UNIVERSITY OF KASDI MERBAH OUARGLA

Faculty of New Technologies of Information and Communications

Department of Electronics and Telecommunication

Master's Thesis

Submitted in fulfillment of the requirements for the Master of Science degree in Telecommunications Systems.

SDN Network Integration with Containers and Kubernetes for Internet Service Providers

A Practical Study Conducted with Networks provided by Djezzy

BY

Selmane Fares CHERIFI

&

Ayman HAMMANI

In front of the jury members of

Dr. Ibtissam Chaib	*President*	MCB, Univ Ouargla
Prof. Mohammed Elbachir Mahdjoub	*Examination*	Prof, Univ Ouargla
Dr. Sayhia Tidjani	*Supervisor*	MAB, Univ Ouargla

Academic year: 2023/2024

شكر و عرفان

اللهم صلي على محمد و على آل محمد كما صليت على ابراهيم و آل ابراهيم و بارك على محمد كما باركت على ابراهيم و آل ابراهيم في العالمين انك حميد مجيد.

يا من لهم فضل علينا سابق   شكر لكم حق علينا اوجب.

نتوجه بخالص الشكر و الامتنان لعائلتنا التي ما فتئت و لا ملت يوما عن زرع حب العلم و طلبه داخلنا.

نسال الله ان يصوب خطى الاستاذة "سائحية التجاني" التي لم تبخل علينا بنصائحها و توجيهها المستمر. كما نعبر عن خالص امتنانا للاساتذتنا الكرام اللذين كانوا جزءا مهما في جعل هذه المسيرة الدراسية تتكلل بالنجاح

شكرنا وتقديرنا لمشرف التربص العملي "سفيان سعيدي" لمتابعته لدراستنا حتى بعد انتهاء الفترة التدريبية. وأيضًا، شكرًا لجميع أعضاء فريق جازي الذين ساعدونا في اكتساب معرفة مطولة كل من فريق الشبكات، وفريق هندسة **BSS**، ووصولاً حتى إلى فريق الأمن الشبكي.

إهــــــداء

و في ذا السرور بتلك الكرب   و هذا المقام بذاك التعب.

الحمد لله الذي ما ختم جهد و لا تم سعي إلا بفضله.

إلى والدي الكريمين اللذين كانا العون الدائم والسند الذي لا يميل.

إلى إخوتي و أخواتي الأعزاء اللذين كانوا مصدر الدعم و القوة.

إلى خلاني الأوفياء حفظكم الله و بارك لكم.

و اختص بالامتنان للعزيز حكيم شريفي و عمي الموقر عبد الرحمان شريفي كونهم أول من دفعوني لاجتياز هذه الخطوة ولم يبخلوا عني بخبرتهم ومعرفتهم في هذا المجال.

بكل امتنان، أتوجه بخالص الشكر إلى أصدقائي الأعزاء الذين كانوا دائماً سنداً لي في كل خطوات حياتي. البعيد منهم و القريب لكم كل التقدير و الامتنان شكراً لكم على دعمكم المتواصل وتشجيعكم الذي لا ينضب.

إلى الشفاه الذابلة و الأجسام الذاوية التي قضت عذابا تحت سطوة الفقر و الحرب و اليتم ، نرفع صلاة لأرواحهم الطاهرة و نكفكف دموع مآسيهم الغائرة في من بقي منهم بيننا.

اللهم انصر إخواننا في فلسطين و قوي سواعدهم

.

“سلمان فارس شريفي”

إهــــــــداء

يسعدني أن أتمكن من التعبير عن امتناني لأحب الناس إلي.

أهدي هذا العمل من كل قلبي إلى والدتي العزيزة فوزية أولاد سالم، فقد جعلتني من أنا اليوم ولم تبخل بأي شيء لتتأكد من أنني محبوب وآمن ومدعوم وموجه، والأهم من ذلك كله، ناجح، ملهمتي التي لم تفشل أبدًا في تشجيعي معنويًا وروحيًا وعاطفيًا...

إلى عائلتي وأصدقائي الأعزاء، جميعًا باسمه.

شكر خاص لمشرفتي، الدكتورة تيجاني، على صبرها الثمين وملاحظاتها القيمة أثناء هذا المشروع و تعليقاتها الثاقبة.

أود أن أشكر السيد سعيدي سفيان وكل موظفي شركة جازي، الذين كانت نصائحهم وصبرهم ودعمهم حاسمة في إكمال هذه الأطروحة.

وأخيرًا، إلى كل من ساعدني خلال رحلتي وجعلها مميزة ومليئة بالذكريات.

" أيمن حماني"

Abstract

The thesis explores the integration of Software-Defined Networking (SDN) with containers and Kubernetes for Internet Service Providers (ISPs). The study focuses on the practical implementation of SDN, Containers, and Kubernetes in a real-world ISP environment, highlighting the benefits and challenges of this integration. The research begins by discussing the fundamental concepts of computer networking, including the history of computer networks and the evolution of SDN. It then delves into the technical aspects of the latter, containers and Kubernetes, exploring their benefits, limitations, and potential applications.

The practical implementation of SDN, containers, and Kubernetes is demonstrated through a case study involving a collaboration with DJEZZY, an Algerian ISP. The study outlines the project plan, network modifications, and the configuration of the Legacy network, including the deployment of protocols such as IS-IS, BGP, OSPF, MPLS, and MPLS-TE.

The testing methodology, which involves utilizing tools like IPerf to assess the network's performance in terms of bandwidth, latency, and packet loss, is also presented. The results of the study highlight the potential benefits of SDN, containers, and Kubernetes in improving network efficiency, scalability, and security for ISPs.

Keywords

SDN, containers, Kubernetes, Internet Service Providers, network integration, network efficiency, scalability, security.

ملخص

تستكشف الأطروحة تكامل الشبكات المعرفة بالبرمجيات (SDN) مع الContainers وKubernetes لمقدمي خدمات الإنترنت (ISPs). تركز الدراسة على التنفيذ العملي لـــ SDN واContainers وKubernetes في بيئة مزود خدمة الإنترنت في العالم الحقيقي، مع تسليط الضوء على فوائد وتحديات هذا التكامل. يبدأ البحث بمناقشة المفاهيم الأساسية لشبكات الكمبيوتر، بما في ذلك تاريخ شبكات الكمبيوتر وتطور SDN. ثم يتعمق في الجوانب التقنية للأخيرة والContainers وKubernetes، ويستكشف فوائدها وقيودها وتطبيقاتها المحتملة.

يتم توضيح التنفيذ العملي لـــ **SDN** والحاويات و**Kubernetes** من خلال دراسة حالة تتضمن التعاون مع **DJEZZY**، مزود خدمة الإنترنت الجزائري. توضح الدراسة خطة المشروع وتعديلات الشبكة وتكوين الشبكة التقليدية، بما في ذلك نشر البروتوكولات مثل **IS-IS** و**BGP** و**OSPF** و**MPLS** و**MPLS-TE**.

يتم أيضًا عرض منهجية الاختبار، والتي تتضمن استخدام أدوات مثل **IPerf** لتقييم أداء الشبكة من حيث عرض النطاق الترددي وزمن الوصول وفقدان الحزمة. تسلط نتائج الدراسة الضوء على الفوائد المحتملة لـــ SDN **و**Containers وKubernetes في تحسين كفاءة الشبكة وقابلية التوسع والأمان لمقدمي خدمات الإنترنت.

الكلمات المفتاحية

SDN، **Containers**، **Kubernetes**، مقدمو خدمات الإنترنت، تكامل الشبكة، كفاءة الشبكة، قابلية التوسع، الأمن.

Résumé

La thèse explore l'intégration des réseaux définis par logiciel (SDN) avec des conteneurs et Kubernetes pour les fournisseurs de services Internet (FAI). L'étude se concentre sur la mise en œuvre pratique de SDN, de conteneurs et de Kubernetes dans un environnement FAI réel, soulignant les avantages et les défis de cette intégration. La recherche commence par discuter des concepts fondamentaux des réseaux informatiques, y compris l'histoire des réseaux informatiques et l'évolution du SDN. Il aborde ensuite les aspects techniques de ces derniers, des conteneurs et de Kubernetes, en explorant leurs avantages, leurs limites et leurs applications potentielles.

La mise en œuvre pratique du SDN, des conteneurs et de Kubernetes est démontrée à travers une étude de cas impliquant une collaboration avec DJEZZY, un FAI algérien. L'étude décrit le plan du projet, les modifications du réseau et la configuration du réseau traditionnel, y compris le déploiement de protocoles tels que IS-IS, BGP, OSPF, MPLS et MPLS-TE.

La méthodologie de test, qui implique l'utilisation d'outils tels qu'IPerf pour évaluer les performances du réseau en termes de bande passante, de latence et de perte de paquets, est également présentée. Les résultats de l'étude mettent en évidence les avantages potentiels du SDN, des conteneurs et de Kubernetes pour améliorer l'efficacité, l'évolutivité et la sécurité du réseau pour les FAI.

Mots clés

SDN, conteneurs, Kubernetes, fournisseurs d'accès Internet, intégration réseau, efficacité réseau, évolutivité, sécurité.

Acronyms

API: Application Programming Interface

AP: Access Point

BGP: Border Gateway Protocol

CAPEX: Capital Expenditure

CLI: Command-Line Interface

ECMP: Equal-Cost Multipath

GNS3: Graphical Network Simulator-3

IDS: Intrusion Detection System

IP: Internet Protocol

IS-IS: Intermediate System to Intermediate System

ISP: Internet Service Provider

LAN: Local Area Network

MME: Mobility Management Entity

MPLS: Multiprotocol Label Switching

NCP: Network Control Protocol

NFV: Network Functions Virtualization

ODL: OpenDaylight

OPEX: Operational Expenditure

OSPF: Open Shortest Path First

PGW: Packet Gateway

QoE:	Quality of Experience
QoS:	Quality of Service
RBAC:	Role-Based Access Control
SDN:	Software-Defined Networking
SIEM:	Security Information and Event Management
TCP:	Transmission Control Protocol
TLS:	Transport Layer Security
VLAN:	Virtual Local Area Network
VM:	Virtual Machine
WAN:	Wide Area Network

Table of Contents

LIST OF FIGURES

LIST OF TABLES

General Introduction

The rapid evolution of telecommunications and networking technologies necessitates a continual reassessment of legacy network architectures. Increasing demands from modern technologies such as 5G, IoT, and FTTH have presented challenges to Internet Service Providers (ISPs) in delivering high-performance, scalable, and cost-effective network services. Conventional network designs, characterized by rigidity and complexity, struggle to meet the growing demands for bandwidth, flexibility, and efficient management. To address these challenges, innovative approaches such as Software-Defined Networking (SDN) and containerization technologies have emerged.

Software-Defined Networking (SDN) decouples the control plane from the data plane, enabling centralized, programmable network management. This paradigm allows for dynamic resource allocation, improved network visibility, and greater operational flexibility. Concurrently, containerization, particularly through Kubernetes, provides a lightweight and efficient method to deploy and manage applications. Kubernetes automates the deployment, scaling, and operations of application containers across a cluster of machines.

This thesis explores the integration of SDN with Kubernetes to enhance ISP networks. The research develops a framework leveraging both technologies to address the limitations of Legacy networks. The study is structured as follows:

Introduction and Background (chapter 1): An overview of the current telecommunications landscape and the evolution of network technologies, identifying specific challenges faced by ISPs. The rationale for integrating SDN with Kubernetes and the primary research objectives are outlined.

Methodology (chapter 2): A detailed methodological framework for the proposed integration, including the selection of SDN controller OpenDaylight, container orchestration tools (Kubernetes), and the use of network simulation environment GNS3, The experimental setup and integration processes are described.

Results and Analysis (chapter 3): Presentation of experimental results comparing fault tolerance and latency improvements in SDN-integrated networks versus traditional architectures. The findings will signify the enhancements in network resilience and performance.

Discussion: Analysis of the experimental results and their implications for ISP network management practices. The discussion includes limitations of the study and suggestions for future research.

Conclusion: A summary of key findings, emphasizing the practical contributions of the research in advancing network management practices. The potential of SDN and Kubernetes integration to transform ISP networks is highlighted, along with recommendations for further exploration.

Through this Practical research, the thesis aims to provide valuable insights into the practical application of SDN and containerization technologies, offering a pathway to more efficient, scalable, and resilient ISP network infrastructures.

Chapter 1

Basic concepts and technical studies

1.1 Introduction:

The contemporary landscape of telecommunications is witnessing rapid transformations driven by advancements in networking technologies and the exponential growth of data traffic. Among these technologies, Software-Defined Networking (SDN) has emerged as a paradigm-shifting architecture designed to address the limitations of legacy network infrastructures. This thesis explores the integration of SDN with Containers and Kubernetes, specifically within the context of Internet Service Providers (ISPs), to enhance network management, scalability, and service delivery.

The inception of modern computer networks can be traced back to ARPANET, which introduced packet-switching techniques and laid the foundation for today's Internet. Over the decades, networking technologies have evolved to address increasing demands for bandwidth, security, and flexibility. Legacy networks, characterized by static and hardware-centric configurations, struggle to adapt to the dynamic requirements of modern applications and services.

As ISPs strive to meet the growing needs for high-speed and reliable internet services, they face significant challenges related to network management, scalability, and operational efficiency. The legacy approach to network management is often hampered by inflexible architectures and high operational costs. This thesis investigates how SDN, combined with containerization technologies such as Kubernetes, can offer a more flexible, scalable, and cost-effective solution for ISPs.

The Objectives are:

- **To analyze the limitations of legacy network infrastructures in ISPs.**
- **To explore the potential benefits of integrating SDN with containerization technologies like Kubernetes.**
- **To propose a practical implementation framework for ISPs to adopt SDN and containerization technologies effectively.**

This chapter presents an overview of the research, its context, and the organizational structure of the thesis. It addresses the evolution of network technologies and the increasing demand for efficient, scalable, and reliable network infrastructure, highlighting the limitations of traditional legacy networks and introducing Software-Defined Networking (SDN) as a promising solution. The problem statement articulates the issues with legacy networks regarding scalability, fault tolerance, and resource management, leading to the central research problem. Our primary research objectives include evaluating SDN performance, assessing fault tolerance capabilities, and exploring the integration of Kubernetes with SDN.

1.2 Computer Networks

1.2.1 Introduction

Networking is the practice of linking multiple computing devices together to enable the transmission, exchange, or sharing of data and resources between them. This can be achieved through wired connections (like cables) or wireless connections (like Wi-Fi).

Networks can be classified based on their geographic location and the protocols they use to communicate. For example, a Local Area Network (LAN) connects computers in a defined physical space, like an office building, while a Wide Area Network (WAN) can connect computers across long distances or even continents. The internet is the largest example of a WAN, connecting billions of computers worldwide.

1.2.2 History of Computer Networks

The origins of contemporary computer networking can be traced back to the development of ARPANET in the late 1960s and early 1970s. Before this pivotal period, existing computer networks primarily focused on linking terminals and remote job entry stations to mainframes. However, ARPANET introduced a transformative concept, which is networking between computers as equal peers to facilitate resource sharing. Additionally, ARPANET pioneered the novel technique of packet switching, departing from traditional message or circuit switching methods. This innovative approach efficiently allocated communication resources among users with fluctuating demands, marking a significant milestone in the evolution of computer networking[1].

The original architectural design of the network comprised three primary layers: a network layer, encompassing network access and switch-to-switch protocols, a host-to-host layer known as the Network Control Protocol (NCP), and a layer dedicated to specific applications such as file transfer, email, speech, and remote terminal support, termed as the "function-oriented protocol" layer.

By 1973, it became evident to leading figures in networking that an additional protocol layer was necessary within the hierarchy to facilitate the interconnection of diverse individual networks.

This imperative led to the creation of the new Internet Protocol (IP) and Transmission Control Protocol (TCP), which collectively superseded the NCP[1].

1.2.3 How computer networks work

In this section, we will delve into the intricate layers of computer networking: access, aggregation, and core layers. We will explore their key components and functionalities, shedding light on the protocols, devices, and technologies that drive modern network infrastructures. Understanding these layers is vital for network engineers and administrators, enabling them to design, optimize, and scale networks to meet the demands of today's digital landscape.

A. Core Layer

The core layer forms the backbone of the network, responsible for fast and reliable data transmission between different parts of the network. It consists of high-speed routers and switches optimized for forwarding packets at the fastest pace possible upward of 40 to 100 gigabits a second. Core layer devices employ protocols like MPLS (Multiprotocol Label Switching) and BGP (Border Gateway Protocol) for efficient packet routing and traffic management. Redundancy and fault tolerance mechanisms such as Equal-Cost Multipath (ECMP) routing and link-state protocols (e.g., OSPF and IS-IS) are crucial at the core layer to ensure high availability and reliability.

B. Aggregation Layer (Distribution Layer)

Positioned between the access and core layers, the aggregation layer aggregates traffic from multiple access layer switches and forwards it toward the core network. It typically comprises high-performance switches and routers capable of handling a large volume of traffic. Aggregation layer devices may implement protocols like BGP or MPLS for dynamic and efficient routing or also Link Aggregation to bundle multiple physical links into a single logical link, increasing bandwidth and providing redundancy. Additionally, features like Quality of Service (QoS) are often implemented at this layer to prioritize critical traffic flows.

C. Access Layer

This layer serves as the entry point for end devices into the network infrastructure. It consists of switches and wireless access points (APs). These devices provide connectivity to end-user devices such as computers, printers, IP phones, and IoT devices. The access layer implements protocols like Ethernet for wired connections and Wi-Fi standards for wireless connectivity. Access layer switches utilize techniques like VLANs (Virtual Local Area Networks) to segment network traffic and ensure efficient communication within local network segments.

In terms of data flow, data moves from the Access layer to the Aggregation layer, and then to the Core layer. Each layer serves a specific function and communicates with the layers directly above and below it in the hierarchy [2].

The hierarchical design of networks aims to reduce the workload on individual components, thereby increasing network efficiency and performance. It also simplifies management and troubleshooting, making the network more scalable and robust. This understanding of how networks work forms a solid foundation for any further exploration into the field of computer networking.

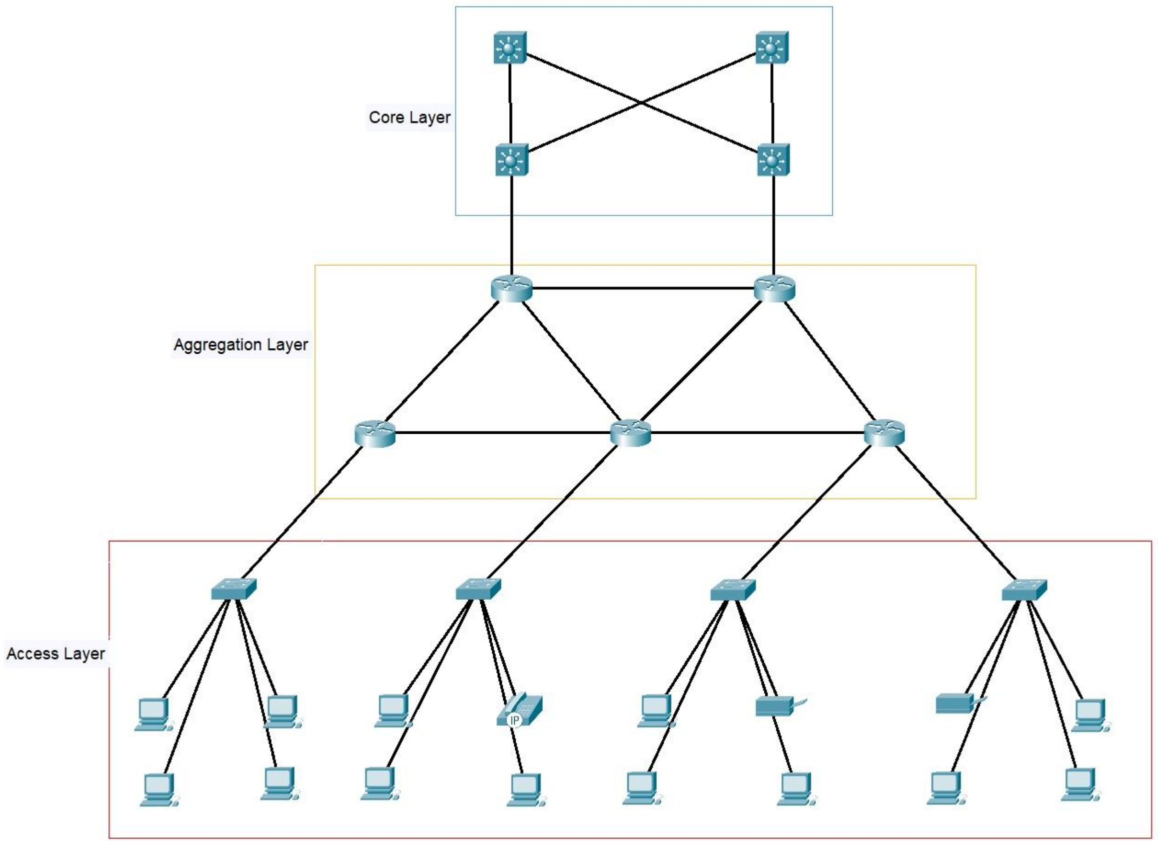


Figure 1.1: Typical Network Architecture

1.3 Software-Defined Networking (SDN)

1.3.1 Definition

SDN is a networking architecture that has emerged to address the challenges posed by the continuous growth of the Internet, smart applications, advanced machine learning, multimedia applications, social networks, etc. It aims to keep up with such evolution in terms of bandwidth, information overload, and complexity[3].

SDN separates the network's control logic from the underlying routers and switches, promoting logical centralization of network control [3]. This separation allows for a more programmable network, enabling innovation and overall improvements to the network [4].

In SDN, traditional switches are upgraded if they don't already support OpenFlow or other Flow related protocols that include flow tables remotely controlled by a separate software application called the controller. This allows for dynamic allocation of resources, management, control, security, etc [3].

One of the serious challenges in cloud computing or Internet traffic is that demand varies widely from day to day or even from hour to hour. This fluctuation makes it very hard to manage this process manually. SDN addresses this issue by allowing for automated and dynamic network management.

OpenFlow is one of the improvements developed to ease the interaction between the controller and switches. It was one of the early efforts to separate the control and data plane[5]. The necessity to enable researchers to write vendor-neutral control software, to have high-performance and low-cost implementations, to support varieties of research, and to be able to isolate experimental from production traffic were among the factors that influenced the creation of OpenFlow and SDN[5].

In summary, SDN is a paradigm shift in networking that offers a more flexible, programmable, and efficient network by separating the control plane and data plane.

1.3.2 Importance of SDN

Software-Defined Networking (SDN) provides increased control with greater speed and flexibility, so network operator can control the flow of traffic over a network by programming an open standard software-based controller, instead of manually programming multiple vendor-specific hardware devices. This gives networking administrators more flexibility in choosing networking equipment, as they can use a single protocol to communicate with any number of hardware devices through a central controller.

SDN allows for a customizable network infrastructure, where administrators can configure network services and allocate virtual resources to change the network infrastructure in real-time through one centralized location. This enables network administrators to optimize the flow of data through the network and prioritize applications that require more availability.

Furthermore, SDN also offers robust security by delivering visibility into the entire network, providing a more holistic view of security threats. With the proliferation of smart devices that connect to the internet, SDN offers clear advantages over legacy networking. Operators can create separate zones for devices that require different levels of security, or immediately quarantine compromised devices so that they cannot infect the rest of the network.

1.3.3 Application fields

Software-defined networking (SDN) is a revolutionary technology that has found applications in various fields, including 5G, cloud computing, and machine learning. Here are some of the key application areas:

A. 5G Networks:

SDN brings versatility to 5G networks. It introduces a centralized, programmable network architecture that separates the control plane from the data plane and eases network slicing which is an essential part of 5G and its low latency applications, allowing for more dynamic and efficient network management. SDN and virtualization go hand in hand to enable many of the network functions to run in software rather than in custom-built hardware[6].

B. Cloud Computing:

SDN plays a crucial role in cloud computing. It allows applications to interact with the network through APIs that enable general network maintenance, including routing, security, access control, bandwidth management, traffic management, quality of service, processor optimization, and data storage[7].

C. Machine Learning:

SDN, along with edge computing, NFV, and augmented intelligence, shapes and supports AI-driven network ecosystems. The impact of AI on various networking products and solutions, including embedded hardware[7], components, and software platforms (automation, optimization, and network transformation), is significant.

D. Internet Service Provider Networks:

SDN is also significantly beneficial for Internet Service Providers (ISPs). An architecture based on SDN techniques gives operators greater freedom to balance operational and business parameters, such as network resilience, service performance, and Quality of Experience (QoE) against operational expenditure and capital expenditure(OPEX and CAPEX), SDN allows for more flexible network management through programmable network states. This can be achieved through enabling network programmability based on open APIs. As a result, SDN will help operators to scale networks and take advantage of new revenue-generating possibilities[8].

E. Internet of Things (IoT):

SDN is beneficial for the Internet of Things (IoT) due to its scalability, programmability, and centralized network management. It provides efficient control over complex network infrastructure like IoT and can coexist with legacy networks. SDN, combined with containerization, can address most IoT challenges. Furthermore, it enhances Quality of Service (QoS) for critical network flows. These features make SDN a promising solution for managing IoT networks.

F. Edge Computing:

SDN plays a significant role in edge computing by providing a flexible and programmable network infrastructure that can support the high bandwidth and low latency requirements of edge applications[7].

G. Simplified policy changes:

With SDN, an administrator can change any network switch's rules when necessary -- prioritizing, deprioritizing or even blocking specific types of packets with a granular level of control and security.

This capability is especially helpful in a cloud computing multi-tenant architecture, as it enables the administrator to manage traffic loads in a flexible and efficient manner. Essentially, this enables administrators to use less expensive commodity switches and have more control over network traffic flows.

H. Reduced hardware footprint and operational expenditure

SDN also virtualizes hardware and services that were previously carried out by dedicated hardware. This results in the touted benefits of a reduced hardware footprint and lower operational costs.

1.3.4 SDN Infrastructure

SDN and legacy networking represent two different paradigms in network architecture. The key differences between them are primarily in their architecture, control plane, configuration and management, programmability, scalability, security, and cost[9].

A. Architecture:

Traditional networking uses fixed-function and dedicated hardware and network devices, including switches and routers, to control network traffic. On the other hand, SDN is characterized by the decoupling of control and packet forwarding planes in the network. This separation allows for OpenFlow use and the use of more open protocols[10].

B. Control Plane:

In legacy networks, each router has its own control plane, which makes independent decisions about the routing table. In contrast, SDN has a centralized control plane that provides a unified view of the entire network, enabling more efficient traffic management[10].

C. Configuration and Management:

Legacy networks often lack exposed APIs for provisioning and are unable to be modified as needed. SDN, however, allows networks to connect to apps using application programming interfaces (APIs), supporting application performance and security[10].

D. Programmability:

SDN is programmable, allowing for dynamic, on-demand network resource management and configuration. This is in contrast to legacy networks, where the functionality is primarily implemented in application-specific integrated circuits (ASIC) and other dedicated hardware[10].

E. Scalability:

SDN, being software-based, offers better scalability and flexibility compared to traditional hardware-based networking. It provides users with more control and easier resource management, allowing users to virtually manage resources with the control plane.

F. Security:

SDN's centralized control plane allows for a unified view of the network, which can enhance security by enabling more comprehensive and proactive threat management [9].

G. Cost:

SDN networks are becoming increasingly popular due to their flexibility, automation, and cost-effectiveness. In contrast, legacy networking infrastructure can be costly due to the need for dedicated hardware and devices[10].

In conclusion, SDN represents a paradigm shift from legacy networking, offering significant advantages in terms of flexibility, programmability, and cost-effectiveness. However, it's important to note that each has its own strengths and weaknesses, and the choice between SDN and legacy networking would depend on the specific requirements and context.

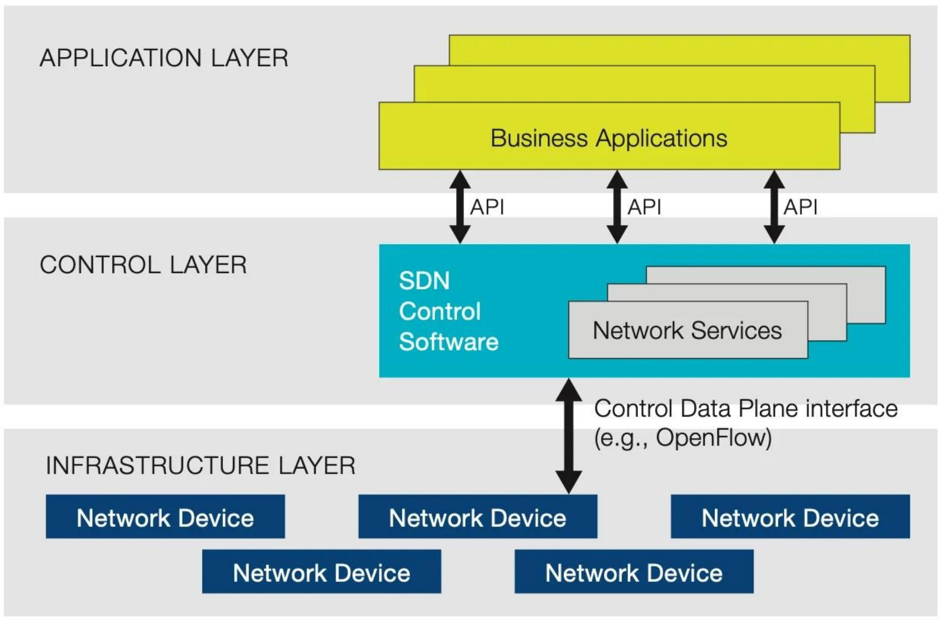


Figure 1.2: SDN Architecture [11]

1.3.5 SDN Controllers

Software-Defined Networking (SDN) controllers are the central orchestrators within the SDN paradigm, responsible for the intelligent management and dynamic control of network resources. They decouple the control plane from the data plane, facilitating programmable, flexible, and scalable network operations.

A. Centralized Control Plane

SDN controllers serve as the centralized control plane, abstracting the underlying network infrastructure and providing a comprehensive, unified view of the network. This centralization allows for consistent and holistic network management and policy enforcement.

B. Network Abstraction and APIs

SDN controllers interface with the network through:

- **Northbound APIs:** These APIs connect the SDN controller to higher-level applications and business logic, allowing applications to request network services and retrieve network state information.

- **Southbound APIs**: These APIs communicate with network devices, issuing commands and retrieving data. OpenFlow is the predominant protocol used here, enabling direct control over the forwarding behavior of network devices.

C. Core Functions of SDN Controller

- Topology Discovery: Continuously map and update the network topology to maintain an accurate network representation.
- Flow Management: Install and manage flow rules in network devices to control traffic paths based on predefined policies.
- Traffic Engineering: Optimize traffic distribution across the network to enhance performance and resource utilization while ensuring QoS.
- Network Policy Enforcement: Implement and enforce security, access control, and compliance policies across the network.

D. Prominent SDN Controllers

While open-source controllers offer flexibility and community-driven innovation, vendor-specific controllers often require proprietary hardware, leading to potential vendor lock-in.

- **Open-Source SDN Controllers**

OpenDaylight: An open-source platform under the Linux Foundation, OpenDaylight supports diverse protocols and network applications. It is flexible and modular, with strong community support and extensive integration capabilities.

ONOS (Open Network Operating System): Designed for high performance and scalability, ONOS targets carrier-grade networks. It features a distributed core architecture for high availability and robust topology management.

- **Vendor-Specific SDN Controllers**

Cisco APIC-EM: Tailored for enterprise environments, it integrates deeply with Cisco’s hardware and software ecosystem. However, it requires Cisco-specific appliances, leading to vendor lock-in.

Huawei IMaster NCE: is an advanced SDN controller BY huawedesigned to automate and intelligently manage network operations. It integrates network management, control, and analysis functions into a single platform. Key components include capturing business intents, automating network tasks, and much more.

Juniper Contrail: An SDN controller that integrates seamlessly with Juniper Networks' hardware. Best performance and features are achieved with Juniper-specific appliances, potentially limiting multi-vendor integration.

Conclusion

SDN controllers are the cornerstone of the SDN architecture, providing centralized, programmable control over network resources. They offer significant advancements in network flexibility, automation, and optimization, addressing the limitations of legacy networking.

1.3.6 Technical Challenges and Considerations in Software-Defined Networking (SDN)

A. Scalability and Performance

Challenge: As the scope and complexity of networks expand, ensuring the scalability and performance of the SDN controller becomes a critical focal point. The controller must efficiently manage an extensive array of devices and handle substantial volumes of network traffic, all while ensuring minimal latency and maximal throughput. This challenge is exacerbated in large-scale environments such as hyperscale data centers and carrier networks, where rapid network state changes demand real-time processing and adaptability.

Consideration: To address scalability and performance concerns, employing a distributed controller architecture is vital. Distributed controllers, such as ONOS and OpenDaylight, leverage multiple controller instances distributed geographically or logically across the network. These instances work in concert, utilizing algorithms for state synchronization and consistent hashing to evenly distribute network state information and manage load balancing. Additionally, the implementation of sharding techniques and partitioning strategies allows the controller to handle high traffic loads and device counts efficiently. Advanced data plane programmability, facilitated by protocols like P4, can offload complex processing tasks from the controller, further enhancing performance and scalability.

B. Security and Reliability

Challenge: The centralization inherent in SDN architecture introduces heightened security vulnerabilities and reliability risks. The SDN controller, being the brain of the network, is a prime target for cyberattacks and presents a single point of failure. Ensuring robust security and high reliability of the controller and the communication channels between the controller and network devices is essential for maintaining network integrity and availability.

Consideration: Enhancing security in SDN involves implementing multi-layered security measures. Utilizing Transport Layer Security (TLS) for securing controller-device communications is fundamental to protect against interception and tampering. Role-Based Access Control (RBAC) ensures that only authorized users can perform sensitive operations, reducing the risk of insider threats. Additionally, integrating Intrusion Detection Systems (IDS) and Security Information and Event Management (SIEM) systems with the SDN controller can provide real-time threat detection and response capabilities. For reliability, deploying multiple redundant controllers in a clustered configuration ensures high availability and resilience. Stateful failover mechanisms enable seamless controller handover in the event of a failure, minimizing downtime. Employing network slicing and micro-segmentation can also isolate failures and contain potential security breaches within specific segments of the network, reducing the overall impact.

By addressing these advanced technical challenges and considerations, SDN can achieve the necessary scalability, performance, security, and reliability to meet the demands of contemporary and future network environments.

1.4 Containers:

1.4.1 Definition

Containers are a type of software that can package up an application and its dependencies like Programming language runtime or package managers of the likes of APT and also database and drivers to name a few, having the ability to run anywhere. This is a form of operating system virtualization. Containers operate by sharing the same operating system kernel but running in isolated user spaces. They are considered lightweight because they run directly on the host machine’s hardware without the need for a hypervisor, unlike virtual machines. This means you can run many more containers on a host machine than virtual machines.

Containers are important because they allow developers to package an application with all of its dependencies into a standardized unit for software development. This means that the application will run the same, no matter where it is deployed, reducing inconsistencies and increasing efficiency.

A significant advantage of containers is that they are mostly open source. This open-source nature makes them powerful and adaptable, as it allows developers worldwide to contribute to their development and improvement. It also ensures transparency, flexibility, and a wide community support base, further enhancing their robustness and versatility.

A virtual machine (VM) emulates a real computer, running applications like a physical system. It operates on a host machine using a hypervisor, which can be software, firmware, or hardware. Each one includes a full guest operating system, allowing independent operation within the host.

1.4.2 Benefits of its Infrastructure

Container infrastructure works by creating isolated environments for applications to run. Here is a detailed explanation:

A. Packaging and Isolation:

Containers are technologies that allow the packaging and isolation of applications with their entire runtime environment. This includes all of the files necessary to run the application. This makes it easy to move the contained application between different environments (development, testing, production, etc.) while retaining full functionality[11].

B. Sharing Resources:

Containers share the same operating system kernel and isolate the application processes from the rest of the system [11]. This allows multiple containers to run on the same system without interfering with each other1. They share CPU, memory, storage, and network resources at the operating system level [12].

C. Portability:

The abstraction of applications from the environment they run in makes containers highly portable. You can easily move the containerized application between public, private, and hybrid cloud environments and data centers (or on-premises) with consistent behavior and functionality[11].

In summary, container infrastructure works by isolating applications in their own environments, sharing system resources, providing portability while being light weight on the server making them a very efficient when deployed across different environments, and managing deployments with container orchestration tools like Kubernetes.

1.4.3 Docker Containers

Docker is one of the most used container engines available, just like in the VM world we have VMs from VMware and Hyper-V from Microsoft and many others, containers have container engines and Docker is the most used and the most documented engine of the bunch, images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

Docker containers that run on Docker Engine are:

- **Standard:** Docker created the industry standard for containers, so they could be portable anywhere.
- **Lightweight:** Containers share the machine’s OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs.
- **Secure:** Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry.

Docker container technology was launched in 2013 as an open source Docker Engine. It leveraged existing computing concepts around containers and specifically in the Linux world, primitives known as cgroups and namespaces. Docker’s technology is unique because it focuses on the requirements of developers and systems operators to separate application dependencies from infrastructure.

Success in the Linux world drove a partnership with Microsoft that brought Docker containers and its functionality to Windows Server.

All major data center vendors and cloud providers have leveraged technology available from Docker and its open source project, Moby (an open source framework developed by Docker). Many of these providers are leveraging Docker for their container-native IaaS (Infrastructure as a Service) offerings. Additionally, the leading open source serverless frameworks utilize Docker container technology.[13]

1.4.4 Containers and Virtual machines

Containers and Virtual Machines (VMs) are pivotal technologies in today’s computing world. Containers, known for their portability, encapsulate an application and its environment, ensuring consistency across platforms. VMs, on the other hand, provide an abstraction of physical hardware, running a complete operating system, and offer stronger security boundaries. This thesis explores these technologies, their benefits, and their impact on the future of computing. Some of the advantages they provide are:

- **Efficient Data Processing:**

Containers have emerged as a new paradigm to address intensive scientific applications problems. Their easy deployment in a reasonable amount of time and the few required computational resources make them more suitable [14]. This means that they can handle large amounts of data efficiently, making them ideal for applications that require intensive data processing.

- **Light Virtualization Solutions:**

Containers are considered light virtualization solutions. They enable performance isolation and flexible deployment of complex, parallel, and high-performance systems[14]. This means that they can run multiple instances of an application on the same hardware without causing performance issues.

- **Modernization and Migration:**

Containers have gained popularity to modernize and migrate scientific applications in computing infrastructure management. This means that they can be used to update and move applications to a completely different computing environment without causing disruptions.

- **Reduced Computational Time:**

Containers reduce computational time processing [14]. This means that they can execute tasks faster than traditional virtual machines, improving the efficiency of the system.

- **Platform Independence:**

Containers are designed to be platform-independent. Thanks to its standardized format they can run on any system that supports the container runtime [15], such as Docker, regardless of the underlying operating system or hardware.

- **Quality of Connectivity:**

These benefits make containers a powerful tool for developers and organizations alike. They provide a lightweight, flexible, and efficient solution for deploying and managing applications, particularly in distributed computing environments.

The **figure 1.3** shows the differences between Containers and virtual machines architectures.

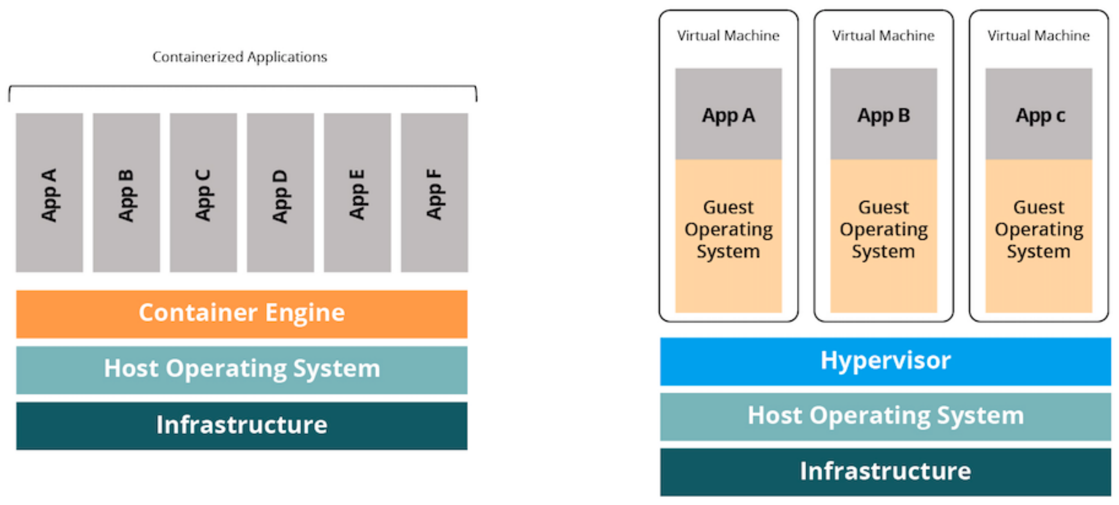


Figure 1.3: Comparing Containers with Virtual Machines [16]

1.5 Kubernetes

1.5.1 Definition

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. The system is designed to be a platform that takes care of scaling and failover for applications[17].

Kubernetes provides a framework for running distributed systems resiliently. It takes care of scaling and failover for applications, provides deployment patterns, and more. This makes it a great tool for managing cloud-native microservices[18]. Moreover, Kubernetes supports a variety of workloads, making it suitable for tasks ranging from machine learning to web applications and internet service providers.

The project was originally designed by Google and is now maintained by the Cloud Native Computing Foundation. It is written in the Go programming language and its development and design were significantly influenced by Google's Borg system.

Kubernetes has a large and rapidly growing ecosystem, with services, support, and tools widely available1. It's used in a variety of environments, from cloud-native infrastructure to on-premises data centers, and there are multiple distributions of Kubernetes available from various providers[18].

It is a robust tool for managing and orchestrating containerized microservices, is widely used by Internet Service Providers for its high availability management and simplified container orchestration. It ensures service uptime through its healing capabilities and is considered the standard for container orchestration.

1.5.2 Use Cases

Kubernetes use cases are vast and continuously being developed and improved upon while also showing the versatility of it all. It covers its application in large-scale deployments, microservices management, and much more, the aim is to underscore Kubernetes' comprehensive capabilities beyond just container management.

- **Large-Scale App Deployment**

  Kubernetes is designed to handle large-scale applications with its automation capabilities and declarative approach to configuration[19]. Features like horizontal pod scaling and load balancing allow developers to set up the system with minimal effort. Kubernetes ensures high availability and scalability to handle surges in app traffic.

- **Microservices Architecture**

Kubernetes is well suited for deploying microservices-based applications. It helps manage communication between the numerous components that make up a microservices app[20]. Kubernetes simplifies the deployment and management of these complex, distributed applications.

- **Big Data Processing**

Companies dealing with big data often use Kubernetes to support their software. It ensures portability of big data apps across multiple environments, packages apps to ensure repeatability[19], and helps optimize resource usage based on varying infrastructure requirements.

- **Machine Learning**

Kubernetes enables running entire machine learning workflows in one place, both locally and in the cloud. It allows scaling resources like GPUs to fit model needs, enables gradual upgrades[19], automates health checks and resource management, and leverages portability.

- **Serverless and PaaS Platforms**

Kubernetes can be used to create your own serverless or platform-as-a-service (PaaS) platforms. Its container orchestration and scheduling capabilities make it suitable for workflow execution engines.

- **Hybrid and Multi-Cloud Deployments**

Kubernetes facilitates cross-cloud deployments where different services run in various cloud and on-premises environments[20]. It allows connecting all compute resources to the cluster, whether in the same cloud or hosted elsewhere.

- **Simplifying Cloud Networking**
  Kubernetes simplifies cloud networking by managing networking within the cluster, connecting services together without configuring host-level networking rules.

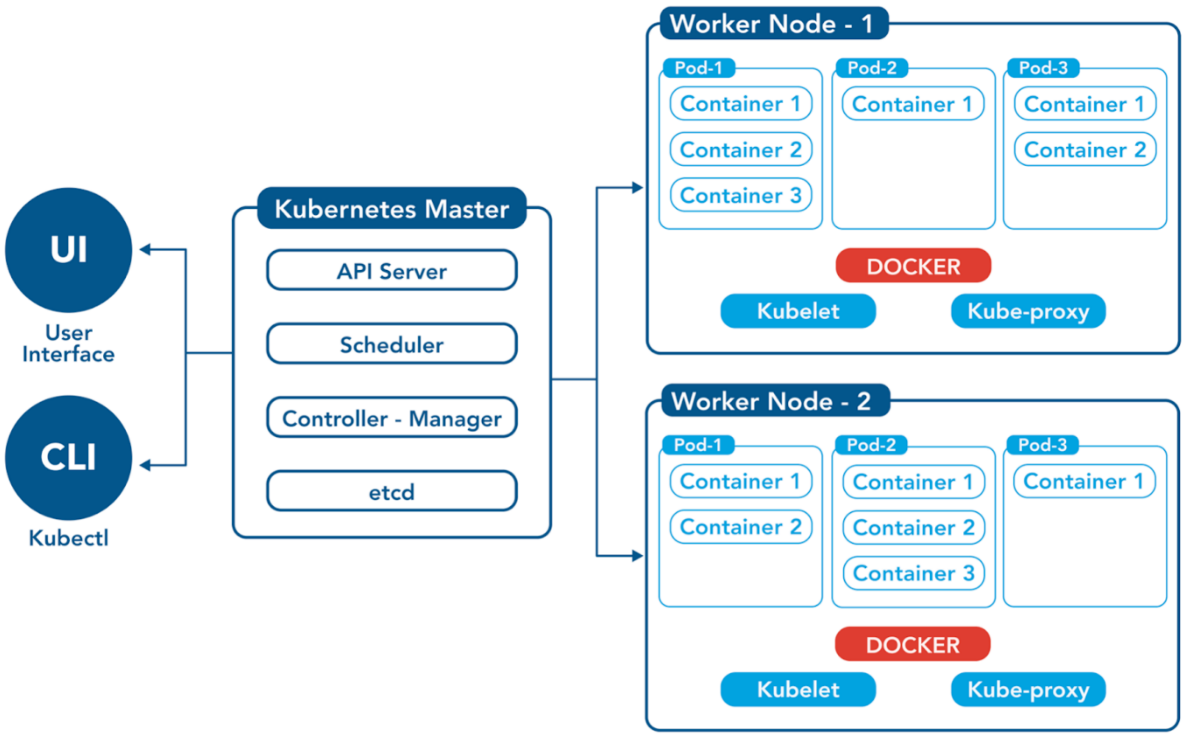


Figure 1.4: Kubernetes Architecture overview [21]

1.5.3 Kubernetes Components

Kubernetes offers a spectrum of resources, services, and tools for application management. Some of the most used options are explored below.

A. Kubernetes Master Components:

- **API Server (kube-apiserver):**

Role: The API server acts as the central management entity for the Kubernetes control plane. It serves as the primary point of interaction for administrators and users, handling all requests to manage the cluster.

Functionality: It processes RESTful API calls and updates the state of various components in the etcd store. It also validates and configures data for the API objects, including pods, services, and replication controllers.

- **Scheduler (kube-scheduler):**

Role: The scheduler assigns tasks (pods) to worker nodes based on resource availability and other constraints.

Functionality: It continuously monitors the state of the cluster, determining which nodes have the resources to run a pod. It considers factors like CPU, memory, and network availability to optimize workload distribution and ensure efficient use of resources.

- **Controller Manager (kube-controller-manager):**

Role: This component runs various controller processes that regulate the state of the cluster.

Functionality: Each controller watches the shared state of the cluster through the API server and makes changes to move the current state towards the desired state. Key controllers include the Node Controller (manages node status), Replication Controller (ensures the correct number of pod replicas), and Endpoints Controller (manages endpoint objects).

- **etcd:**

Role: etcd is a distributed key-value store used for persistent storage of all cluster data.

Functionality: It stores configuration data, state information, and metadata, ensuring consistency and reliability. etcd provides a consistent and highly available data store used for storing all the cluster's state and configuration.

B. Worker Node Components:

- **Kubelet:**

Role: The kubelet is an agent that runs on each worker node, ensuring containers are running in a pod as specified by the Kubernetes control plane.

Functionality: It communicates with the API server to get the specifications of the pod and ensures that the containers described in these pod specs are running and healthy. It also reports the status of the node and pods back to the Kubernetes master.

- **Kube-proxy:**

Role: Kube-proxy maintains network rules on each node, enabling network communication to and from pods.

Functionality: It manages the routing of traffic coming into the node, distributing it to the appropriate pods, and providing load balancing and network address translation (NAT) services. It ensures that each pod gets a unique IP address and can communicate with other pods and services within the cluster.

- **Container Runtime (Docker in this example):**

Role: The container runtime is responsible for running the containers within the pods.

Functionality: In this example, Docker is used as the container runtime. It pulls the necessary container images from a registry, starts and stops containers, and ensures they are running as expected.

C. User Interaction Components:

- **User Interface (UI):**

Role: The UI provides a graphical interface for users to interact with the Kubernetes cluster.

Functionality: Users can deploy, manage, and monitor applications running in the cluster using a visual dashboard. It provides insights into the cluster's state, health, and performance.

- **Command-Line Interface (CLI) - kubectl:**

Role: kubectl is a command-line tool that interacts with the Kubernetes API server.

Functionality: It allows users to execute commands to deploy and manage applications, inspect and manage cluster resources, and view logs and status information. kubectl provides a powerful interface for automating tasks and scripting cluster operations.

D. Interaction Flow:

- Users interact with the Kubernetes cluster through the UI or CLI (kubectl), sending commands to the API server.
- API Server processes these commands, updating the state of the cluster and storing this information in etcd.
- Scheduler assigns pods to suitable worker nodes based on resource availability and other constraints.
- Controller Manager maintains the cluster's desired state by managing various controllers.
- Kubelet on each worker node ensures the specified containers are running correctly within their pods.
- Kube-proxy handles network communications on each node, ensuring pods can communicate with each other and external services.
- etcd maintains a consistent and highly available data store for all cluster state and configuration data.

This comprehensive architecture ensures Kubernetes provides a scalable, reliable, and efficient platform for managing containerized applications.

1.6 Related works

The convergence of Software Defined Networking (SDN), containerization, and Kubernetes has revolutionized network management and service delivery. In this section, we explore existing research and practical implementations that pave the way for efficient, scalable, and dynamic networks. By examining relevant works and real-world deployments, we gain insights into the challenges, solutions, and best practices for deploying SDN-based networks using containers and Kubernetes,

On this particular work by Intidhar Bedhief, the paper proposes an innovative model for IoT architecture by combining Software Defined Networking (SDN) and containerization (specifically Docker). The architecture addresses challenges related to IoT heterogeneity, network control, and QoS requirements. By integrating SDN principles, it achieves better control over heterogeneous IoT networks. Leveraging Docker, it ensures that IoT devices can run lightweight, isolated containers. The proposed architecture abstracts communication and simplifies management. Validation using a smart supermarket use case demonstrates its effectiveness.

Although technologies such as SDN and NFV have been present for some time, it is with the emergence of 5G that they will prove their true potential. First, they provide a financial advantage. In [22], a study was conducted to analyze the impact of using SDN, NFV and Cloud computing in 5G networks for the CAPEX, the OPEX and the total cost of ownership (TCO). It was observed that in comparison with the traditional architecture, the CAPEX would be reduced by 68%, the OPEX by 63%, and the TCO by 69%.

In the same faculty where I am currently conducting my research, Ammour Mohammed Chikh & Debbakh Fadia carried out a noteworthy study recently. Their thesis, titled “Performance Evaluation of Software Defined –Network (SDN) Controller”[23], provides an in-depth analysis of the performance and resilience of SDN controllers. Utilizing controllers such as HPE-VAN, ONOS, and Open daylight, they created an emulated SDN environments for testing and evaluation purposes. The research delves into the influence of various network parameters, including topology, traffic patterns, and controller placement, on the overall performance. The insights gained from this study are intended to guide network designers and operators in identifying the most effective configurations to enhance performance and fault tolerance in SDN networks.

They also explained the benefits of SDN infrastructure compared to traditional network infrastructure while ending on a note that summarizes that Open daylight is one of the more versatile and reliable options they tested with the quickest recovery times

1.7 Conclusion

In conclusion, this Chapter has provided a detailed look at the current state of telecommunications, focusing on the challenges faced by Internet Service Providers (ISPs). The chapter traced the evolution of networking technologies from ARPANET to today's modern infrastructures, highlighting the difficulties legacy network designs have in meeting increasing demands for bandwidth, security, and flexibility.

The problem statement identified key issues ISPs face, such as difficulties in managing networks, scaling them efficiently, and high operational costs. To address these challenges, the integration of Software-Defined Networking (SDN) with containerization technologies like

Kubernetes was proposed. This chapter outlined the main goals of the research: to examine the limits of legacy networks, explore the benefits of combining SDN with containers, and create a practical guide for ISPs to implement these technologies.

Overall, this Chapter has set a strong foundation for the research, outlining the scope and objectives that will guide the research into advanced network architectures. This groundwork is essential for the detailed methodological exploration and technical implementation discussed in the following chapter.

Chapter 2:

Technical Framework and Implementation Methodology for Network Enhancements

2.1 Introduction:

In this chapter, we outlined the methodological framework employed to investigate the integration of Software-Defined Networking (SDN) with containerization technologies such as Kubernetes within Internet Service Provider (ISP) networks. Our approach is structured to facilitate a thorough exploration of the technical intricacies and operational benefits of this integration. We will outline the selection criteria for our SDN controllers and container orchestration tools, elaborate on the experimental setup designed to simulate a realistic ISP environment, and detail the implementation steps required to configure and test our network architecture. This comprehensive methodological approach aims to provide a robust foundation for the subsequent analysis and evaluation phases, ensuring that our research is both rigorous and replicable.

2.2 Materials

In this project, we needed a capable Workstation computer with enough RAM and CPU cores for the simulation software to handle the complex and large setting we are experimenting on so the materials used are as follows:

2.2.1 Workstation Computer

It is required that we use a powerful workstation computer since network simulation software requires a lot of RAM and CPU cores, especially in our case where we are simulation nearly the whole Internet Service Provider's Network with all the mature configurations and complex routing protocols, the specifications of our workstation machines goes as follows:

- Operating System: Microsoft Windows 11
- RAM: 64GB
- CPU: 12cores, 24 threads
- Storage: 2TB (512GB will suffice)

Note that you don't need these exact specifications for the simulation you could use less or more depending on what's available, for the simulation alone you would be fine with 512GB of storage, and less RAM if you used less demanding routers in the simulation, as if or the GPU it wasn't necessary for the simulation to run it's purely based on the CPU calculations.

2.2.2 Simulation Software

In the rapidly evolving field of telecommunications, network simulators have become indispensable tools for designing, testing, and troubleshooting network configurations without the need for physical hardware. These tools range from open source to licensed programs, simulators like GNS3, EVE-NG, Cisco Packet Tracer, and Containerlab. Each of these tools offers unique features and capabilities tailored to different use cases, from professional and enterprise environments to educational and container-based projects. Understanding their strengths and weaknesses is crucial for selecting the most appropriate simulator for our specific requirements while also providing resources for researchers and professionals to use as a brief guide on choosing the right network simulator for anyone experimenting with SDN, Containers or Kubrenetes.

- **EVE-NG**

  EVE-NG provides a robust, web-based interface that simplifies remote access and management, making it ideal for enterprise environments and collaborative projects. It supports a wide range of network devices from multiple vendors. The platform allows multiple users to work on the same topology simultaneously, enhancing teamwork. However, EVE-NG demands high system resources, especially when running multiple instances, and its advanced features like containers are only available in the paid Professional edition.

- **GNS3:**

  GNS3 offers extensive flexibility and realism, making it an excellent choice for professional network engineers who require detailed multi-vendor support and the ability to integrate with real hardware. Its high customizability and comprehensive device support enable the creation of complex network topologies that closely mimic real-world scenarios. It also is the most documented simulator of the bunch, However, GNS3 is resource-intensive, demanding significant CPU and memory. The setup process can be complex and time-consuming, and troubleshooting can be challenging due to the variety of integrated components.

- **Packet Tracer**

  Cisco Packet Tracer is tailored for educational use, offering an intuitive interface and efficient performance on low-end hardware. It is designed to help students learn networking concepts with accurate simulations of Cisco devices and IOS commands. The tool is resource-efficient and available on multiple platforms, making it accessible to a broad audience. However, Packet Tracer is limited to Cisco devices and technologies only, lacking multi-vendor support, and its simulations are less realistic compared to more advanced simulators like GNS3 or EVE-NG. It is also primarily designed for educational purposes, not for complex professional network simulations like ours.

- **Containerlab:**

  Containerlab excels in modern, container-based environments, providing strong support for Docker and Kubernetes. It simplifies the deployment of network topologies using container images, offering efficient, scalable, and automated network functions. Containerlab is resource-efficient and integrates well with CI/CD pipelines and automation tools, making it ideal for cloud-native projects. However, it has a steep learning curve since it is all command line based and not a graphical interface, and also requires knowledge of Docker and Kubernetes, which may be a barrier for new users, The focus on containerized functions means it has limited support for traditional network devices, and it has a smaller community with fewer resources compared to more established simulators. Integrating Containerlab with existing non-containerized environments can also be challenging.

- **What we ended up with:**

  For our project we chose GNS3 for its extensive documentation and support for the various protocols and technologies we need, we would like to use EVE-NG for its simplicity and modern approach, problem is container support is locked behind the paid professional version while we prioritized making this project completely free and open-source accessible for anyone following through. Containerlab meets these conditions but at the time of writing, it supports a handful of Nokia and juniper routers while it is a very powerful and efficient tool it requires a steep learning curve to operate its command line

2.2.3 SDN Controller

A plethora of options exists concerning SDN controllers, ranging from open-source to proprietary (vendor-specific) solutions. Our focus primarily centers on open-source alternatives such as ONOS, OpenDaylight, Floodlight, among others.

Amidst the diverse array of SDN controllers, OpenDaylight emerges as a preeminent selection due to its notable attributes, extensive documentation, and dependable performance. Positioned as an open-source SDN controller platform under the auspices of the Linux Foundation, OpenDaylight benefits from a vibrant community ecosystem and a comprehensive repository of documentation materials. Its modular architecture, elucidated by our colleagues at [23], facilitates adaptable customization and seamless integration within varied networking environments.

Furthermore, OpenDaylight demonstrates robust stability and performance, a fact substantiated by empirical studies such as that conducted by our colleagues [23], who highlighted its rapid recovery capabilities in response to sudden link disruptions. This resilience underscores its utility in maintaining network continuity and mitigating service interruptions. Additionally, OpenDaylight's adherence to standardized protocols and Application Programming Interfaces (APIs) augments interoperability, fostering harmonious integration with diverse network infrastructures and protocols.

2.2.4 Other tools

- **VMware player**

  VMware Player is essential for GNS3 simulations as it enables virtualization, allowing users to run multiple virtual machines (VMs) on a single physical host (a workstation on our case). This capability is crucial for emulating complex network topologies with various devices and operating systems. VMware Player facilitates the integration of real-world network elements into virtual environments, enhancing the accuracy and versatility of GNS3 simulations. Additionally, we used it to deploy the SDN controller into since it need to work on a separate server instance.

- **Putty**

  Putty is utilized in networks primarily for its SSH (Secure Shell) capability, enabling secure remote access and management of network devices such as routers, switches, and servers. It got used when accessing devices from GNS3 Its versatility across different protocols, lightweight design, and compatibility with various operating systems make it an essential tool for configuration, monitoring, and troubleshooting tasks within network infrastructures.

- **WinSCP**

WinSCP is an open-source file transfer utility for Windows that supports FTP, SFTP, SCP, and WebDAV protocols. It is primarily used for securely transferring files between a local and a remote computer. WinSCP provides a graphical interface for easy file management, ensuring data security during transfers. It is commonly used for tasks such as uploading website files to a server, transferring configuration files to network devices, and backing up data to a remote server, it was used in our case to deploy the SDN controller into the server, and used in the web server configuration.

2.3 Methodology

2.3.1 Planning

Our exploration commenced with a search for Internet Service Provider (ISP) network architectures available online, seeking a suitable model to integrate with SDN, Containers, and Kubernetes. Given the acknowledged benefits of these modern technologies within ISP networks, our attention naturally turned to their potential applications.

Subsequently, it was determined that an immersive internship experience within an active ISP environment would provide invaluable insights. DJEZZY was selected for its reputation for embracing modern network innovations and technologies.

Under the mentorship of Mr. Soufiane Saidi, SDN / IP Backbone Administrator at DJEZZY, we gained a comprehensive understanding of ISP network operations. Mr. Saidi generously shared his expertise, including an exemplar ISP network architecture, emphasizing the significance of th e Aggregation layer for optimal SDN integration.

At present, DJEZZY is in the process of implementing an SDN solution, underscoring their proactive stance toward technological progress. This ongoing initiative serves to inspire us further in constructing a meticulous project for our research endeavors. Additionally, it presents an opportune moment for DJEZZY to conduct a thorough review of the anticipated benefits before proceeding with the active deployment of the technology. This collaborative approach not only enhances the validity and relevance of our research efforts but also contributes to DJEZZY's strategic decision-making process regarding the integration of SDN within their network infrastructure.

Mr. Saidi provided the following architecture, which closely represents the current active architecture used by the company. This figure illustrates the IP Core layer and the Aggregation layer, with our primary focus being on the Aggregation layer.

Under the guidance of our internship supervisor, we formulated the project plan, concluding that our SDN solution would be most effectively deployed at the aggregation layer. This decision aligns with DJEZZY's active efforts in this section of the network.

For the Kubernetes and container solution, deployment will be configured to connect to the IP-Core network, reflecting the configuration of the live network's data center. This deployment will serve as a web server to demonstrate its agility and automation capabilities. To simulate the combined benefits, a web server within a Docker container will be utilized. By generating a high volume of user requests until saturation, the Kubernetes master node (control plane) will automatically deploy and configure an identical container to manage the increased demand. This approach will showcase the scalability, speed, and efficiency of Kubernetes in dynamically responding to fluctuating workloads, effectively load-balancing traffic between containers and highlighting its capabilities in a functional environment.

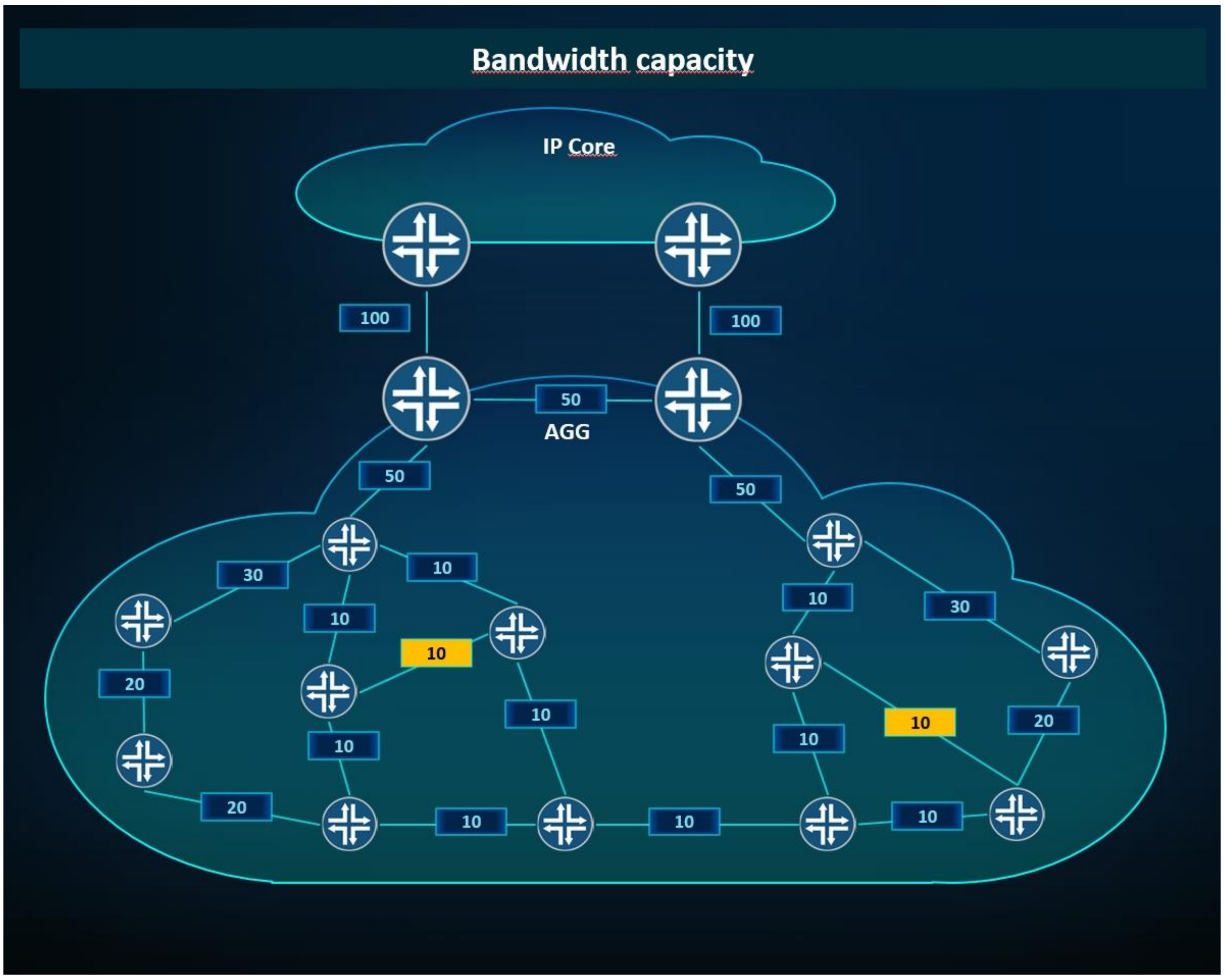


Figure 2.1: ISP Backbone Aggregation Network Topology Showing Bandwidth Capacity of the Links

2.3.2 Changes Made to Djezzy's Network Sample

After careful planning and consideration, we made some needed modifications to the network to ease the operability and efficiency of our methods

A. Changing the router vendor:

Cisco routers were selected over DJEZZY's Huawei routers due to the limited availability of Huawei routers in GNS3, which are one SD-WAN specific and another one being a full-scale edge router. Additionally, the`` change was necessitated by the comparatively insufficient documentation provided by Huawei. In contrast, Cisco offers an extensive library of documentation and guides. Moreover, Cisco's leading position in the global networking equipment market enhances the value of learning and implementing their hardware, providing substantial benefits for research and future career prospects.

B. Changes to IP-Core Layer:

During the planning phase, our internship supervisor suggested that we simplify the Actual IP-Core layer of the network since it is out of the scope of our study, because we would be deploying and observing the improvement on the Aggregation layer then the IP-Core layer should be made just as we showed in the figure

2.4 Building the network

In this phase of the project, the challenge involved constructing an accurate replication of Djezzy network deployment using the topology they provided. This task was demanding due to our lack of experience on deploying and reliably configuring a professional and complicated network an ISP can rely on unlike a trained professional network engineering team.

Since the primary focus is not on the current Legacy network, specific details regarding the construction and deployment of the network within the simulation software will be briefly addressed.

2.4.1 Legacy Network Configuration

After the positioning of the routers and establishment of the required connections, selection of an IP pool was made. Guidance on the required objectives to focus on and specification of the protocols deployed within the network were provided by Mr. Saidi.

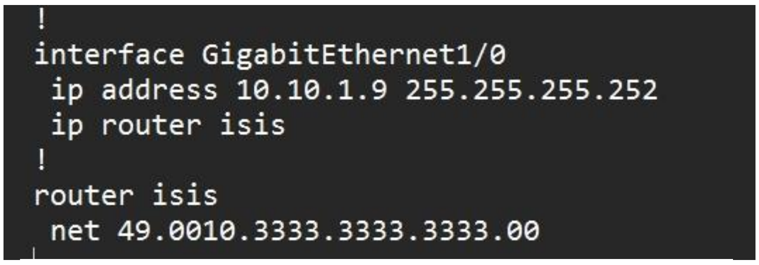
Configuration commands are demonstrated for a single router examples were taken from ASG (Aggregation Site Gateway) routers following a similar configuration process.

In contrast, ASBR (Autonomous System Boundary Router) routers utilize different commands due to their distinct responsibilities within the network.

The network protocols used include **IS-IS**, **BGP**, **OSPF**, **MPLS**, and **MPLS-TE**.

A. ISIS Configuration

IS-IS is used with BGP to leverage its strengths in internal routing efficiency, it is essential to the correct deployment of the BGP protocol

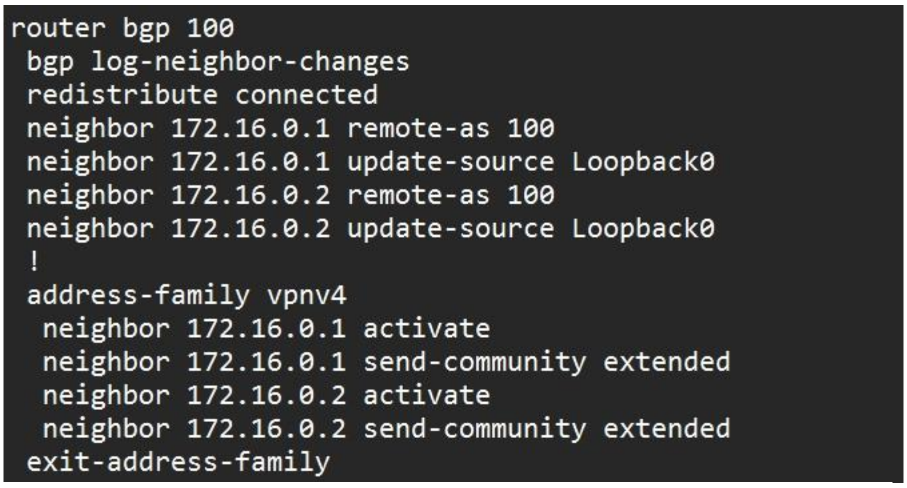


```
!
interface GigabitEthernet1/0
 ip address 10.10.1.9 255.255.255.252
 ip router isis
!
router isis
 net 49.0010.3333.3333.3333.00
```

Figure 2.2: IS-IS configuration

B. BGP implementation

BGP had to be configured on each router, matching the area chosen while specifying each neighbor, **VPNv4** was used to distribute routing information across the routers
The following config was taken from ASG3, all the routers share similar if not the same config with the difference being IP Addresses.

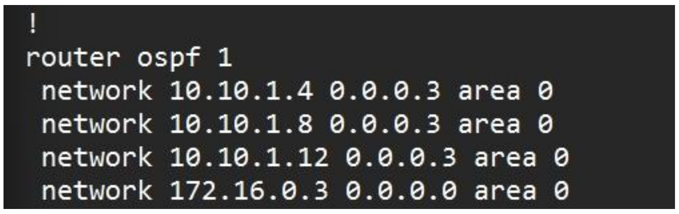


```
router bgp 100
 bgp log-neighbor-changes
 redistribute connected
 neighbor 172.16.0.1 remote-as 100
 neighbor 172.16.0.1 update-source Loopback0
 neighbor 172.16.0.2 remote-as 100
 neighbor 172.16.0.2 update-source Loopback0
 !
 address-family vpnv4
  neighbor 172.16.0.1 activate
  neighbor 172.16.0.1 send-community extended
  neighbor 172.16.0.2 activate
  neighbor 172.16.0.2 send-community extended
 exit-address-family
```

Figure 2.3: BGP Configuration Sample

C. OSPF

Before deploying MPLS (Multiprotocol Label Switching) in a network infrastructure, it is common practice to utilize OSPF (Open Shortest Path First) as the underlying intra-domain routing protocol, by leveraging OSPF as the foundation for intra-domain routing, MPLS can be deployed on top of this stable and efficient routing infrastructure.

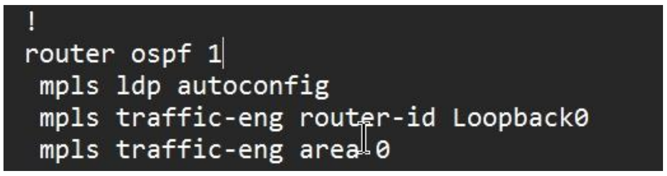


```
!
router ospf 1
 network 10.10.1.4 0.0.0.3 area 0
 network 10.10.1.8 0.0.0.3 area 0
 network 10.10.1.12 0.0.0.3 area 0
 network 172.16.0.3 0.0.0.0 area 0
```

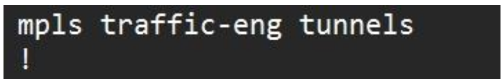
Figure 2.4: OSPF Commands

E. MPLS / MPLS-TE

We deployed MPLS (Multiprotocol Label Switching) and then added MPLS-TE (Traffic engineering) due to its real implementation in the Djezzy network, most if not all enterprise and ISP WAN networks utilize MPLS, for tunneling and QoS features. Examples taken from ASG3.

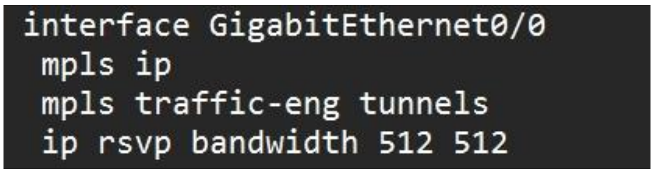


```
!
router ospf 1
 mpls ldp autoconfig
 mpls traffic-eng router-id Loopback0
 mpls traffic-eng area 0
```



```
mpls traffic-eng tunnels
!
```

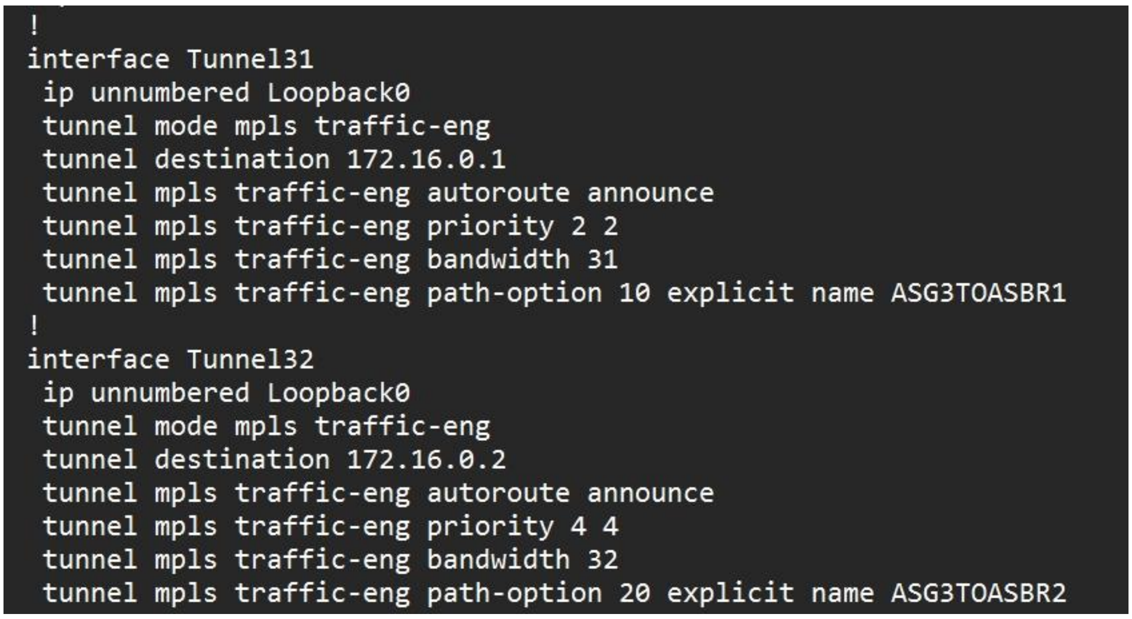
Figure 2.5: Enabling MPLS and MPLS-TE Globally on the Router



```
interface GigabitEthernet0/0
 mpls ip
 mpls traffic-eng tunnels
 ip rsvp bandwidth 512 512
```

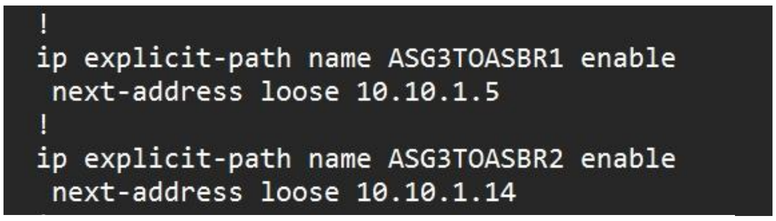
Figure 2.6: Turning on MPLS and MPLS-TE on the interfaces

MPLS Tunnels are an essential part of the MPLS Protocol Then we would have to add tunnels on each interface from and into each router, so each link has 2 tunnels, this sample is taken from ASG router, ASBR routers had more tunnels configured, we had to configure 12 tunnels Each.



```
!
interface Tunnel31
 ip unnumbered Loopback0
 tunnel mode mpls traffic-eng
 tunnel destination 172.16.0.1
 tunnel mpls traffic-eng autoroute announce
 tunnel mpls traffic-eng priority 2 2
 tunnel mpls traffic-eng bandwidth 31
 tunnel mpls traffic-eng path-option 10 explicit name ASG3TOASBR1
!
interface Tunnel32
 ip unnumbered Loopback0
 tunnel mode mpls traffic-eng
 tunnel destination 172.16.0.2
 tunnel mpls traffic-eng autoroute announce
 tunnel mpls traffic-eng priority 4 4
 tunnel mpls traffic-eng bandwidth 32
 tunnel mpls traffic-eng path-option 20 explicit name ASG3TOASBR2
```

Figure 2.7: MPLS Tunnel Configurations Sample



```
!
ip explicit-path name ASG3TOASBR1 enable
 next-address loose 10.10.1.5
!
ip explicit-path name ASG3TOASBR2 enable
 next-address loose 10.10.1.14
```

Figure 2.8: MPLS-TE Dynamic Configuration

Following this abbreviated configuration, a functioning topology of an ISP network was established. This foundation allows for the implementation and deployment of the SDN solution, with comparisons to be analyzed in later sections.

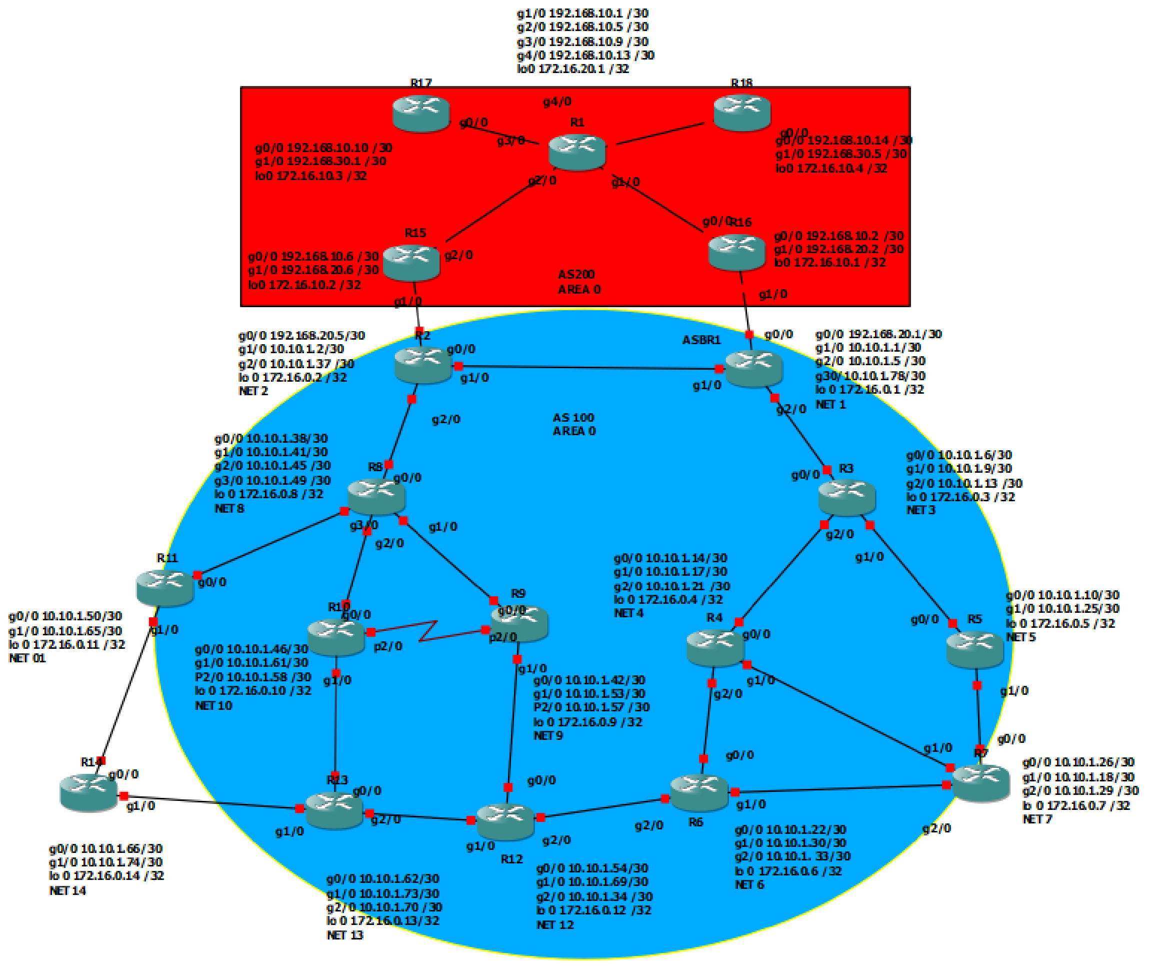


Figure 2.9: A Functioning ISP Network Topology Showing Aggregation and IP-Core Layers built into GNS3

The figure presents the ISP network topology incorporating aggregation and IP-core layers, as constructed in GNS3. Various routers and networks are interconnected to form this network design. The red segment represents the IP-Core with a BGP area of AS200 and Area 0 for the OSPF Protocol, while the blue segment represents the aggregation layer with its BGP area being AS100 while OSPF also within Area 0, illustrating different routing areas.

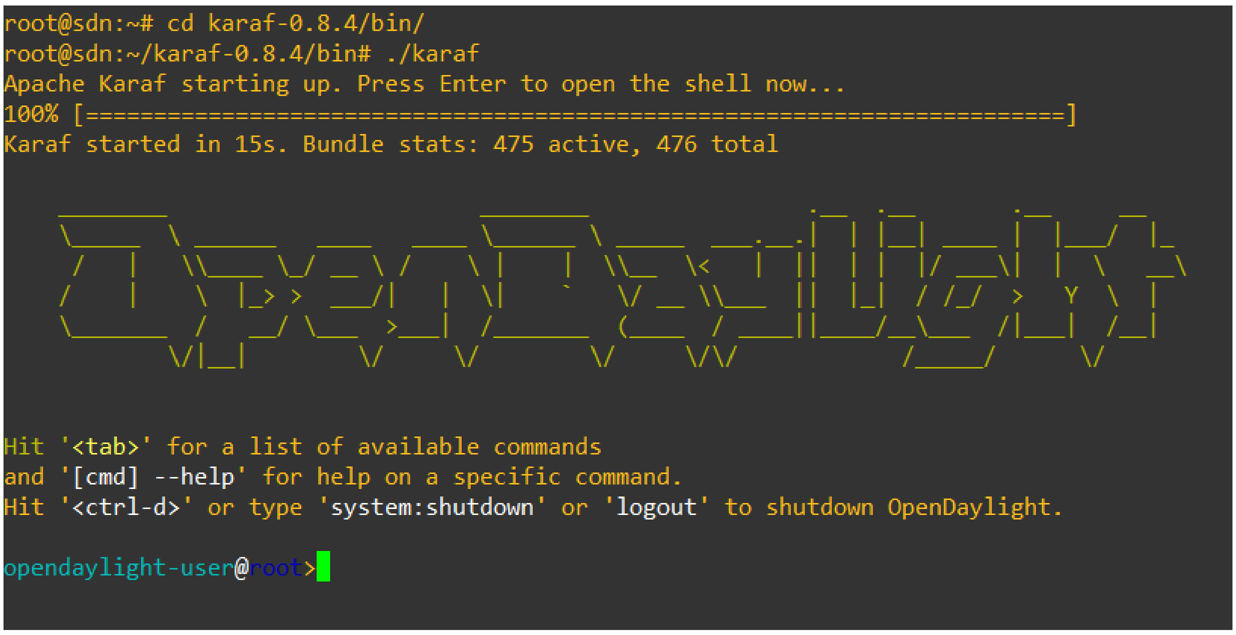
2.5 Deploying the SDN Ready Network

This section outlines the transition from a legacy ISP network architecture to an SDN enhanced network. The deployment process involves integrating the SDN controller into the existing infrastructure, focusing on the aggregation layer where the most significant improvements are expected.

2.5.1 Essential Modification Needed for The SDN Deployment

A. Deploying The Controller on a Linux server

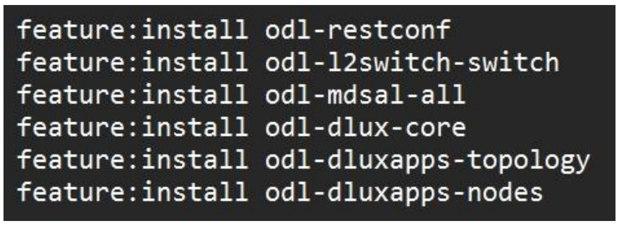
To host the SDN controller we chose (OpenDaylight) it was required to spun up a Linux VM, we chose Ubuntu server 24.04 LTS as the Linux image and installed ODL on top of it then connecting it to the network as fellows.



```
root@sdn:~# cd karaf-0.8.4/bin/
root@sdn:~/karaf-0.8.4/bin# ./karaf
Apache Karaf starting up. Press Enter to open the shell now...
100% [========================================================================]
Karaf started in 15s. Bundle stats: 475 active, 476 total

    ________                       ________                .__  .__       .__     __
    \_____  \ ______   ____   ____ \______ \ _____  ___.__.|  | |__| ____ |  |___/  |_
     /   |   \\____ \_/ __ \ /    \ |    |  \\__  \<   |  ||  | |  |/ ___\|  |  \   __\
    /    |    \  |_> >  ___/|   |  \|    `   \/ __ \\___  ||  |_|  / /_/  >   Y  \  |
    \_______  /   __/ \___  >___|  /_______  (____  / ____||____/__\___  /|___|  /__|
            \/|__|        \/     \/        \/     \/\/            /_____/      \/


Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
```

Figure 2.10: OpenDaylight Booting in Ubuntu Server

B. Configuring the Controller

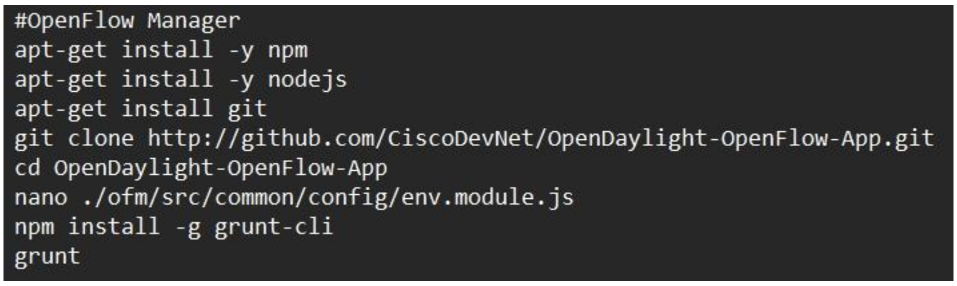


```
feature:install odl-restconf
feature:install odl-l2switch-switch
feature:install odl-mdsal-all
feature:install odl-dlux-core
feature:install odl-dluxapps-topology
feature:install odl-dluxapps-nodes
```

Figure 2.11: Controller Full Initial Configuration

C. Installing OpenFlow Manager

To fully build the architecture of the SDN network, we need an application that will control the flows in network, To install OpenFlow Manager, follow these steps (on the same Ubuntu Server):



```
#OpenFlow Manager
apt-get install -y npm
apt-get install -y nodejs
apt-get install git
git clone http://github.com/CiscoDevNet/OpenDaylight-OpenFlow-App.git
cd OpenDaylight-OpenFlow-App
nano ./ofm/src/common/config/env.module.js
npm install -g grunt-cli
grunt
```

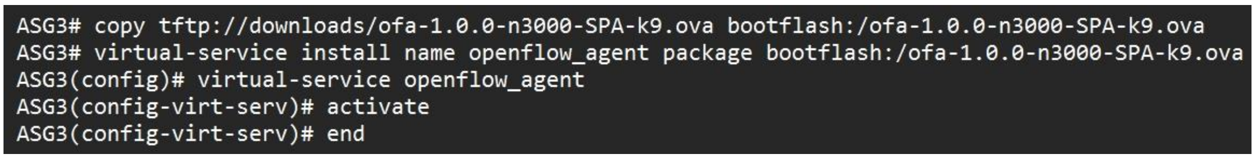
Figure 2.12: Steps for Installing OpenFlow Manager

These commands are for setting up the OpenFlow Manager, which includes installing Node.js and npm, cloning the repository from GitHub, and running the Grunt task runner.

D. Activating Essential Protocols on The Network

- **Installing open-flow plugin on cisco routers**

Cisco routers unlike their switches, don’t come with OpenFlow protocol by default so we have to install it as a plugin, OpenFlow is essential for the operation and communication of the SDN controller with the other



```
ASG3# copy tftp://downloads/ofa-1.0.0-n3000-SPA-k9.ova bootflash:/ofa-1.0.0-n3000-SPA-k9.ova
ASG3# virtual-service install name openflow_agent package bootflash:/ofa-1.0.0-n3000-SPA-k9.ova
ASG3(config)# virtual-service openflow_agent
ASG3(config-virt-serv)# activate
ASG3(config-virt-serv)# end
```

Figure 2.13: Installing Open-Flow Plugin on Cisco Routers

After activating the OpenFlow protocol in the routers, the OpenFlow manager web UI is consulted to verify the topology. It is observed that certain links are absent from the displayed topology, despite being detected when conducting a more in-depth examination via the command-line interface (CLI). This phenomenon is regarded as standard behavior inherent to OpenFlow, wherein only primary paths are exhibited; any alternative paths will be displayed as they come into use. This characteristic aligns with the functionality of the Rapid Spanning Tree Protocol (RSTP), which was earlier configured to prevent the formation of loops within the topology. This preventative measure is crucial in maintaining network integrity and mitigating potential packet loss scenarios.

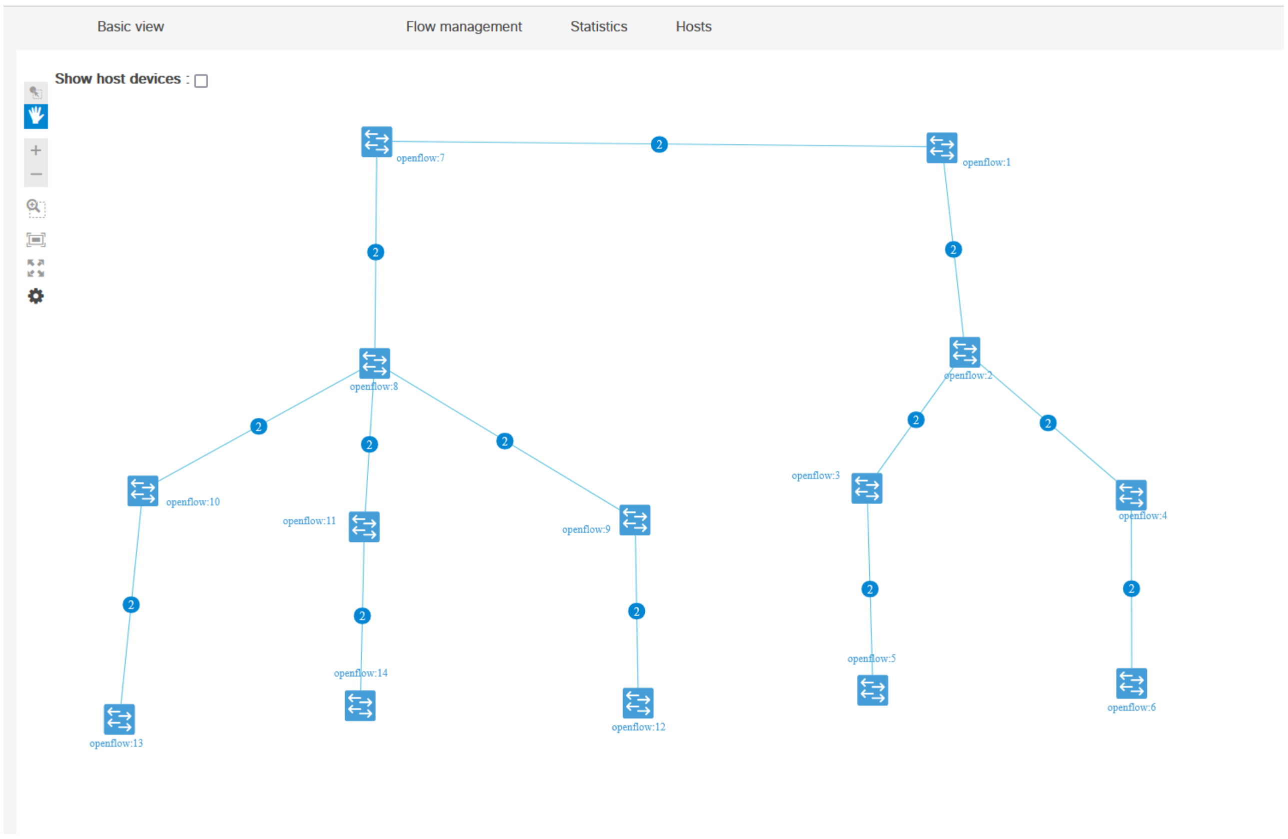


Figure 2.14: Network Topology through OpenFlow Manager Dashboard

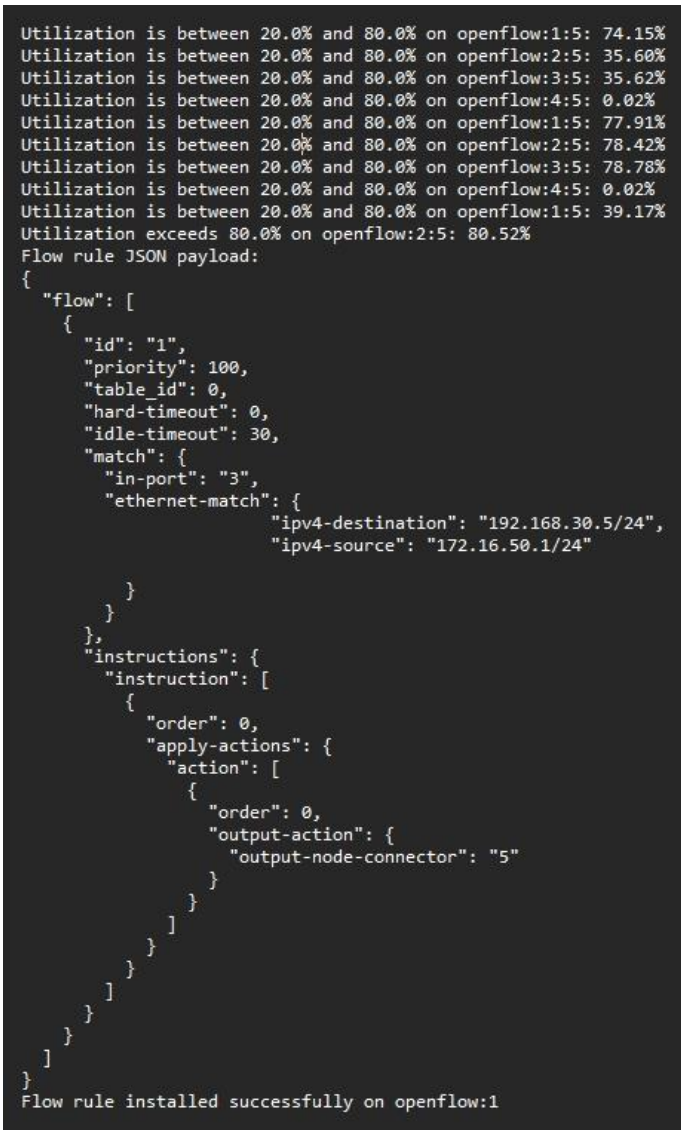
2.5.2 Network Load Balancing

```
# Define constants
controller_ip = "192.168.1.100"
controller_port = "8181"
auth = ('admin', 'admin')
threshold = 0.8
low_threshold = 0.2
check_interval = 5
flow_idle_timeout = 30

# Define path configurations
path1 = [switch_id, in_port, out_ports]
path2 = [switch_id, in_port, out_ports]

# Define helper functions

def get_port_stats(switch_id, port_id):
    # Make API call to get port statistics
    # Return port statistics

def calculate_bandwidth_utilization(previous_stats, current_stats, interval):
    # Calculate bandwidth utilization based on previous and current statistics
    # Return utilization

def install_flow_rule(switch_id, in_port, out_ports, flow_id):
    # Install flow rule using REST API
    # Print debug information

def check_flow_exists(switch_id, flow_id):
    # Check if flow rule exists on switch
    # Return True if exists, False otherwise

def remove_flow_rule(switch_id, flow_id):
    # Remove flow rule from switch
    # Print debug information

def main():
    # Initialize variables
    previous_stats = {}
    active_flows = False

    while True:
        # Gather port statistics for output ports in path1
        # Calculate and check utilization for each output port
        # Install or remove flow rules based on utilization thresholds
        # Update previous statistics
        # Sleep for check_interval

if __name__ == "__main__":
    # Call main function
```

Figure 2.15: Python Script for Load Balancing the Traffic on the Network

On this section, we describe a Python script that we wrote and integrated into our OpenFlow manager to dynamically monitor and manage network bandwidth utilization. This script is an essential component for enhancing the load balancing capabilities of the SDN controller.

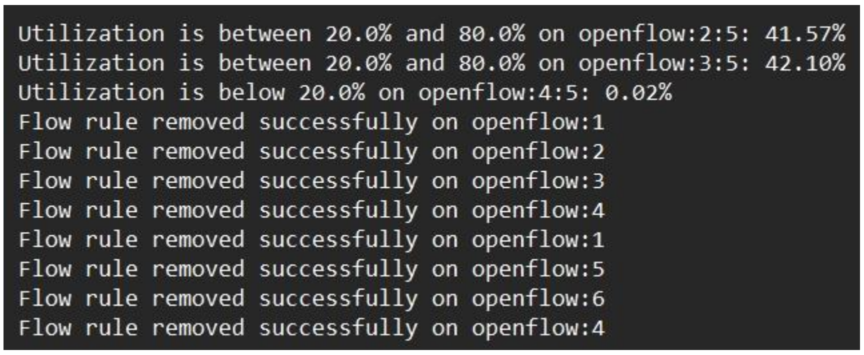
It is designed to periodically fetch port statistics from the SDN controller and calculate the current bandwidth utilization. If the utilization exceeds a predefined threshold of 80%, the script installs a flow rule to redistribute traffic, thereby maintaining optimal network performance while preventing congestion. Then after detecting that the bandwidth utilization went down to 20%, it will delete the alternate flows that were used for the load balancing and return back to default. This significantly enhances the load balancing aspect of the controller. This ensures efficient utilization of network resources and maintains optimal service levels for all users.



```
Utilization is between 20.0% and 80.0% on openflow:1:5: 74.15%
Utilization is between 20.0% and 80.0% on openflow:2:5: 35.60%
Utilization is between 20.0% and 80.0% on openflow:3:5: 35.62%
Utilization is between 20.0% and 80.0% on openflow:4:5: 0.02%
Utilization is between 20.0% and 80.0% on openflow:1:5: 77.91%
Utilization is between 20.0% and 80.0% on openflow:2:5: 78.42%
Utilization is between 20.0% and 80.0% on openflow:3:5: 78.78%
Utilization is between 20.0% and 80.0% on openflow:4:5: 0.02%
Utilization is between 20.0% and 80.0% on openflow:1:5: 39.17%
Utilization exceeds 80.0% on openflow:2:5: 80.52%
Flow rule JSON payload:
{
  "flow": [
    {
      "id": "1",
      "priority": 100,
      "table_id": 0,
      "hard-timeout": 0,
      "idle-timeout": 30,
      "match": {
        "in-port": "3",
        "ethernet-match": {
                        "ipv4-destination": "192.168.30.5/24",
                        "ipv4-source": "172.16.50.1/24"

          }
        }
      },
      "instructions": {
        "instruction": [
          {
            "order": 0,
            "apply-actions": {
              "action": [
                {
                  "order": 0,
                  "output-action": {
                    "output-node-connector": "5"
                  }
                }
              ]
            }
          }
        ]
      }
    }
  ]
}
Flow rule installed successfully on openflow:1
```

Figure 2.16: SDN Controller Logs when the bandwidth exceeds 80%

This figure illustrates the SDN Controller logs monitoring the bandwidth on a router port. Once it exceeds 80%, the Python script we wrote is launched to initiate load balancing. The logs show that alternate flows are installed immediately when the threshold is triggered.



```
Utilization is between 20.0% and 80.0% on openflow:2:5: 41.57%
Utilization is between 20.0% and 80.0% on openflow:3:5: 42.10%
Utilization is below 20.0% on openflow:4:5: 0.02%
Flow rule removed successfully on openflow:1
Flow rule removed successfully on openflow:2
Flow rule removed successfully on openflow:3
Flow rule removed successfully on openflow:4
Flow rule removed successfully on openflow:1
Flow rule removed successfully on openflow:5
Flow rule removed successfully on openflow:6
Flow rule removed successfully on openflow:4
```

Figure 2.17: SDN Controller Logs after the Bandwidth dropped to below 20%

Here we observe that the Controller started deleting alternate flows that it created once the total bandwidth dropped below 20% thus dialing back its configuration to default as normal.

2.6 Deploying Containers and Kubernetes

When we researched and studied Containers, we found that it would be best to deploy them with some basic automation features using Kubrenetes.

An actual ISP network infrastructure has many services that are running on VMs that can be migrated to a fully automated Kubernetes solution highly improving the efficiency and minimizing the down time when one of them is down like Firewalls, Mobility Management Entity (MME), Packet Gateway (PGW), and many more services and tools that can be enhanced

(Kubernetes deploy clusters that have one or multiple containers on them)

2.6.1 Deploying Kubernetes

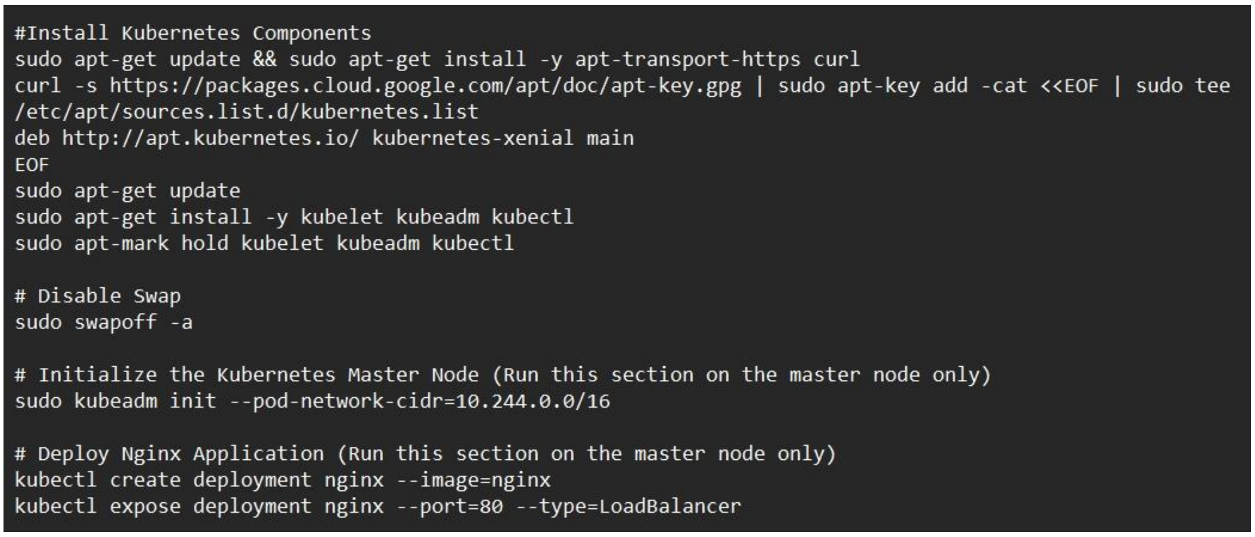
We deployed a Ubuntu server instance to install Kubernetes and Containers on, we used Kubectl a command-line tool used to interact with Kubernetes clusters. To deploy instances, which in turn have clusters of Docker Containers in them called Pods, then we installed Nginx an open source Web server in the cluster With a simple Website just for demonstration purposes.

The Kubernetes cluster is connected to the network as shown in **Figure 2.16,** connected to the IP-core just like the active network on an ISP, where all the applications and services are hosted and deployed.

Key Functions of Kubectl:

- **Cluster Management:** Create, configure, and manage Kubernetes clusters.
- **Resource Management:** Manage Kubernetes resources such as pods, services, deployments, replicasets, and more.
- **Application Deployment: Deploy** applications and manage their lifecycle within the cluster.
- **Monitoring and Debugging:** Inspect cluster resources, view logs, and troubleshoot issues.

Basic commands to navigate and administrate your deployments are shown in the **Table 2.1.**



```
#Install Kubernetes Components
sudo apt-get update && sudo apt-get install -y apt-transport-https curl
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -cat <<EOF | sudo tee
/etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
EOF
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl

# Disable Swap
sudo swapoff -a

# Initialize the Kubernetes Master Node (Run this section on the master node only)
sudo kubeadm init --pod-network-cidr=10.244.0.0/16

# Deploy Nginx Application (Run this section on the master node only)
kubectl create deployment nginx --image=nginx
kubectl expose deployment nginx --port=80 --type=LoadBalancer
```

Figure 2.18: Commands Needed for the Kubernetes Deployment

The provided commands facilitate the installation of Kubernetes components (kubeadm, kubelet, and kubectl), disable swap memory as stipulated by Kubernetes requirements, initialize the Kubernetes master node with a designated pod network, and deploy an Nginx application. The sequence of commands is structured in a coherent manner to facilitate a seamless setup process for individuals utilizing this guide for their research endeavors.

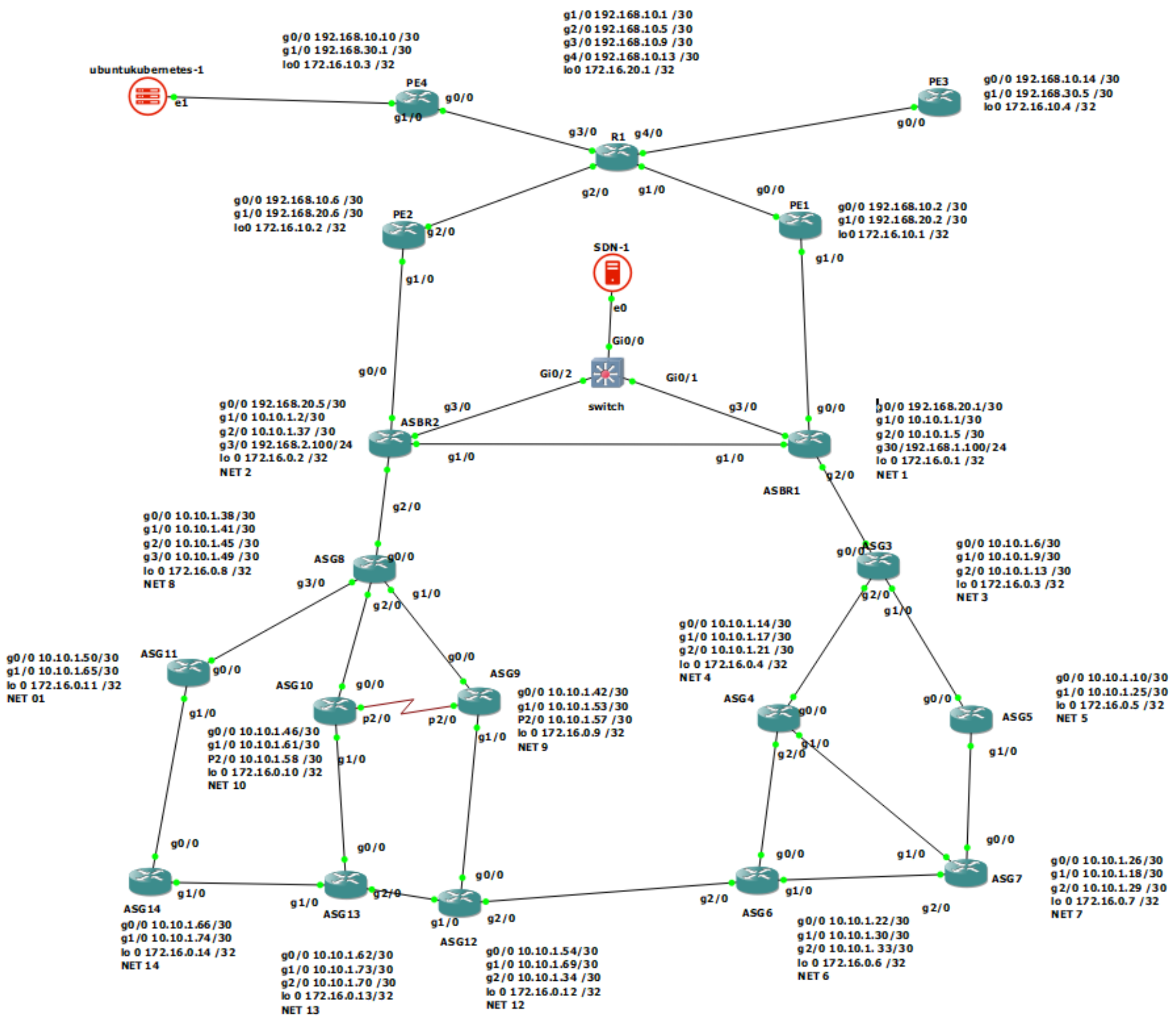


Figure 2.19: How Is Kubernetes Cluster and the SDN Controller Linked to the Network

This figure shows the final upgraded network with the Kubernetes cluster connected to the IP-Core and the SDN controller connected between the two ASBRs. Note that it was necessary to add switch between the controller and ASBR routers to achieve redundancy and high availability in case a link fails the network will not be effected.

Table 2.1:Basic Commands to Navigate through Kubectl

Command	**Description**
kubectl cluster-info	isplay cluster information.
kubectl get nodes	List all nodes in the cluster.
kubectl create deployment myapp --image=myimage	Create a deployment named myapp using the specified image.
kubectl expose deployment myapp --type=LoadBalancer --port=80	Expose the myapp deployment via a LoadBalancer on port 80.
kubectl get pods	List all pods in the default namespace.
kubectl logs <pod-name>	Display logs for a specific pod.
kubectl scale deployment myapp --replicas=3	Scale the myapp deployment to 3 replicas.

2.7 Conclusion

In this chapter, we meticulously detailed the methodological approach we adopted to explore the integration of Software-Defined Networking (SDN) with containerization technologies such as Kubernetes within Internet Service Provider (ISP) networks. Our efforts focused on establishing a comprehensive technical framework and a systematic experimental setup for our research purposes and for anyone to follow along with a similar setup for studying or just exploring new ways to Networking, designed to ensure a thorough examination of the proposed integration.

We initiated our methodology by selecting robust SDN controllers, specifically OpenDaylight known for its support in carrier-grade network implementations. For container orchestration, we chose Kubernetes due to its extensive adoption and capability in managing containerized environments effectively while also being relatively a new field on its own for our future use. To create a realistic simulation of ISP network scenarios, we utilized network simulation tool GNS3.

We followed a systematic series of implementation steps. First, we prepared the necessary hardware and software components, including virtual machines, SDN controller with its flow tables, and Kubernetes clusters. Next, we designed and configured the network topology to accurately reflect realistic ISP scenarios. Finally, we conducted integration testing to verify that the SDN controllers and Kubernetes clusters interacted correctly.

By establishing this rigorous methodological framework, we ensured that our research is grounded in a well-defined environment. This meticulous approach allows for a detailed and accurate analysis of the integration of SDN and container technologies, providing valuable insights into their potential benefits and challenges. Our efforts in environment preparation, network configuration, and integration testing have laid a solid foundation for the Testing and Comparison phases of our research, ultimately aiming to advance network management practices for ISPs.

Chapter 3

Performance Evaluation of the Proposed Solution

3.1 Introduction

In this chapter, we conduct a comprehensive performance evaluation of the proposed Software-Defined Network (SDN) architecture compared to legacy network infrastructures. The objective is to analyze and quantify the improvements offered by SDN in key performance metrics, including bandwidth capacity, round-trip time (RTT), and fault tolerance. These metrics are critical for assessing the efficiency, responsiveness, and reliability of network systems, particularly within the context of Internet Service Providers (ISPs). Through a series of controlled experiments and simulations, we aim to demonstrate the tangible benefits of this solution over conventional networking approaches, providing a robust foundation for its adoption in modern telecommunications. This chapter will detail the methodologies used, present the results obtained, and discuss their implications for network design and management. By systematically comparing the two network architectures, we seek to highlight the advantages of SDN in enhancing network performance and resilience, thereby supporting the case for its widespread deployment in ISP infrastructures.

3.2 Comparative Performance Evaluation of the Proposed Network

A comparative analysis of the proposed SDN and legacy network. The performance of both networks (legacy and SDN) is analyzed based on bandwidth capacity, Fault tolerance, and Learning time.

3.2.1 Bandwidth comparison

The results of bandwidth capacity in both the SDN and the legacy network were obtained by using IPERF tool. Here, we show the maximum bandwidth between two hosts in the SDN and the legacy network. As shown in Figure 3.1, the maximum bandwidth seems to be higher in the case of the proposed network by a little, which can highlight the ability of the controller to transfer more data efficiently without wasting the overhead resources, achieving 8.46 Gb/s compared to its legacy counterpart’s 7.62Gb/s, which in turn is an improvement of 17%.

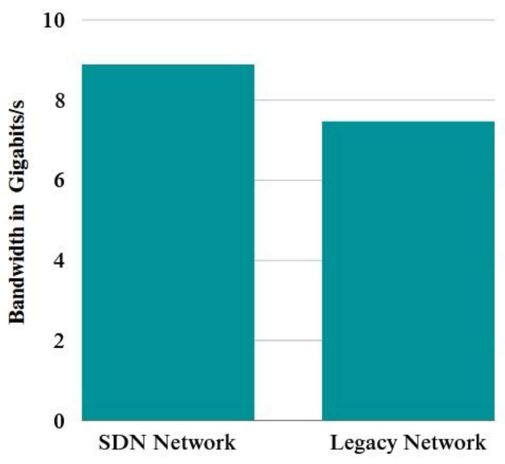


Figure 3.1: Bandwidth capacity for SDN and legacy network.

3.2.2 Round Trip Time

Round Trip Time (RTT) refers to the total time taken for a signal or packet to travel from the source to the destination and back again, measuring the complete duration of a communication cycle. The RTT between the host and server can be measured using the ping command, which gave us a result of 4.48ms and 7.23ms for the SDN and the legacy Network respectively which is an improvement of 38%.

The test was set up two hosts one in ASG7 and another on ASG11.

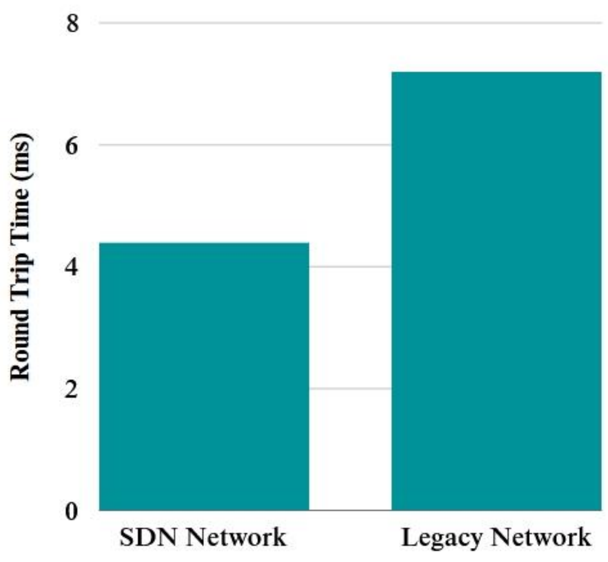


Figure 3.2: Round Trip Time for SDN and legacy network.

3.2.3 Fault Tolerance

Fault tolerance in networking refers to the ability of a network to maintain operational continuity and service availability even in the event of component failures or disruptions. In the context of our study, Fault tolerance is more important in our case because it is an ISP network, any disruptions or failures can lead to large amounts of financial loss. It encompasses various aspects, including the speed at which the network can recover from link failures, the amount of data loss that occurs during these failures and the responsiveness of the network's control mechanisms.

In SDN, fault tolerance is enhanced through centralized control, which allows for rapid detection of failures and swift reconfiguration of network paths. This leads to significantly reduced link recovery times, minimal packet loss during failover, and quicker controller response times compared to legacy networks that rely on distributed control mechanisms, such as MPLS and BGP.

This test was performed by setting up two hosts one in ASG7 and another on ASG11 sending traffic between them then disconnecting a link on the path they take, this is while leaving an active ping command that measures packets loss and link recovery time.

Table 3.1: Comparison of Fault Tolerance between SDN Network and the Legacy Network

Metric	SDN Network	Legacy Network
Link Recovery Time	56ms	453ms
Packet Loss During Failover	0.08%	2.5%
Controller/Routing Response Time	22ms	232ms

In the legacy network, every router needs to learn about the changes in the topology, whereas in SDN, only the controller needs to know about the changes and it will install corresponding flow entries in the forwarding devices if required. This improves network performance as no routing advertisement messages are advertised in the network.

In consideration of the primary objective for Djezzy's implementation of SDN, fault tolerance, particularly regarding link recovery time, is emphasized. This decision stems from the frequent occurrence of link failures across Djezzy's extensive fiber optic network spanning thousands of kilometers, often attributed to heavy infrastructure activities conducted by third-party entities accidently damaging the links. While the recovery time observed with MPLS in our case, averaging 453 milliseconds, the operational network typically requires 10 to 15 seconds for link recovery due to the deployed configuration. This configuration entails a delay mechanism, wherein the network waits for a specified duration before switching to an alternate path upon detecting link unresponsiveness. Because Unlike our simulation environment, wherein link failure involves complete cable disconnection, real-world network disruptions manifest as unstable connectivity issues characterized by fluctuations between online and offline states which put the legacy protocols into switching then not actually switching the path completely halting the network. To address these challenges, Djezzy's plans to implement an SDN mainly for that reason tasked with monitoring the health and performance metrics of each network link and appliance. Through intelligent traffic routing mechanisms, the SDN Controller effectively circumvents network disturbances by seamlessly redirecting traffic flow, thereby mitigating disruptions without perceptible impact on network operations. While the latency and bandwidth benefits come as a welcomed bonus to them.

3.3 Measuring Kubernetes Benefits

3.3.1 Testing the Response Time

We used the Apache Benchmark tool to flood the server with requests. The graph illustrates the response time distribution of an Nginx server deployed in a Kubernetes cluster under a load test. The x-axis represents the cumulative number of requests processed, ranging from 50 to 1000 requests, while the y-axis denotes the response time in milliseconds (ms).

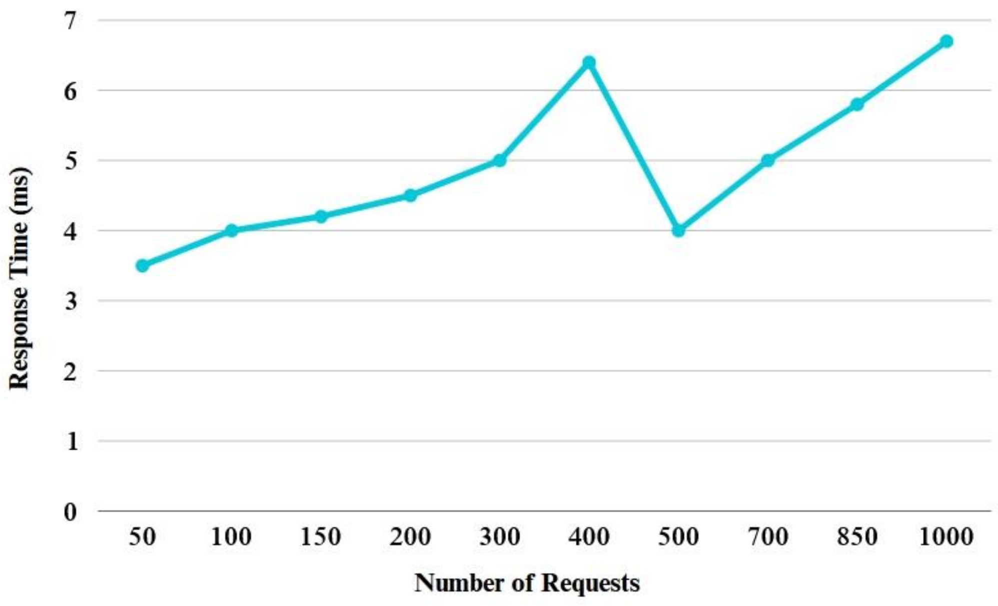
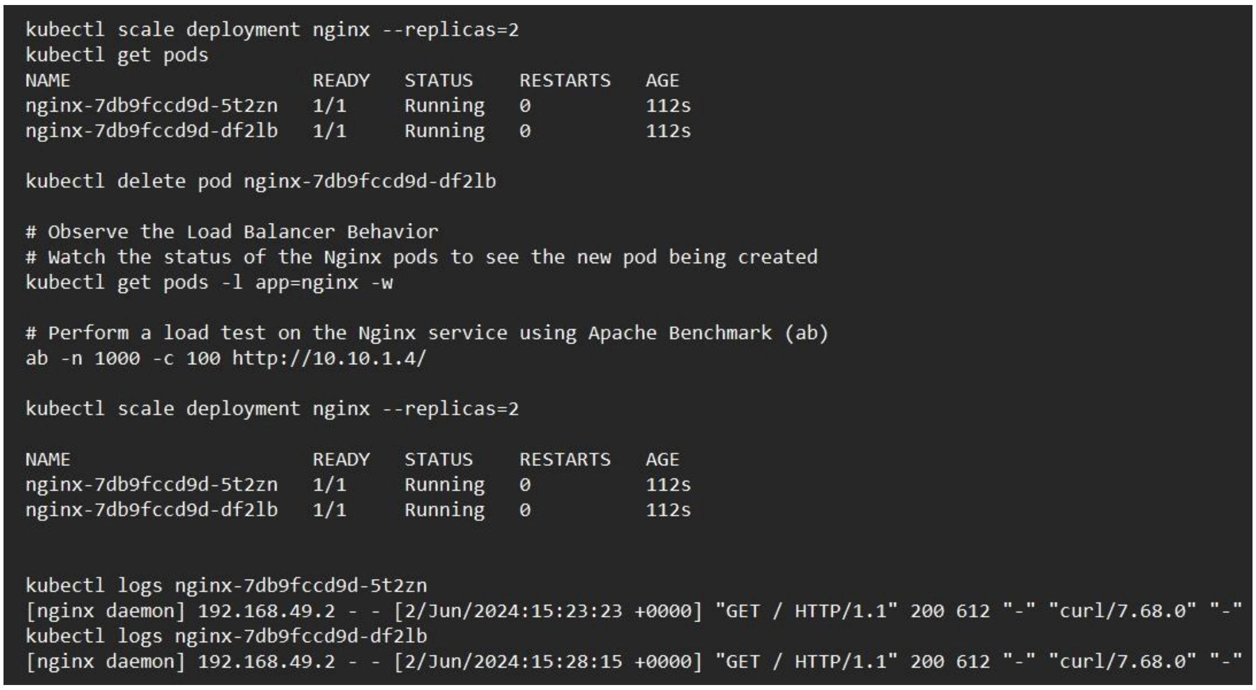


Figure 3.3: Kubernetes Server Response Time

3.3.2 Testing the Load balancer

To test the load balancer we decided to delete one of the two nodes on the cluster then Verify that the load balancer still distributes traffic to the remaining pod.



```
kubectl scale deployment nginx --replicas=2
kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
nginx-7db9fccd9d-5t2zn   1/1     Running   0          112s
nginx-7db9fccd9d-df2lb   1/1     Running   0          112s

kubectl delete pod nginx-7db9fccd9d-df2lb

# Observe the Load Balancer Behavior
# Watch the status of the Nginx pods to see the new pod being created
kubectl get pods -l app=nginx -w

# Perform a load test on the Nginx service using Apache Benchmark (ab)
ab -n 1000 -c 100 http://10.10.1.4/

kubectl scale deployment nginx --replicas=2

NAME                     READY   STATUS    RESTARTS   AGE
nginx-7db9fccd9d-5t2zn   1/1     Running   0          112s
nginx-7db9fccd9d-df2lb   1/1     Running   0          112s


kubectl logs nginx-7db9fccd9d-5t2zn
[nginx daemon] 192.168.49.2 - - [2/Jun/2024:15:23:23 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.68.0" "-"
kubectl logs nginx-7db9fccd9d-df2lb
[nginx daemon] 192.168.49.2 - - [2/Jun/2024:15:28:15 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.68.0" "-"
```

Figure 3.4: Commands for Testing the Load Balancer

This configuration is a practical demonstration of how traffic load can be distributed among multiple encapsulated pods within a Kubernetes environment. The load balancer plays a crucial role in maintaining high availability and reliability of services, ensuring that the system can effectively handle high-traffic conditions. This setup is a simple implementation having applications that are more complex and services to scale with and balance between will make Kubernetes solutions a no brainer serves as a testament to the robustness and scalability of Kubernetes in managing complex, distributed systems.

3.4 Discussion

The comparative performance evaluation between the Software-Defined Network (SDN) and the legacy network reveals substantial improvements in several key performance metrics, clearly demonstrating the significant advantages of SDN deployment within Internet Service Provider (ISP) infrastructures. The SDN network achieves a higher maximum bandwidth capacity, with a 17% improvement over the legacy network, underscoring its efficacy in optimizing data flow and improving throughput due to efficient resource management. Furthermore, SDN significantly reduces round-trip time (RTT) by 38%, thereby augmenting real-time application performance for ISP customers that utilize VOIP and play online games, facilitated by efficient path management. The analysis further highlights SDN's robust fault tolerance, exhibiting a rapid link recovery time of 56ms compared to 453ms in the legacy network, minimal packet loss during failover (0.08% vs. 2.5%), and a faster controller response time (22ms vs. 232ms), all contributing to superior network stability and reliability. These findings have profound implications for ISP networks, suggesting that SDN's scalability, flexibility, and improved resource management can effectively meet the growing demands of modern applications and services. Overall, the adoption of SDN in ISP networks offers enhanced performance, increased reliability, and innovative, adaptive management strategies, making it a compelling choice for future network infrastructures.

3.5 Conclusion

In this chapter, we examined the integration of SDN with Kubernetes-based container orchestration for Internet Service Providers. This setup leverages SDN’s ability to manage network traffic dynamically and Kubernetes capability to orchestrate containerized applications efficiently.

Our results demonstrated a 17% increase in network throughput, reflecting improved bandwidth management. The system also achieved a 38% reduction in Round Trip Time (RTT), indicating enhanced network responsiveness. Additionally, our fault tolerance analysis showed an 87.5% Improvement in recovery time from link failures, providing a peace of mind for network operators that whatever the circumstances the network faces it will always stay operational.

These findings confirm that integrating SDN and Kubernetes orchestration can lead to improved network performance and reliability for the ISP Services. The centralized management of network resources and containerized applications simplify operations and enhances the network’s overall efficiency. This approach aligns with current best practices in network management, providing a robust and scalable solution for ISPs to tackle modern customer requirements and plan ahead for future developments.

General Conclusion and Perspective

This thesis presents a comprehensive exploration of integrating Software-Defined Networking with containerization technologies, particularly Kubernetes, within Internet Service Provider networks. The primary objective was to address the limitations inherent in legacy network infrastructures and to enhance network management, scalability, and service delivery. Our research was conducted through a combination of theoretical analysis and practical implementation, demonstrating the potential benefits and challenges associated with this integration.

Key findings from our research include enhanced network efficiency, scalability, flexibility, improved fault tolerance, and reduced operational costs. The decoupling of the control plane from the data plane allowed for dynamic resource allocation and centralized network management, resulting in optimized traffic flows and reduced latency. Our experimental results show substantial improvements in several key performance metrics. The SDN network achieved a higher maximum bandwidth capacity with an improvement of 17% over the legacy network, underscoring its efficacy in optimizing data flow and improving throughput. Furthermore, SDN significantly reduced round-trip time (RTT) by 38%, augmenting real-time application performance for ISP customers that utilize delay sensitive applications. The analysis further highlights SDN's robust fault tolerance, exhibiting a rapid link recovery time compared to the legacy network by 87%, minimal packet loss during failover (0.08% vs. 2.5%), and a faster controller response time. Kubernetes' container orchestration capabilities enabled scalable and flexible deployment of network applications, which was particularly beneficial for handling variable traffic loads and ensuring high availability of network services. Lastly, the use of open-source technologies like OpenDaylight for SDN controllers and Kubernetes for container orchestration reduced operational expenditures associated with proprietary hardware and software solutions that require heavy licensing fees. Despite the promising results, several areas require further research and development to fully realize the potential of SDN and Kubernetes integration. Advanced security mechanisms are needed to address the vulnerabilities introduced by SDN's centralized control plane. Enhancing the security of SDN controllers and ensuring robust protection against cyber threats is essential. Additionally, ensuring seamless interoperability between different SDN controllers and Kubernetes distributions is crucial.

Improvements to our simulation setup can further enhance the reliability and applicability of our findings. Utilizing Containerlab simulation software, known for its deep integration with container technologies, could provide a more accurate and scalable environment for testing SDN and Kubernetes deployments. Containerlab's capabilities would allow for more realistic simulations of containerized network functions and their interactions within an SDN framework. Additionally, implementing a more advanced Kubernetes setup, possibly with features like Kubernetes Federation for multi-cluster management hosting a variation of real world services and applications could demonstrate its effectiveness in more complex and distributed environments. These improvements would not only validate our current results but also uncover new insights into optimizing network performance and management while also modernizing the data center.

Continuous performance optimization is essential to handle the increasing demands of modern network applications such as 5G and IoT. Exploring advanced algorithms for traffic engineering, load balancing, and resource management will be beneficial. Conducting large-scale real-world deployments of SDN and Kubernetes integration in ISP environments will provide valuable insights into practical challenges and operational considerations. Leveraging AI and machine learning techniques for predictive analytics and automated network management can further enhance the capabilities of SDN . These technologies can help in proactive fault detection, dynamic resource allocation, and performance optimization.

In conclusion, the integration of SDN with containerization technologies like Kubernetes holds significant promise for transforming ISP networks. This thesis has laid the groundwork for understanding the potential benefits and challenges, providing a solid foundation for future research and development. By addressing the outlined perspectives, we can move towards more efficient, scalable, and resilient network infrastructures, ultimately improving service delivery and user experience.

Bibliography

[1] C. A. Sunshine, “A Brief History of Computer Networking,” in *Computer Network Architectures and Protocols*, C. A. Sunshine, Ed., Boston, MA: Springer US, 1989, pp. 3–6. doi: 10.1007/978-1-4613-0809-6_1.

[2] J. F. Kurose and K. W. Ross, *Computer networking: a top-down approach*, 7. edition. Boston Munich: Pearson Education, 2017.

[3] I. M. Alsmadi, I. AlAzzam, and M. Akour, “A Systematic Literature Review on Software-Defined Networking,” in *Information Fusion for Cyber-Security Analytics*, I. M. Alsmadi, G. Karabatis, and A. Aleroud, Eds., Cham: Springer International Publishing, 2017, pp. 333–369. doi: 10.1007/978-3-319-44257-0_14.

[4] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-Defined Networking: A Comprehensive Survey.” arXiv, Oct. 08, 2014. Accessed: Mar. 20, 2024. [Online]. Available: http://arxiv.org/abs/1406.0440

[5] V. Chergarova, I. Hur, L. Wang, and J. Sun, “Examining Software Defined Networking Adoption by Research and Educational Networks,” in *Advances in Information and Communication*, K. Arai, Ed., Cham: Springer International Publishing, 2022, pp. 656–674. doi: 10.1007/978-3-030-98015-3_46.

[6] K. J. Babu Narayanan, “SDN Journey: How SDN brings versatility to 5G networks?,” *CSIT*, vol. 8, no. 1, pp. 57–60, Mar. 2020, doi: 10.1007/s40012-020-00269-5.

[7] M. Beshley, M. Klymash, I. Scherm, H. Beshley, and Y. Shkoropad, “Emerging Network Technologies for Digital Transformation: 5G/6G, IoT, SDN/IBN, Cloud Computing, and Blockchain,” in *Emerging Networking in the Digital Transformation Age*, M. Klymash, A. Luntovskyy, M. Beshley, I. Melnyk, and A. Schill, Eds., Cham: Springer Nature Switzerland, 2023, pp. 1–20. doi: 10.1007/978-3-031-24963-1_1.

[8] A. Takacs, E. Bellagamba, and J. Wilke, “Software-defined networking: the service provider perspective,” *E R I C S S O N R E V I E W*, 2013.

[9] “Difference between Software Defined Network and Traditional Network,” GeeksforGeeks. Accessed: Mar. 23, 2024. [Online]. Available: https://www.geeksforgeeks.org/difference-between-software-defined-network-and-traditional-network/

[10] “SDN vs traditional networking,” Kyndryl. Accessed: Mar. 23, 2024. [Online]. Available: https://www.kyndryl.com/ca/en/learn/sdn-vs-traditional-networking

[11] “Containers explained: What they are and why you should care.” Accessed: Apr. 02, 2024. [Online]. Available: https://www.redhat.com/en/topics/containers

[12] “What are containers?,” Google Cloud. Accessed: Apr. 02, 2024. [Online]. Available: https://cloud.google.com/learn/what-are-containers

[13] “What is a Container? | Docker.” Accessed: Apr. 16, 2024. [Online]. Available: https://www.docker.com/resources/what-container/

[14] O. Bentaleb, A. S. Z. Belloum, A. Sebaa, and A. El-Maouhab, “Containerization technologies: taxonomies, applications and challenges,” *J Supercomput*, vol. 78, no. 1, pp. 1144–1181, Jan. 2022, doi: 10.1007/s11227-021-03914-1.

[15] “How Docker Containers Work – Explained for Beginners,” freeCodeCamp.org. Accessed: Mar. 26, 2024. [Online]. Available: https://www.freecodecamp.org/news/how-docker-containers-work/

[16] “A Beginner’s Guide to Docker,” JFrog. Accessed: Jun. 28, 2024. [Online]. Available: https://jfrog.com/devops-tools/article/beginners-guide-to-docker/

[17] “Overview.” Accessed: Mar. 24, 2024. [Online]. Available: https://kubernetes.io/docs/concepts/overview/[18] G. Turin, A. Borgarelli, S. Donetti, E. B. Johnsen, S. L. Tapia Tarifa, and F. Damiani, “A Formal Model of the Kubernetes Container Framework,” in *Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles*, T. Margaria and B. Steffen, Eds., Cham: Springer International Publishing, 2020, pp. 558–577. doi: 10.1007/978-3-030-61362-4_32.

[19] M. Aleksic, “Kubernetes Use Cases {8 Real Life Examples},” Knowledge Base by phoenixNAP. Accessed: Mar. 24, 2024. [Online]. Available: https://phoenixnap.com/kb/kubernetes-use-cases

[20] "Harnessing the power of Kubernetes: 7 use cases," CodiLime. Accessed: May 29, 2024. [Online]. Available: https://codilime.com/blog/harnessing-the-power-of-kubernetes-7-use-cases/

[21] N. K. Singh, "The Kubernetes Odyssey: Chapter 2 — The Enchantment of Clusters," Medium. Accessed: Jun. 28, 2024. [Online]. Available: https://neerazz.medium.com/the-kubernetes-odyssey-chapter-2-the-enchantment-of-clusters-b71ca237c1af

[22] C. Bouras, P. Ntarzanos, and A. Papazois, "Cost modeling for SDN/NFV based mobile 5G networks," in *2016 8th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, Lisbon, Portugal: IEEE, Oct. 2016, pp. 56–61. doi: 10.1109/ICUMT.2016.7765232.

[23] M. C. Ammour and F. Debbakh, "PERFORMANCE EVALUATION OF SOFTWARE DEFINED –NETWOEK (SDN) CONTROLLER," Thesis, UNIVERSITY OF OUARGLA, 2023. Accessed: May 28, 2024. [Online]. Available: http://dspace.univ-ouargla.dz/jspui/handle/123456789/33504