Multi-Agent Systems, the PAAMS Collection, ser. Advances in Intelligent Systems and Computing.* Cham: Springer International Publishing, June 2016, vol. 473, pp. 363—367.

- A. Fernández-Fernández, C. Cervelló-Pastor, **L. Ochoa-Aday** and **P. Grosso**, "An Online Power-Aware Routing in SDN with Congestion-Avoidance Traffic Reallocation," in *Proc. 17th IFIP-TC6 International Networking Conference (NETWORKING'18)*, Zurich, Switzerland, May. 2018, to be published.

- A. Fernández-Fernández, C. Cervelló-Pastor and **L. Ochoa-Aday**, "Achieving Energy Efficiency: An Energy-Aware Approach in SDN," in *Proc. of the 59th IEEE Conference on Global Communications (GLOBECOM'16)*, Washington DC, USA, Dec. 2016, pp. 1–7.

- A. Fernández-Fernández, C. Cervelló-Pastor and **L. Ochoa-Aday**, "Improved Energy-Aware Routing Algorithm in Software-Defined Networks," in *Proc. of the 41st IEEE Conference on Local Computer Networks (LCN'16)*, Dubai, UAE, Nov. 2016, pp. 196–199.

- A. Fernández-Fernández, C. Cervelló-Pastor and **L. Ochoa-Aday**, "A Distributed Energy-Aware Routing Algorithm in Software-Defined Networks," in *Advances in Intelligent Systems and Computing. Trends in Practical Applications of Scalable Multi-Agent Systems, the PAAMS Collection.* Springer International Publishing, June 2016, vol. 473, pp. 369–373.

National Conference

- A. Fernández-Fernández, C. Cervelló-Pastor and **L. Ochoa-Aday**, "Evaluating the Impact of Energy-Aware Routing on Software-Defined Networking Performance," in *Proc. of the*

XIII Jornadas de Ingeniería Telemática (JITEL'17), Valencia, Spain, Sep. 2017, pp. 241–248.

References

[1] "Cisco Visual Networking Index: Forecast and Methodology, 2016–2021," White Paper, Cisco Systems, Sept. 2017. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[2] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[4] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-deployed Software Defined WAN," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, Oct. 2013.

[6] T. Bakhshi, "State of the Art and Recent Research Advances in Software Defined Networking," *Wireless Communications and Mobile Computing*, vol. 2017, no. 4, pp. 1–35, Jan. 2017.

[7] "Autonomic Computing: IBM's Perspective on the State of Information Technology," White Paper, IBM Press, 2001. [Online]. Available: https://www.bibsonomy.org/bibtex/292d2eb8c354a1241e18416471572758c/neilernst

[8] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[9] "An architectural Blueprint for Autonomic Computing," White Paper, IBM Press, June 2006. [Online]. Available: https://pdfs.semanticscholar.org/0e99/837d9b1e70bb35d516e32ecfc345cd30e795.pdf

[10] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A Survey of Autonomic Communications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 2, pp. 223–259, Dec. 2006.

[11] M. C. Huebscher and J. A. McCann, "A Survey of Autonomic Computing—Degrees, Models, and Applications," *ACM Computing Surveys (CSUR)*, vol. 40, no. 3, pp. 1–28, Aug. 2008.

[12] B. Jennings, S. V. D. Meer, S. Balasubramaniam, D. Botvich, M. O. Foghlu, W. Donnelly, and J. Strassner, "Towards Autonomic Management of Communications Networks," *IEEE Communications Magazine*, vol. 45, no. 10, pp. 112–121, Oct. 2007.

[13] R. Boutaba, J. Martin-Flatin, J. Hellerstein, R. Katz, G. Pavlou, and C.-T. Lea, "Recent Advances in Autonomic Communications (Guest Editorial)," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 1, pp. 1–3, Jan. 2010.

[14] Z. Movahedi, M. Ayari, R. Langar, and G. Pujolle, "A Survey of Autonomic Network Architectures and Evaluation Criteria," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 2, pp. 464–490, May 2011.

[15] S. Kuklinski and P. Chemouil, "Network Management Challenges in Software-Defined Networks (Invited Paper)," *IEICE Transactions on Communications*, vol. E97-B, no. 1, pp. 2–9, Jan. 2014.

[16] G. Poulios, K. Tsagkaris, P. Demestichas, A. Tall, Z. Altman, and C. Destré, "Autonomics and SDN for Self-Organizing Networks," in *Proc. of the 11th International Symposium on Wireless Communications Systems (ISWCS)*, Barcelona, Spain, Jan. 2014, pp. 830–835.

[17] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, Sept. 2016.

[18] J. Garay, J. Matias, J. Unzilla, and E. Jacob, "Service Description in the NFV Revolution: Trends, Challenges and a Way Forward," *IEEE Communications Magazine*, vol. 54, no. 3, pp. 68–74, Mar. 2016.

[19] R. Jain and S. Paul, "Network Virtualization and Software Defined Networking for Cloud Computing: A Survey," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 24–31, Nov. 2013.

[20] Y. Li and M. Chen, "Software-Defined Network Function Virtualization: A Survey," *IEEE Access*, vol. 3, pp. 2542–2553, Oct. 2015.

[21] T. Wood, K. K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang, “Toward a Software-Based Network: Integrating Software Defined Networking and Network Function Virtualization,” *IEEE Access*, vol. 29, no. 3, pp. 36–41, June 2015.

[22] E. Hernandez-Valencia, S. Izzo, and B. Polonsky, “How Will NFV/SDN Transform Service Provider OPEX?” *IEEE Network*, vol. 29, no. 3, pp. 60–67, May 2015.

[23] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, “Current Trends of Topology Discovery in OpenFlow-based Software Defined Networks,” *UPCommons*, Sept. 2015. [Online]. Available: https://upcommons.upc.edu/handle/2117/77672

[24] ——, “A Distributed Algorithm for Topology Discovery in Software-Defined Networks,” in *Trends in Practical Applications of Scalable Multi-Agent Systems, the PAAMS Collection*, ser. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, June 2016, vol. 473, pp. 363–367.

[25] ——, “Discovering the Network Topology: An Efficient Approach for SDN,” *Advances in Distributed Computing and Artificial Intelligence Journal (ADCAIJ)*, vol. 5, no. 2, pp. 101–108, Nov. 2016.

[26] ——, “Self-healing Topology Discovery Protocol for Software Defined Networks,” *IEEE Communications Letters*, vol. 22, no. 5, pp. 1070–1073, May 2018.

[27] L. Ochoa-Aday, C. Cervelló-Pastor, A. Fernández-Fernández, and P. Grosso, “An Online Algorithm for Dynamic NFV Placement in Cloud-Based Autonomous Response Networks,” *Symmetry*, vol. 10, no. 5, pp. 163:1–163:18, May 2018.

[28] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, “Are We Ready for SDN? Implementation Challenges for Software-Defined Networks,” *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, July 2013.

[29] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, “Advancing Software-Defined Networks: A Survey,” *IEEE Access*, vol. 5, pp. 25 487–25 526, 2017.

[30] A. Lara, A. Kolasani, and B. Ramamurthy, “Network Innovation using OpenFlow: A Survey,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, First Quarter 2014.

[31] 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,” Internet Requests for Comments, RFC 5810, Mar. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5810.txt

[32] “OpFlex: An Open Source Approach,” White Paper, Cisco Systems, Mar. 2014. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731304.html

[33] H. Song, "Protocol-Oblivious Forwarding: Unleash the Power of SDN Through a Future-Proof Forwarding Plane," in *Proc. of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, Hong Kong, China, Aug. 2013, pp. 127–132.

[34] S. Li, D. Hu, W. Fang, S. Ma, C. Chen, H. Huang, and Z. Zhu, "Protocol Oblivious Forwarding (POF): Software-Defined Networking with Enhanced Programmability," *IEEE Network*, vol. 31, no. 2, pp. 58–66, Mar. 2017.

[35] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, Feb. 2014.

[36] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN Control: Survey, Taxonomy and Challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 333–354, Dec. 2017.

[37] W. Braun and M. Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, May 2014.

[38] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Automatic Bootstrapping of OpenFlow Networks," in *Proc. of the 19th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN)*, Brussels, Belgium, Apr. 2013, pp. 1–6.

[39] L. Schiff, S. Schmid, and P. Kuznetsov, "In-Band Synchronization for Distributed SDN Control Planes," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 1, pp. 37–43, Jan. 2016.

[40] M. Aslan and A. Matrawy, "On the Impact of Network State Collection on the Performance of SDN Applications," *IEEE Communications Letters*, vol. 20, no. 1, pp. 5–8, Jan. 2016.

[41] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically Centralized? State Distribution Trade-offs in Software Defined Networks," in *Proc. of the First Workshop on Hot Topics in Software Defined Networks*, Helsinki, Finland, Aug. 2012, pp. 1–6.

[42] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software-Defined Networking: State of the Art and Research Challenges," *CoRR*, vol. abs/1406.0124, 2014. [Online]. Available: http://arxiv.org/abs/1406.0124

[43] S. Khan, A. Gani, A. A. Wahab, M. Guizani, and M. K. Khan, "Topology Discovery in Software Defined Networks: Threats, Taxonomy, and State-of-the-Art," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 303–324, Firstquarter 2017.

[44] T. Alharbi, M. Portmann, and F. Pakzad, "The (in)security of Topology Discovery in Software Defined Networks," in *Proc. of the IEEE 40th Conference on Local Computer Networks (LCN)*, Oct. 2015, pp. 502–505.

[45] "OpenFlow Discovery Protocol and Link Layer Discovery Protocol," GENI Wiki, Global Environment for Network Innovations (GENI). [Online]. Available: http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol/

[46] "IEEE Standard for Local and Metropolitan Area Networks– Station and Media Access Control Connectivity Discovery," *IEEE Std 802.1AB-2016 (Revision of IEEE Std 802.1AB-2009)*, pp. 1–146, Mar. 2016.

[47] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *Proc. of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX)*, Monterey, California, Oct. 2010, pp. 19:1–19:6.

[48] A. Azzouni, N. T. M. Trang, R. Boutaba, and G. Pujolle, "Limitations of OpenFlow Topology Discovery Protocol," *CoRR*, vol. abs/1705.00706 [cs], 2017. [Online]. Available: http://arxiv.org/abs/1705.00706

[49] N. Agoulmine, S. Balasubramaniam, D. Botvitch, J. Strassner, E. Lehtihet, and W. Donnelly, "Challenges for Autonomic Network Management," in *Proc. of the First IEEE International Workshop on Modelling Autonomic Communications Environments (MACE)*, Dublin, Ireland, Oct. 2006.

[50] W. Jiang, M. Strufe, and H. Schotten, "Autonomic Network Management for Software-Defined and Virtualized 5G Systems," in *Proc. of the 23th European Wireless Conference*, Dresden, Germany, May 2017, pp. 1–6.

[51] N. Samaan and A. Karmouch, "Towards Autonomic Network Management: an Analysis of Current and Future Research Directions," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 3, pp. 22–36, 3rd Quarter 2009.

[52] S. Neti and H. A. Muller, "Quality Criteria and an Analysis Framework for Self-Healing Systems," in *Proc. of the International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'2007)*, Minneapolis, MN, USA, May 2007, pp. 1–10.

[53] H. Psaier and S. Dustdar, "A Survey on Self-healing Systems: Approaches and Systems," *Computing*, vol. 91, no. 1, pp. 43–73, Jan. 2011.

[54] D. Ghosh, R. Sharman, H. Raghav Rao, and S. Upadhyaya, "Self-healing Systems – Survey and Synthesis," *Decision Support Systems*, vol. 42, no. 4, pp. 2164–2185, Jan. 2007.

[55] I. Al-Oqily, S. Bani-Mohammad, B. Subaih, and J. J. Alshaer, "A Survey for Self-healing Architectures and Algorithms," in *Proc. of the International Multi-Conference on Systems, Signals Devices (SSD'2012)*, Chemnitz, Germany, Mar. 2012, pp. 1–5.

[56] J.-P. Vasseur, M. Pickavet, and P. Demeester, *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[57] P. C. d. R. Fonseca and E. S. Mota, "A Survey on Fault Management in Software-Defined Networks," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2284–2321, Fourthquarter 2017.

[58] Open Networking Foundation, "OpenFlow Switch Specification v1.4.0," Technical Specification, Oct. 2013. [Online]. Available: https://www.opennetworking.org/

[59] N. M. Sahri and K. Okamura, "Fast Failover Mechanism for Software Defined Networking: OpenFlow Based," in *Proc. of the Ninth International Conference on Future Internet Technologies (CFI'2014)*, Tokyo, Japan, June 2014, pp. 16:1–16:2.

[60] Y. Lin, H. Teng, C. Hsu, C. Liao, and Y. Lai, "Fast Failover and Switchover for Link Failures and Congestion in Software Defined Networks," in *Proc. of the IEEE International Conference on Communications (ICC'2016)*, Kuala Lumpur, Malaysia, May 2016, pp. 1–6.

[61] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "In-band Control, Queuing, and Failure Recovery Functionalities for OpenFlow," *IEEE Network*, vol. 30, no. 1, pp. 106–112, Jan. 2016.

[62] E. G. Pereira, R. Pereira, and A. Taleb-Bendiab, "Performance Evaluation for Self-healing Distributed Services and Fault Detection Mechanisms," *Journal of Computer and System Sciences*, vol. 72, no. 7, pp. 1172–1182, Nov. 2006.

[63] N. M. K. Chowdhury and R. Boutaba, "Network Virtualization: State of the Art and Research Challenges," *IEEE Communications Magazine*, vol. 47, no. 7, pp. 20–26, July 2009.

[64] ——, "A Survey of Network Virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, Apr. 2010.

[65] "Network Functions Virtualisation – An Introduction, Benefits, Enablers, Challenges & Call for Action," NFV White Paper, ETSI, pp. 1–21, Oct. 2012. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[66] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network Function Virtualization: Challenges and Opportunities for Innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[67] ETSI, "Network Functions Virtualisation (NFV) – Architectural Framework," ETSI GS NFV 002 V1.2.1, pp. 1–21, Dec. 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.02.01_60/gs_nfv002v010201p.pdf

[68] M. S. Bonfim, K. L. Dias, and S. F. L. Fernandes, “Integrated NFV/SDN Architectures: A Systematic Literature Review,” *CoRR*, vol. abs/1801.01516, 2018. [Online]. Available: http://arxiv.org/abs/1801.01516

[69] J. G. Herrera and J. F. Botero, “Resource Allocation in NFV: A Comprehensive Survey,” *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, Sept. 2016.

[70] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, “A Survey on Service Function Chaining,” *Journal of Network and Computer Applications*, vol. 75, no. C, pp. 138–155, Nov. 2016.

[71] T. Wen, H. Yu, G. Sun, and L. Liu, “Network Function Consolidation in Service Function Chaining Orchestration,” in *Proc. of the IEEE International Conference on Communications (ICC’2016)*, Kuala Lumpur, Malaysia, May 2016, pp. 1–6.

[72] J. F. Riera, E. Escalona, J. Batallé, E. Grasa, and J. A. García-Espín, “Virtual Network Function Scheduling: Concept and Challenges,” in *Proc. of the International Conference on Smart Communications in Network Technologies (SaCoNeT)*, Vilanova i la Geltru, Spain, June 2014, pp. 1–5.

[73] Open Networking Foundation, “OpenFlow Switch Specification v1.5.1,” Technical Specification, Apr. 2015. [Online]. Available: https://www.opennetworking.org/

[74] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, “Efficient Topology Discovery in Software Defined Networks,” in *Proc. of the 8th International Conference on Signal Processing and Communication Systems (ICSPCS’2014)*, Gold Coast, QLD, Australia, Dec. 2014, pp. 1–8.

[75] ——, “Efficient Topology Discovery in OpenFlow-based Software Defined Networks,” *Computer Communications*, vol. 77, pp. 52–61, Mar. 2016.

[76] POX Network Software Platform. [Online]. Available: https://noxrepo.github.io/pox-doc/html/

[77] Mininet – An Instant Virtual Network on Your Laptop (or Other PC) (version 2.3.0d4). [Online]. Available: http://mininet.org/

[78] D. Hasan and M. Othman, “Efficient Topology Discovery in Software Defined Networks: Revisited,” *Procedia Computer Science*, vol. 116, pp. 539–547, 2017.

[79] E. Rojas, J. Alvarez-Horcajo, I. Martinez-Yelmo, J. A. Carral, and J. M. Arco, “TEDP: An Enhanced Topology Discovery Service for Software-Defined Networking,” *IEEE Communications Letters*, vol. 22, no. 8, pp. 1540–1543, Aug. 2018.

[80] A. Azzouni, R. Boutaba, N. T. M. Trang, and G. Pujolle, “sOFTDP: Secure and Efficient Topology Discovery Protocol for SDN,” *CoRR*, vol. abs/1705.04527 [cs], 2017. [Online]. Available: http://arxiv.org/abs/1705.04527

[81] Floodlight – Open Source Software for Building Software-Defined Networks (version 1.2). [Online]. Available: http://www.projectfloodlight.org/floodlight/

[82] J. J. Mambretti, F. I. Yeh, J. H. Chen, P. Tsai, J. Hu, C. Yang, W. Huang, T. Liu, and S. Lin, "Design and Implementation of an Automatic Network Topology Discovery System for the Future Internet Across Different Domains," in *Proc. of the 26th International Conference on Advanced Information Networking and Applications Workshops (WAINA'2012)*, Fukuoka-shi, Japan, Mar. 2012, pp. 903–908.

[83] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, July 2008.

[84] G. Tarnaras, E. Haleplidis, and S. Denazis, "SDN and ForCES Based Optimal Network Topology Discovery," in *Proc. of the First IEEE Conference on Network Softwarization (NetSoft'2015)*, London, UK, Apr. 2015, pp. 1–6.

[85] G. Tarnaras, F. Athanasiou, and S. Denazis, "Efficient Topology Discovery Algorithm for Software-Defined Networks," *IET Networks*, vol. 6, no. 6, pp. 157–161, Nov. 2017.

[86] 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, May 2015.

[87] J. S. Choi, S. Kang, and Y. Lee, "Design and Evaluation of a PCEP-based Topology Discovery Protocol for Stateful PCE," *Optical Switching and Networking*, vol. 26, pp. 39–47, Nov. 2017.

[88] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang, "Rosemary: A Robust, Secure, and High-performance Network Operating System," in *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Scottsdale, Arizona, USA, Nov. 2014, pp. 78–89.

[89] A. Capone, C. Cascone, A. Q. T. Nguyen, and B. Sansò, "Detour Planning for Fast and Reliable Failure Recovery in SDN with OpenState," in *Proc. of the 11th International Conference on the Design of Reliable Communication Networks (DRCN)*, Kansas City, MO, USA, Mar. 2015, pp. 25–32.

[90] C. Cascone, L. Pollini, D. Sanvito, and A. Capone, "Traffic Management Applications for Stateful SDN Data Plane," in *Proc. of the Fourth European Workshop on Software-Defined Networks*, Bilbao, Spain, Oct. 2015, pp. 85–90.

[91] P. Thorat, S. Jeon, and H. Choo, "Enhanced Local Detouring Mechanisms for Rapid and Lightweight Failure Recovery in OpenFlow Networks," *Computer Communications*, vol. 108, pp. 78–93, Aug. 2017.

[92] P. Thorat, S. M. Raza, D. S. Kim, and H. Choo, “Rapid Recovery from Link Failures in Software-Defined Networks,” *Journal of Communications and Networks*, vol. 19, no. 6, pp. 648–665, Dec. 2017.

[93] N. L. M. v. Adrichem, B. J. v. Asten, and F. A. Kuipers, “Fast Recovery in Software-Defined Networks,” in *Proc. of the 3rd European Workshop on Software-Defined Networks*, Budapest, Hungary, Sept. 2014, pp. 61–66.

[94] D. Katz and D. Ward, “Bidirectional Forwarding Detection (BFD),” Internet Requests for Comments, RFC 5880, June 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5880.txt

[95] B. Raeisi and A. Giorgetti, “Software-based Fast Failure Recovery in Load Balanced SDN-based Datacenter Networks,” in *Proc. of the 6th International Conference on Information Communication and Management (ICICM)*, Hatfield, UK, Oct. 2016, pp. 95–99.

[96] P. Thorat, S. M. Raza, D. T. Nguyen, G. Im, H. Choo, and D. S. Kim, “Optimized Self-healing Framework for Software Defined Networks,” in *Proc. of the 9th International Conference on Ubiquitous Information Management and Communication (IMCOM)*, Bali, Indonesia, Jan. 2015, pp. 7:1–7:6.

[97] J. M. Sánchez, I. G. B. Yahia, N. Crespi, T. M. Rasheed, and D. Siracusa, “Softwarized 5G Networks Resiliency with Self-Healing,” *CoRR*, vol. abs/1507.02951, 2015. [Online]. Available: http://arxiv.org/abs/1507.02951

[98] J. M. Sánchez, I. G. B. Yahia, and N. Crespi, “POSTER: Self-Healing Mechanisms for Software-Defined Networks,” *CoRR*, vol. abs/1507.02952, 2015. [Online]. Available: http://arxiv.org/abs/1507.02952

[99] ——, “THESARD: On the Road to Resilience in Software-Defined Networking Through Self-diagnosis,” in *Proc. of the IEEE NetSoft Conference and Workshops (NetSoft)*, Seoul, South Korea, June 2016, pp. 351–352.

[100] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, “Near Optimal Placement of Virtual Network Functions,” in *Proc. of the IEEE Conference on Computer Communications (INFOCOM)*, Kowloon, Hong Kong, May 2015, pp. 1346–1354.

[101] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, “An Approach for Service Function Chain Routing and Virtual Function Network Instance Migration in Network Function Virtualization Architectures,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2008–2025, Aug. 2017.

[102] H. Li, P. Li, S. Guo, and A. Nayak, “Byzantine-Resilient Secure Software-Defined Networks with Multiple Controllers in Cloud,” *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 436–447, Sept. 2014.

[103] S. Mehraghdam, M. Keller, and H. Karl, “Specifying and Placing Chains of Virtual Network Functions,” in *Proc. of the 3rd IEEE International Conference on Cloud Networking (CloudNet)*, Luxembourg City, Luxembourg, Oct. 2014, pp. 7–13.

[104] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, “Piecing Together the NFV Provisioning Puzzle: Efficient Placement and Chaining of Virtual Network Functions,” in *Proc. of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Ottawa, ON, Canada, May 2015, pp. 98–106.

[105] M. C. Luizelli, W. L. da Costa Cordeiro, L. S. Buriol, and L. P. Gaspary, “A Fix-and-Optimize Approach for Efficient and Large Scale Virtual Network Function Placement and Chaining,” *Computer Communications*, vol. 102, no. C, pp. 67–77, Apr. 2017.

[106] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, “On Orchestrating Virtual Network Functions,” in *Proc. of the 11th International Conference on Network and Service Management (CNSM)*, Barcelona, Spain, Nov. 2015, pp. 50–56.

[107] T. Li, H. Zhou, and H. Luo, “A New Method for Providing Network Services: Service Function Chain,” *Optical Switching and Networking*, vol. 26, pp. 60–68, Nov. 2017.

[108] C. H. Hsieh, J. W. Chang, C. Chen, and S. H. Lu, “Network-Aware Service Function Chaining Placement in a Data Center,” in *Proc. of the 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Kanazawa, Japan, Oct. 2016, pp. 1–6.

[109] D. Erickson, “The Beacon OpenFlow Controller,” in *Proc. of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, Hong Kong, China, Aug. 2013, pp. 13–18.

[110] J. Medved, R. Varga, A. Tkacik, and K. Gray, “OpenDaylight: Towards a Model-Driven SDN Controller Architecture,” in *Proc. of the IEEE 15th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, Sydney, NSW, Australia, June 2014, pp. 1–6.

[111] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O’Connor, P. Radoslavov, W. Snow *et al.*, “ONOS: Towards an Open, Distributed SDN OS,” in *Proc. of the Third ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, Chicago, IL, USA, Aug. 2014, pp. 1–6.

[112] ETSI, “An Architectural Reference Model for Autonomic Networking, Cognitive Networking and Self-Management,” ETSI GS AFI 002 V1.1.1, pp. 1–167, Apr. 2013. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf

[113] R. Callon, "Use of OSI IS-IS for Routing in TCP/IP and Dual Environments," Internet Requests for Comments, RFC 1195, Dec. 1990. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1195.txt

[114] A. DeKok and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions," Internet Requests for Comments, RFC 6929, Apr. 2013. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6929.txt

[115] A. C. Risdianto, J.-S. Shin, and J. W. Kim, "Deployment and Evaluation of Software-Defined Inter-Connections for Multi-domain Federated SDN-Cloud," in *Proc. of the 11th International Conference on Future Internet Technologies*, New York, USA, June 2016, pp. 118–121.

[116] OMNeT++ – Discrete Event Simulator (version 5.4.1). [Online]. Available: https : //www.omnetpp.org/

[117] G. Anggono and T. Moors, "A Flow-Level Extension to OMNeT++ for Long Simulations of Large Networks," *IEEE Communications Letters*, vol. 21, no. 3, pp. 496–499, Mar. 2017.

[118] A. W. Malik, K. Bilal, S. U. Malik, Z. Anwar, K. Aziz, D. Kliazovich, N. Ghani, S. U. Khan, and R. Buyya, "CloudNetSim++: A GUI Based Framework for Modeling and Simulation of Data Centers in OMNeT++," *IEEE Transactions on Services Computing*, vol. 10, no. 4, pp. 506–519, July 2017.

[119] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0-Survivable Network Design Library," *Networks*, vol. 55, no. 3, pp. 276–286, May 2010.

[120] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[121] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and Performance Evaluation of an OpenFlow Architecture," in *Proc. of the 23rd International Teletraffic Congress (ITC)*, San Francisco, CA, USA, Sept. 2011, pp. 1–7.

[122] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting Carrier-grade Recovery Requirements," *Computer Communications*, vol. 36, no. 6, pp. 656–665, Mar. 2013.

[123] V. Jacobson, "Congestion Avoidance and Control," *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4, pp. 314–329, Aug. 1988.

[124] D. Clark, "Window and Acknowledgement Strategy in TCP," Internet Requests for Comments, RFC 813, July 1982. [Online]. Available: http://www.rfc-editor.org/rfc/rfc813.txt

[125] L. Wang, Z. Lu, X. Wen, R. Knopp, and R. Gupta, "Joint Optimization of Service Function Chaining and Resource Allocation in Network Function Virtualization," *IEEE Access*, vol. 4, pp. 8084–8094, Nov. 2016.

[126] Gurobi Optimizer (version 7.5). Gurobi Optimization. [Online]. Available: http://www.gurobi.com/