UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

AUTOMATED RESOURCE DEPLOYMENT IN OPTICALLY INTERCONNECTED DATA CENTER INFRASTRUCTURES FOR IIOT SERVICES

A Master's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Luis Antonio Lino Vivanco

In partial fulfilment

of the requirements for the degree of

MASTER DEGREE IN ADVANCED TELECOMMUNICATION TECHNOLOGIES (MATT)

Advisors: Albert Pagès Cruz, Salvatore Spadaro

Barcelona, September 2023

Title of the thesis: Automated resource deployment in optically interconnected data center infrastructures for IIoT services.

Author: Luis Antonio Lino Vivanco

Advisors: , Albert Pagès Cruz, Salvatore Spadaro

Abstract

The relentless expansion of the Industrial Internet of Things (IIoT) has revolutionized modern data center architectures, introducing unparalleled challenges in resource management and service optimization. This thesis investigates the efficacy of automated resource deployment in data centers interconnected with optical technologies to augment IIoT service efficiency.

By merging Python-driven latency measurement and network route optimization with the agility of Virtual Machine (VM) orchestration via OpenStack, this research delineates a relevant methodology to mitigate latency issues and provide resource allocation. Consequently, the principal outcome of this study is an innovative Python tool that harnesses optical technologies to address these pivotal concerns. Employing the Mininet emulator demonstrates promising outcomes in enhancing responsiveness, flexibility, and judicious resource distribution in IIoT-centric data centers.

Ultimately, this research underscores the necessity of innovatively leveraging optical technologies, performance measurements (KPIs), automation, and orchestration, thereby establishing a foundational framework for the next generation of optically-enhanced IIoT data centers.

Dedication:

To my parents,

For your unwavering belief and enduring support,

This Master's Thesis stands as a testament to the values you instilled in me.

Thank you for lighting the path and guiding my journey.

Acknowledgements

At this juncture, I find it paramount to express my deepest gratitude to those who have contributed, in both big and small ways, to the completion of this Master's Thesis.

First and foremost, I extend my sincere appreciation to my advisors, Albert Pagès and Salvatore Spadaro, for their invaluable guidance, insightful critiques, and unyielding patience throughout the course of this research. Their expertise and wisdom have been instrumental in shaping my research journey.

Revision history and approval record

Revision	Date	Purpose
0	10/04/2023	Document creation
1	31/07/2023	Document revision
2	28/08/23	Document revision

Written by:		Reviewed and approved by:	
Date	28/08/2023	Date	28/08/2023
Name	Luis Lino Vivanco	Name	Albert Pagès
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables

1. Introduction

In the modern era, as industries rapidly digitize and automate processes, the world stands on the cusp of an unprecedented technological revolution, driven predominantly by the Industrial Internet of Things (IIoT). IIoT encompasses a myriad of technologies, including automated factory floors and smart infrastructure, that leverage interconnected devices and systems to optimize operations and improve efficiency.

Specifically, automated factory floors represent a critical component of IIoT, wherein machines and devices communicate and coordinate with each other in real-time to optimize production and minimize downtime. Also, the evolution of networks, from 5G to 6G, not only facilitates the emergence of these advanced IIoT applications but is an intrinsic requirement for their successful implementation. Enhanced connectivity, higher bandwidth, and lower latency, hallmarks of the transition from 5G to 6G, are critical enablers of IIoT applications that demand real-time responsiveness and seamless communication between devices.

1.1 Actual Problematics of IIoT Deployments and Operations:

The modern landscape of IIoT is one marked by a myriad of complexities and challenges. As industries increasingly integrate IIoT services into their operations, the demands on data center infrastructures have grown exponentially. This surge in demand necessitates a range of requirements that must be met to ensure optimal performance and efficiency.

Firstly, the dynamic nature of IIoT workloads necessitates an agile resource deployment mechanism that can adapt in real-time. Traditional static resource allocation strategies are ill-equipped to handle the fluctuations in demand encountered in IIoT applications, often leading to suboptimal performance or resource wastage.

Secondly, the integration of cloud computing resources via seamless VM orchestration is a critical requirement. The management of virtual resources in the cloud, particularly in environments with optical interconnections, is a complex task that necessitates seamless orchestration to optimize resource allocation and utilization.

Thirdly, network latency and route link optimization are of paramount importance. High network latency can severely impact the performance of time-sensitive IIoT applications, making it essential to optimize network routes and manage latency efficiently.

Lastly, the ability to dynamically respond to IIoT workloads is a critical requirement. IIoT applications often have fluctuating workloads, and a system that can dynamically respond to these changes is crucial for maintaining optimal performance.

In summary, the actual problematic of IIoT deployments and operations centers around the need for agile resource deployment, seamless VM orchestration, efficient network latency and route link optimization, and dynamic workload response. Addressing these challenges is crucial for optimizing IIoT service performance in data center environments.

1.2 Requirements and Specifications:

The challenges associated with the actual problematic of IIoT deployments and operations, as discussed in the previous section, necessitate a set of specific requirements and specifications to address these challenges effectively:

- **Agile Resource Deployment:** A mechanism that can dynamically allocate resources in real-time is essential. This mechanism must be able to respond to the fluctuating demands of IIoT applications by provisioning and de-provisioning resources as needed to optimize performance and minimize resource wastage.
- **Seamless VM Orchestration:** A seamless integration with VM orchestration is required to leverage the advantages of cloud computing. This includes the ability to dynamically provision and manage virtual resources in environments marked by optical interconnections, a crucial aspect given the complexities associated with managing optical networks.
- **Efficient Network Latency and Route Link Optimization:** A system capable of managing network latency and optimizing network routes is essential. This system must be able to analyze network traffic in real-time, identify bottlenecks, and dynamically adjust network routes to minimize latency and optimize data transfer.
- **Dynamic Workload Response:** A system capable of measuring, analyzing, and dynamically responding to IIoT workloads is required. This system must be able to monitor workload patterns, predict fluctuations, and dynamically adjust resource allocation to maintain optimal performance.

Together, these requirements and specifications form the foundation for an innovative approach to address the pressing challenges associated with IIoT deployments and operations in modern data center environments.

1.3 Statement of Purpose:

The primary objectives of this research, in light of the detailed requirements and specifications laid out earlier, are as follows:

- **Design a Python-Driven System for Latency Measurement and Route Optimization:** Develop a Python-driven system capable of measuring network latency and optimizing network routes in real-time. This involves the creation of a custom Python sensor that can actively measure the latency between various host pairs within a network, identify bottlenecks, and dynamically optimize network routes to ensure efficient data transfer. This objective directly addresses the need for efficient network latency and route link optimization, as specified in the requirements.
- **Integration of Automated Resource Deployment with VM Orchestration:** Develop a mechanism that can seamlessly integrate automated resource deployment with VM orchestration in environments marked by optical interconnections. This involves leveraging the capabilities of OpenStack for dynamic VM provisioning and management, and integrating it with the Python-driven system developed in the first objective. This objective directly addresses the need for agile resource deployment and seamless VM orchestration, as specified in the requirements.

- **Evaluation of Proposed Solution:** Evaluate the efficacy of the proposed solution in enhancing IIoT data center performance. This involves conducting a series of experiments using the Mininet network emulator and the developed Python sensor to measure the improvement in network latency and route link optimization efficiency in a data center environment. This objective directly addresses the need for a dynamic workload response system, as specified in the requirements.

Ultimately, the successful realization of these objectives will contribute to the development of a comprehensive solution that addresses the pressing challenges associated with IIoT deployments and operations in modern data center environments.

1.4 Methods and Procedures:

This work is built upon the foundational knowledge of IIoT data center management, leveraging tools and methodologies that have stood the test of rigorous academic scrutiny. The methods and procedures used in this research include:

- **Foundational Principles:** Utilizing established principles of IIoT data center management to guide the research.
- **Mininet Network Emulator:** Employing the Mininet network emulator as the primary testing ground. This involves setting up a network, running tests, and measuring performance metrics such as latency, using the Python sensor developed.
- **OpenStack for VM Deployment:** Utilizing OpenStack for VM deployment. This involves setting up VMs, deploying resources, and managing network configurations.
- **Python Sensor Development:** Developing a novel Python sensor to measure network latency and optimize routes. This involves designing, coding, and testing the sensor in the Mininet and OpenStack environments.

Importantly, the unique Python sensor, developed during this research, acts as a linchpin, bridging various components. This work, while novel in its approach, draws inspiration and techniques from previous research but pushes boundaries in its application and scope.

1.5 Work Plan:

The research was approached systematically, divided into the following tasks and milestones:

- **Literature Review:** A comprehensive review of current methodologies, technologies, and challenges in IIoT, data center management, and network optimization was conducted to pinpoint gaps and areas for improvement.
- **Python Sensor Development:** The design, development, and testing of a unique Python sensor capable of measuring network latency and optimizing network routes.
- **Integration with Mininet:** The Python sensor was integrated with the Mininet network emulator to establish a controlled emulation environment for testing and evaluation.

- **OpenStack Integration:** The research leveraged OpenStack for dynamic VM deployment, integrating it with the Python sensor and Mininet to create a comprehensive testing environment.
- **Data Collection and Analysis:** Data on network latency, route optimization efficiency, and VM deployment was collected, analyzed, and interpreted to evaluate the performance and efficacy of the proposed solution.

1.6 Gantt Diagram:

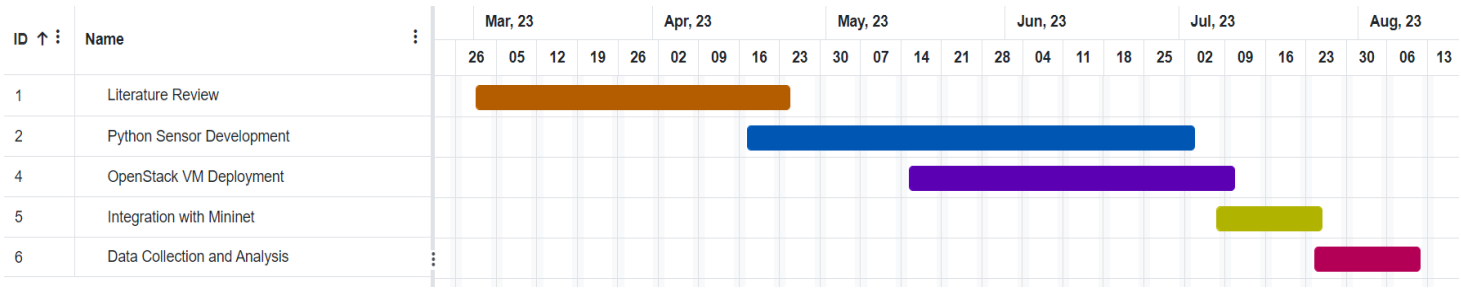

Figure N°1. Gantt Diagram

1.7 Deviations and Eventualities:

While the project largely adhered to the initial plan, some deviations were encountered. The integration with OpenStack proved to be more complex than anticipated, requiring a deeper dive into its functionalities.

Additionally, while the Python sensor was a unique contribution, its scalability in larger real-world data center environments remains a topic for future exploration. Despite these challenges, the project navigated through them, evolving and refining its approach, ensuring the research stayed robust and relevant.

In essence, this research not only aims to address pressing challenges but also stands as a testament to the possibilities that emerge when technology and innovation converge. The following chapters delve deeper into the methodologies, findings, and implications of this research.

2. State of the art of the technology used or applied in this thesis

2.1 Literature Review

Navigating the complex tapestry of technology shaping the foundation of this thesis, it becomes pivotal to dissect the existing literature and comprehend the contemporary technological advancements. This literature review seeks to illuminate the cornerstone technologies that underpin our research, contextualizing them within the broader academic discourse.

2.1.1 Industrial Internet of Things (IIoT):

The Industrial Internet of Things (IIoT), sometimes dubbed as the "Fourth Industrial Revolution", is rapidly changing the landscape of industries across the globe. Characterized by its intelligent interconnectivity between machines and systems, the IIoT is enhancing industry operations in numerous ways.

The term "IIoT" finds its roots in the broader concept of the Internet of Things (IoT), which essentially involves the connection of devices – anything from everyday appliances to industrial machinery – to the internet. What distinguishes IIoT is its specific application in industrial contexts, including manufacturing, logistics, oil and gas, transportation, energy/utilities, mining and metals, aviation, and other industrial sectors.

Several scholars have pointed out the major drivers behind the rapid adoption of IIoT. Key among them are the significant advancements in sensor technology, cloud computing, edge computing, and data analytics. These technologies allow industries to monitor, collect, exchange, analyze, and deliver valuable new insights, leading to smarter, more decision-driven operations.

In a comprehensive study by Gartner, it was estimated that the IIoT would include 5.4 billion IoT devices by 2020, making it a dominant force in the industrial sector. This immense growth has several implications. Firstly, it has resulted in increased automation. Systems can communicate with each other, process vast amounts of data in real-time, and make decisions without human intervention. Furthermore, there's the advantage of predictive maintenance where equipment failures are predicted and mitigated before they even happen, resulting in considerable cost savings.

With the rise of IIoT, there's been a significant increase in the volume of data that industries have to process and manage. This surge has placed enormous demands on data centers. In their seminal work, Khan et al. discussed the evolving role of data centers in the age of IIoT. They emphasized the need for these centers to not just store data, but to rapidly process and analyze it. The rise of edge computing, where data processing occurs closer to where it is generated, is a testament to this shift.

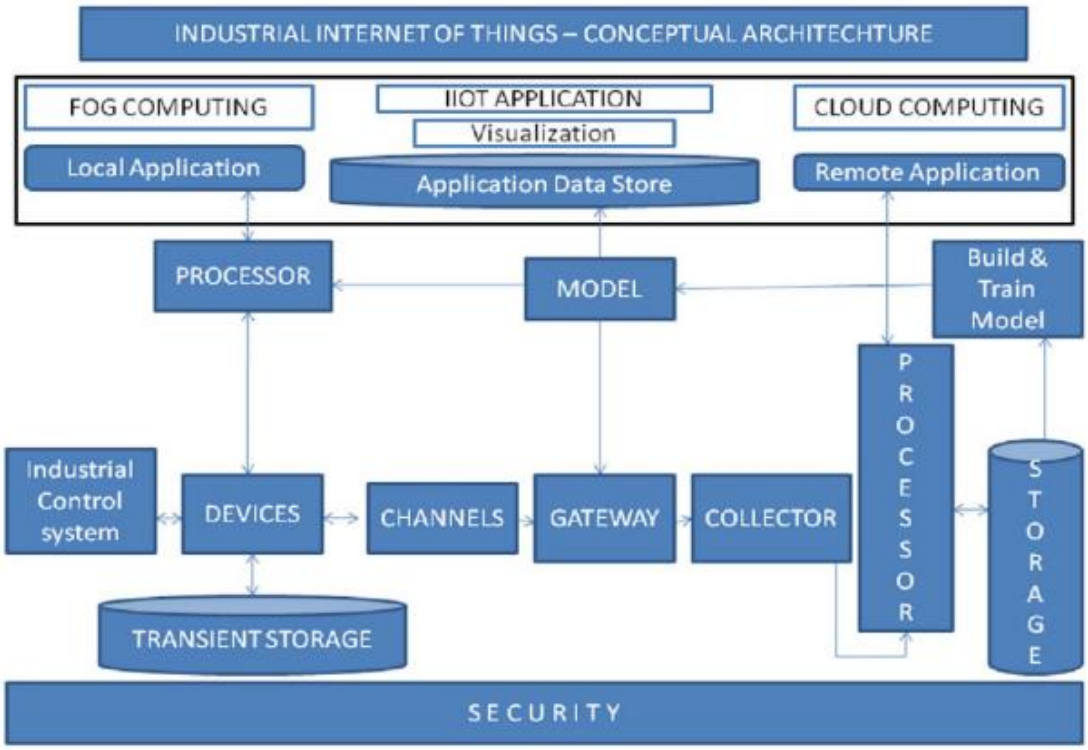

Figure N°2 Theoretical Architechture of IIoT

2.1.2 Optically Interconnected Data Centers

With the surge in data volumes, optically interconnected data centers have gained traction. Scholars and industry experts are increasingly pointing towards the advantages of optical interconnections, particularly in terms of speed, bandwidth, and energy efficiency. Several studies have proposed novel architectures and protocols tailored for these data centers.

The IIoT, characterized by its vast array of interconnected devices and systems, inherently demands robust, high-speed data processing infrastructure.In contrast, optical interconnections, with their inherent high-speed capabilities, are proving to be pivotal in handling IIoT data workloads.

Key advantages of optical interconnections, specifically concerning IIoT-centric data centers, have been delineated in the literature:

- **Speed and Bandwidth:** The data throughput capabilities of optical interconnections are instrumental in ensuring real-time processing of IIoT data, thereby enhancing the responsiveness of IIoT applications.

- **Energy Efficiency:** Energy considerations, are paramount, especially given the always-on nature of IIoT systems. Optical interconnections, with their lower energy footprints, contribute to more sustainable IIoT data centers.

- **Reduced Latency:** The latency-sensitive nature of many IIoT applications, from factory automation to smart grids, mandates minimal data transmission delays. Optical interconnections deliver on this front.

The shift towards optical interconnections in the IIoT context is fostering research into architectures that cater specifically to IIoT workloads.

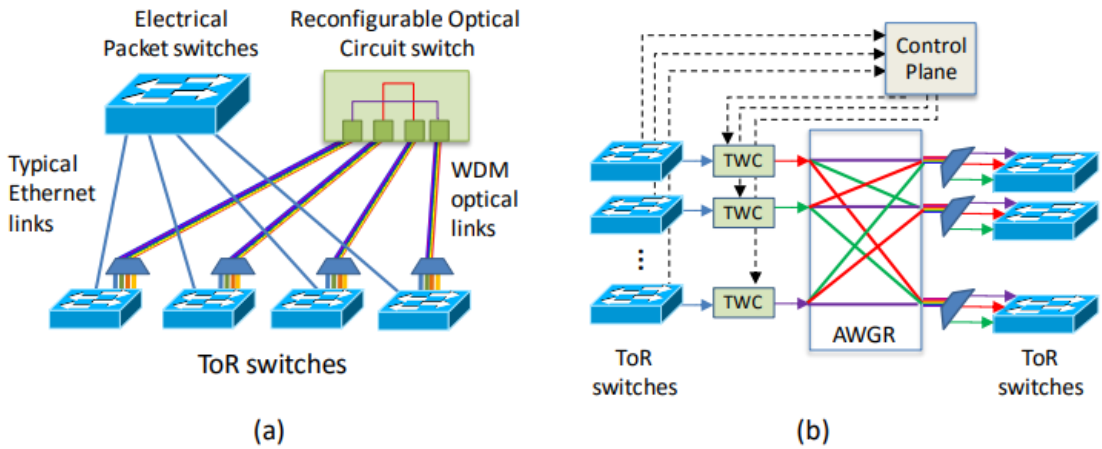

Figure N°3. Proposed Optical architectures: (a) Hybrid network with optical circuit switching and (b) optical packet switching using tunable wavelength converters (TWCs) and arrayed waveguide grating router (AWGR).

2.1.3 Automated Resource and VM Deployment

The modern landscape of data center management is rapidly evolving, with automation at its helm. As data centers grapple with the requirements of the IIoT era, the role of automated resource and VM deployment stands out as a crucial pivot to maintain efficiency and responsiveness.

Historically, data center operations required significant manual intervention, leading to inefficiencies, delays, and human errors.

The impetus for automated resource deployment and VM orchestration in the context of IIoT is underpinned by several key benefits:

- **Real-time Responsiveness:** IIoT applications often necessitate swift data processing. Automation allows data centers to dynamically allocate resources based on real-time needs, enhancing their responsiveness to the demands of IIoT applications.

- **Optimal Resource Utilization:** Traditional data center management often led to underutilization or overprovisioning of resources. Automation ensures optimal allocation, thereby maximizing ROI and reducing resource wastage, which is crucial for the efficient operation of IIoT data centers.

- **Scalability:** With the unpredictable growth of IIoT devices, scalability becomes paramount. Automated systems can expand or contract resource allocation in sync with demand, enabling data centers to adapt to the changing needs of IIoT applications.

- **Real-time Responsiveness:** IIoT applications often necessitate swift data processing. Automated deployment allows data centers to dynamically allocate resources based on real-time needs.

- **Optimal Resource Utilization:** Traditional data center management often led to underutilization or overprovisioning of resources. Automation ensures optimal allocation, thereby maximizing ROI.

- **Scalability:** With the unpredictable growth of IIoT devices, scalability becomes paramount. Automated systems can expand or contract resource allocation in sync with demand.

Furthermore, recent research has underscored the importance of automated VM deployment in optimizing resource utilization and ensuring a seamless user experience. Tools such as Kubernetes, OpenStack, and Docker Swarm have been at the forefront, providing platforms that allow for effective VM deployment. Each tool, with its unique features, aims to simplify the deployment, scaling, and management of containerized applications across clusters of hosts.

Future Trends

The current trajectory suggests an increasing integration of VM deployment and orchestration with artificial intelligence (AI) and machine learning (ML) techniques. These integrations aim to create self-healing infrastructures that can predict failures and auto-mitigate them. Additionally, the advancements in edge computing and the rise of hybrid and multi-cloud environments are driving the evolution of VM orchestration tools. These tools are being developed to manage and coordinate resources across disparate cloud and edge environments seamlessly, which is especially significant for IIoT applications that generate data at the edge and require real-time processing.

Moreover, the integration of AI and ML with VM orchestration is also enabling the development of more intelligent and adaptive resource management systems. These systems can analyze real-time data, predict resource requirements, and dynamically adjust resource allocation to optimize performance and reduce costs.

Finally, it is evident that dynamic VM deployment and orchestration is no longer a luxury but a necessity for modern data center infrastructures, especially those catering to IIoT applications. The dynamic nature of today's applications and services necessitates an equally agile and responsive backend. VM orchestration, integrated with AI and ML techniques, stands tall as the answer to this imperative, enabling more efficient, adaptive, and resilient IIoT data center operations.

2.1.4 Cloud Platforms

The advent of cloud computing has revolutionized the way applications and services are delivered and managed. Various cloud platforms, such as AWS, Azure, Google Cloud Platform, and OpenStack, have emerged as key players in this domain, offering a range of services and tools to manage large-scale deployments, including those in IIoT environments.

Cloud Platforms' Relevance to IIoT

Cloud platforms are especially relevant for IIoT implementations that require agile, scalable, and resilient infrastructure. AWS's IoT Core, for example, enables the connection of IoT devices to the cloud, supporting billions of devices and trillions of messages. Azure IoT Suite provides a collection of preconfigured solutions for quickly starting and scaling IoT

projects. Google Cloud IoT offers fully managed and integrated services for securely connecting, managing, and ingesting IoT data from globally dispersed devices.

In this sense, OpenStack was selected for the present research for its availability in the test bed and its open source nature. Also its got a great advantage due to its modular architecture that offers resource provisioning, auto-scaling, and compatibility with SDN controllers and network function virtualization (NFV). This makes it an ideal choice for managing IIoT-centric data centers.

Challenges and Opportunities

While OpenStack is powerful, its intricate architecture presents challenges, including upgrade complexities, resource fragmentation, and integration issues with legacy systems. However, the OpenStack community is rapidly innovating to address these challenges. The growth of edge computing and IIoT further accentuates OpenStack's relevance as stakeholders seek ways to manage distributed architectures.

In this sense, the journey of cloud platforms, with OpenStack as a key player, exemplifies the digital infrastructures' dynamic evolution. OpenStack's ability to adapt to the ever-changing requirements of the IIoT landscape solidifies its position as a cornerstone in modern data center management. However, it is important to note that other platforms like AWS, Azure, and Google Cloud Platform also play crucial roles in the IIoT ecosystem, each with its unique features and capabilities. For this thesis, OpenStack was selected for its open-source nature and compatibility with the specific requirements of the IIoT application under consideration. Nonetheless, other platforms could have been used depending on the application's specific needs and the organization's preferences.

2.1.5 Network Latency and Jitter Sensing

In the evolving landscape of Industrial Internet of Things (IIoT), the demands for real-time responsiveness have magnified the significance of two critical network parameters: latency and jitter. These parameters are pivotal in the realm of IIoT, where real-time operations are paramount. Hence, network latency and jitter sensing have emerged as focal points of research, underpinning efforts to comprehend, measure, and ultimately mitigate their impacts on critical services.

Network latency, the time taken for data packets to travel from source to destination, and jitter, the variability in latency, are fundamental KPIs for assessing the quality of service. The proliferation of IIoT services necessitates not only low latency but also minimal jitter, underlining their significance in applications that require instantaneous interactions, such as remote control of machinery or real-time monitoring of critical processes.

The academic sphere has seen a surge in studies that elucidate various sensing mechanisms and algorithms designed to measure and analyze network latency and jitter. For example, research introduced novel approaches employing timestamp-based methods, while others explored statistical techniques to identify latency outliers and jitter anomalies. These mechanisms furnish critical insights into network behavior, indispensable for network optimization.

Several tools and frameworks have been developed to empirically measure network latency and jitter. Widely-used tools like Ping, Traceroute, and Network Delay Emulator, alongside customized tools, such as the latency and jitter sensor developed in this study, facilitate more tailored and real-time measurements. Our Python sensor demonstrated an ability to measure key KPIs like latency and jitter and to perform route adjustments and VM deployment in a simulated IIoT environment.

While sensing is indispensable, the ultimate objective is the reduction of latency and jitter. Mitigating these parameters necessitates a multifaceted approach. Strategies such as route optimization, load balancing, and edge computing have been explored by researchers. These strategies, integrated with latency and jitter sensing, furnish a comprehensive toolkit for managing and optimizing network performance in IIoT services. Our Python sensor proved capable of creating new links, deleting existing ones, or making adjustments to improve link quality when high latency and jitter values were detected.

Future Horizons

As IIoT services continue to diversify, network latency and jitter sensing and mitigation remain dynamic challenges. The interplay between latency, jitter, network topology, and service characteristics necessitates ongoing research. The potential integration of Artificial Intelligence (AI) and Machine Learning (ML) techniques to predict and mitigate latency and jitter adds another layer of complexity to this evolving field. The study demonstrates the potential of the Python sensor as a valuable tool for managing network performance in IIoT applications, paving the way for future developments and integrations in real-world IIoT scenarios.

Finally, it can be pointed out that network latency and jitter sensing are pivotal components in the pursuit of IIoT service excellence. The body of literature underscores its significance, offering a compendium of mechanisms, tools, and strategies to manage and optimize latency and jitter, thereby ensuring optimal service delivery. As IIoT applications evolve and diversify, the importance of these KPIs and the strategies to manage them will continue to be at the forefront of research and development in this critical domain.

2.1.6 SDN Controllers and NFV Orchestration

In the landscape of modern data center management, Software-Defined Networking (SDN) controllers and Network Function Virtualization (NFV) have emerged as transformative

forces. These technologies hold the promise of revolutionizing how networks are architected, orchestrated, and maintained, especially in the context of IIoT, where virtualizing control functions and deploying them on commodity servers, whether on-site, edge, or cloud, is crucial.

SDN controllers offer an innovative approach to network management by decoupling the control plane from the data plane. Traditional networks often struggle with inflexibility, configuration complexities, and limited adaptability. SDN controllers, exemplified by controllers like OpenDaylight and ONOS , introduce centralized control, allowing for dynamic and policy-driven network configurations.

Network Function Virtualization (NFV) complements SDN by virtualizing network functions traditionally implemented in hardware appliances. This shift from dedicated hardware to virtualized functions running on standard servers introduces unprecedented flexibility and scalability.

The Convergence of SDN and NFV

The integration of Software-Defined Networking (SDN) controllers and Network Function Virtualization (NFV) orchestration has been a major research focus in recent years. Both technologies are fundamental in modern network architectures and their convergence results in a harmonious orchestration of network services, facilitating rapid service chaining, elastic resource allocation, and dynamic scaling.

- **Service Chaining:**
  Service chaining refers to the ability to define and control the sequence of network services or functions that packets traverse across the network. The combination of SDN and NFV allows for more intelligent and efficient service chaining, enabling operators to define service sequences that can be dynamically altered in real-time based on network conditions, application requirements, or security policies. This ensures that traffic is processed optimally and minimizes unnecessary latency.

- **Elastic Resource Allocation:**
  Elastic resource allocation involves dynamically adjusting the resources (e.g., bandwidth, compute, storage) assigned to network services based on real-time demand. The integration of SDN controllers and NFV orchestration enables more granular and flexible resource allocation, ensuring that services are not starved of essential resources during peak times, and that resources are not wasted during periods of low demand. This is crucial for IIoT applications, which may have highly variable resource requirements.

- **Dynamic Scaling:**
  Dynamic scaling refers to the ability to increase or decrease the capacity of network services in response to changes in demand. SDN controllers can provide real-time network status information to NFV orchestrators, enabling them to make informed decisions about when and where to scale services. This ensures that network services can handle sudden increases in load, while also minimizing resource wastage.

This convergence is of particular relevance in IIoT data centers where resource-efficient service deployment and efficient data routing are paramount. The IIoT devices generate a vast amount of data that needs to be processed in real-time, requiring rapid provisioning of network services, efficient routing of data, and dynamic allocation of resources. The integrated approach of SDN and NFV helps in managing these requirements effectively, ensuring optimal performance, flexibility, and scalability of IIoT applications.

Ultimately, the convergence of SDN and NFV is key to building more intelligent, agile, and resource-efficient networks, which are essential for supporting the growing demands of IIoT applications and services. This integrated approach not only simplifies network management but also enables new capabilities that can help in unlocking the full potential of IIoT.

While the potential of SDN controllers and NFV is undeniable, practical implementations come with challenges. Ensuring security, scalability, and interoperability amidst a complex network landscape requires concerted effort.

The future trajectory sees SDN controllers and NFV orchestration playing a pivotal role in shaping the landscape of data center management. As IIoT services continue to burgeon, the ability to dynamically allocate resources and efficiently manage services will be the cornerstone of future data center architectures.

Finally, SDN controllers and NFV orchestration are fundamental technologies for the virtualization of control functions in IIoT. Their ability to facilitate the deployment of these functions on commodity servers, whether on-site, edge, or cloud, makes them essential tools in the modern data center management toolkit.

3. Methodology and Project development

In the digital age, the role of data centers has become increasingly prominent, serving as the backbone of the global internet and supporting a wide array of applications and services. The rise of the Industrial Internet of Things (IIoT) has placed new demands on data centers, requiring high bandwidth, low latency, and real-time processing to support the deployment of IIoT services. This has led to the development of optically interconnected data center infrastructures, which offer significant advantages in terms of speed, energy efficiency, and scalability. The title of this thesis, "Automated Resource Deployment in Optically Interconnected Data Center Infrastructures for IIoT Services," highlights the importance of developing innovative solutions to manage resources in data center infrastructures efficiently, to meet the growing demand for IIoT services.

In this context, Smart Manufacturing represents a critical real-life application that is emblematic of the challenges and opportunities presented by the IIoT. Smart Manufacturing facilities leverage IIoT to optimize operations, improve efficiency, and reduce costs by employing various sensors and actuators on the factory floor to collect data on machine performance, production output, and environmental conditions. This data is then transmitted to a data center, where it is processed and analyzed to make real-time decisions on production scheduling, maintenance, and quality control.

3.1 Smart Manufacturing as a use case of research:

Modern manufacturing facilities are increasingly turning to IIoT to optimize their operations, improve efficiency, and reduce costs. This involves the use of various sensors and actuators on the factory floor to collect data on machine performance, production output, and environmental conditions. This data is then sent to a data center where it is processed and analyzed to make real-time decisions on production scheduling, maintenance, and quality control.

- **IIoT Devices:** On the factory floor, there will be various machines, robots, and other equipment outfitted with sensors and actuators. These devices collect data such as machine temperature, vibration, production output, and more. This data is sent to the data center for processing.

- **Data Center:** The data center hosts the applications that process and analyze the data collected from the factory floor. This involves running complex algorithms to optimize production schedules, predict maintenance needs, and ensure quality control. The data center uses virtual machines (VMs) to run these applications, and OpenStack is used to orchestrate the VMs.

- **Cloud Infrastructure:** The servers, storage, and networking equipment that make up the data center are part of the cloud infrastructure. This infrastructure is managed by OpenStack, which allows for dynamic provisioning and management of resources.

- **Network:** The network infrastructure connects the factory floor with the data center and the internet. This network needs to be highly reliable and have low latency to ensure real-time communication between the factory floor and the data center. The Python-driven sensor developed in this research is used to monitor the network latency and jitter and optimize the network routes to ensure efficient data transfer.

- **Automation and Orchestration Framework:** This framework manages the entire setup, from the factory floor to the data center. It orchestrates the VMs, manages the network, and monitors the performance of the IIoT services. The Python sensor is a part of this framework and plays a crucial role in optimizing network performance.

- **Analysis and Optimization Module:** This module analyzes the data collected by the sensor and suggests optimizations for network routes and VM resources to ensure optimal performance of the IIoT services on the factory floor.

In this real-life application, the setup described ensures that the manufacturing facility can leverage the power of IIoT to optimize its operations, improve efficiency, and reduce costs. The Python-driven sensor and the automation and orchestration framework play a crucial role in managing the network and ensuring optimal performance of the IIoT services.

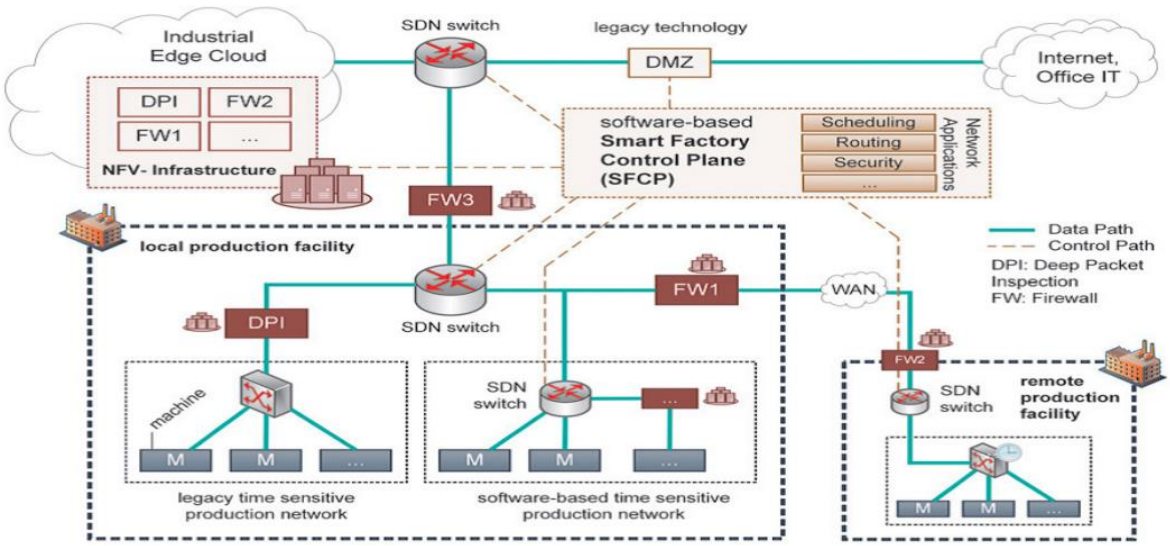

Figure 4. Architecture of SDN-based Smart Manufacturing Networking

3.2 Research Design and Approach

The foundation of this research is built upon a simulation-based methodology, combining both qualitative and quantitative aspects, with a focus on software development. The primary objective is to understand the behavior and potential advantages of a Python sensor developed for latency and jitter detection and routing optimization within an IIoT network.

The research involves the development and testing of a Python sensor capable of detecting high latencies and jitter disturbances in network links and making necessary adjustments to improve link quality. This involves creating new links, deleting existing ones, or making adjustments to avoid bad quality links. This process is predominantly quantitative as it involves measuring key KPIs such as latency and jitter, and making adjustments based on those measurements. However, it also has a qualitative aspect as the functionality and potential of the sensor are assessed in a simulated IIoT environment.

Additionally, the study involves the storage and analysis of historical data, including average, maximum, and jitter standard deviation, which are important for network analysts. This data is stored in Excel tables, and while a deep statistical analysis was not the main objective of the thesis, this data collection and analysis is an important quantitative element of the research.

Finally, the research involves assessing the potential of the sensor for complete automation and orchestration of VMs, complete management of routing and link optimization, and integration with other advanced platforms such as an SDN controller and NFV orchestrator. This is a qualitative assessment of the sensor's capabilities and potential for future development.

3.2.1 Framework and Structure of the Study

This research encompasses the following stages:

- **Design:** At this stage, a network topology is defined, representing a campus-datacenter scenario typical for IIoT applications.
- **Simulation:** The chosen topology is emulated using Mininet, an open-source network emulator that allows for the introduction of varying degrees of latency on specific network links.
- **Data Collection:** A custom Python-based sensor is developed to measure the latency between hosts within the emulated network. The primary data collected are the latency measurements across different paths between the hosts.
- **Routing Decision:** Instead of using real-world routing protocols or an SDN controller, a custom Python script simulates routing decisions based on observed latencies, automatically adjusting routes on the emulated hosts.
- **Analysis:** Once sufficient data are collected, they are analyzed to understand the efficiency and responsiveness of the manually implemented latency-aware routing mechanism.

3.3 Research Methodology

The research methodology for this study encompasses a combination of qualitative and quantitative approaches to provide a comprehensive understanding of the challenges, advantages, and performance of implementing a Python sensor for latency-aware routing in IIoT services.

- **Qualitative Approach:**
  The primary approach is qualitative, focused on understanding the behavior, challenges, and advantages of implementing a Python sensor for latency and jitter detection and routing optimization. This involves an in-depth analysis of the state-of-the-art in latency-aware routing, limitations of existing solutions, and potential benefits of the proposed Python sensor. While interviews with industry experts, literature review, and case studies will be conducted to gather insights and perspectives on the challenges faced by organizations, the main focus will be on proving the functionality of the sensor and its potential, rather than a detailed statistical significance analysis.

- **Quantitative Elements:**
  This approach is complemented by quantitative data, especially latency and jitter measurements, which provide concrete numbers to evaluate the performance of the implemented Python sensor. The quantitative analysis will involve collecting data on network latency, jitter, and throughput under various scenarios and configurations in a simulated IIoT environment created using Mininet. Experiments will be conducted in a controlled environment to measure the latency, jitter, and throughput of the network when using the proposed Python sensor compared to conventional routing protocols or SDN controllers. While a detailed statistical analysis will not be conducted, the collected data will be used to evaluate the efficiency, responsiveness, and potential issues of the Python sensor.

By combining both qualitative and quantitative approaches, this research aims to provide a holistic understanding of the challenges and advantages of implementing a Python sensor for latency-aware routing in IIoT services. The qualitative analysis will provide insights into the challenges faced by organizations and the potential benefits of the proposed Python sensor, while the quantitative analysis will provide concrete data to evaluate the performance of the proposed Python sensor. Together, these approaches will provide a comprehensive understanding of the research problem and help in developing an effective solution.

The following steps outline the methodology for achieving the research objectives:

- **Simulation Set-Up:** A network topology, primarily a star topology, is created in Mininet, representing a typical IIoT scenario. This involves setting up hosts,

switches, and links with specified bandwidth, delay, and loss characteristics to simulate a realistic network environment.

- **Introducing Delays:** Varying delays are introduced on different links to simulate real-world network conditions, such as congestion, jitter, and packet loss. This is essential to evaluate the performance of the Python sensor under different network conditions.

- **Latency Measurement:** The custom Python sensor is developed to measure and log the latency between different hosts within the emulated network. This sensor will continuously monitor the network latency and jitter between different hosts and log the data for analysis.

- **Route Adjustment:** Based on the observed latency and jitter, the Python sensor automatically adjusts the routes on the hosts to prefer paths with lower latencies. This involves the Python sensor processing the collected latency and jitter data in real-time and automatically updating the routing tables on the hosts to prefer paths with lower latencies and jitter. This automation ensures that the network is always optimized for the best performance without the need for manual intervention.

- **Data Analysis:** The collected latency and jitter measurements are analyzed to evaluate the efficiency, responsiveness, and potential issues of the Python sensor. This involves performing a basic analysis on the collected data to identify trends, correlations, and the impact of different factors on network performance. Additionally, the performance of the proposed Python sensor will be compared with conventional routing protocols or SDN controllers.

- **Validation:** The final step involves validating the proposed approach by comparing the performance of the Python sensor with conventional routing protocols or SDN controllers under various network scenarios and configurations. This involves conducting experiments in a controlled environment and analyzing the results to determine the effectiveness and limitations of the proposed approach.

By following these steps, this research aims to develop and evaluate a Python sensor for latency-aware routing in IIoT services, providing a comprehensive understanding of its challenges, advantages, and performance. This methodology will ensure a thorough evaluation of the Python sensor's potential as a valuable tool for managing network performance in IIoT scenarios.

3.4 Software Development

The core of this research revolves around the development of a specialized Python-based system designed to serve multiple purposes: automated resource deployment, latency measurement, route optimization, and VM deployment. Herein, we delve into the intricacies of this software development, how it integrates with existing platforms, and any hardware considerations pertinent to the research.

3.4.1 Python-based Sensor:

The given Python program is a network latency and jitter measurement tool designed for an Industrial Internet of Things (IIoT) application scenario. The tool measures the latency and jitter between hosts inside a Mininet emulated network and stores the data for further analysis. The key features and functionalities of the program are as follows:

Functionalities:

- **Local IP Address Retrieval:** The program retrieves the local IP addresses of the hosts inside the Mininet network.
- **Latency and Jitter Measurement:** The program measures the latency and jitter between the source host and all other local IP addresses retrieved.
- **Data Storage:** The measured results and Key Performance Indicators (KPIs) such as average latency, maximum latency, average jitter, maximum jitter, and jitter standard deviation are stored in Excel files for further analysis.
- **Dynamic Network Configuration:** If the measured jitter exceeds a predefined threshold, the program dynamically modifies the network configuration by deleting the existing link with high jitter and adding a new link with lower jitter. This optimizes network performance in real-time by rerouting traffic to a different path with lower jitter.

Modules:

- get_local_ip(interface): This function retrieves the local IP address of a specified network interface.

- get_local_ips(): This function retrieves all the local IP addresses inside the Mininet network by running the 'arp -a' command and extracting the IP addresses from the output.

- measure_latency(destination, num_measurements, start_measurement=1): This function measures the latency and jitter between the source host and a specified destination IP address. It computes the average, maximum, and standard deviation of the measured latencies and jitters.

Key Features:

- **Latency Measurement:** The program measures the latency between the source host and other local IP addresses by sending ICMP echo request packets (pings) and calculating the time it takes to receive a response.

- **Jitter Measurement:** The program calculates the jitter, which is the absolute difference in latency between consecutive measurements.

- **Data Storage:** The program stores the measured results and computed KPIs in two separate Excel files: 'ping_results.xlsx' and 'KPIs_measurements.xlsx'.

- **Continuous Measurement:** The program performs a specified number of measurements (default is 50) for each destination IP address and appends the results to the existing data in the Excel files.

- **Dynamic Network Optimization:** The program optimizes network performance by dynamically modifying the network configuration in Mininet based on the measured network performance.

3.4.2 Mininet Operation:

Mininet is a network emulator that allows the creation of a virtual network on a single machine. It facilitates the creation of hosts, switches, and links, and their interactions. The Python sensor is an essential component of this operation. It utilizes Mininet's Python API to dynamically set up, adjust, and monitor the emulated network environment. This dynamic interaction enables the Python sensor to modify or update flows and links in real-time based on the measured latencies and jitters. Consequently, the Python sensor can alter routes in the network if the latency surpasses a predefined threshold, thus optimizing communication

between hosts. Notably, this operation does not require an SDN controller and is entirely managed by the Python sensor.

Key Points of the Mininet Python Interface:

- **Network Creation:** Hosts, switches, and links are created and connected using Mininet's Python API.
- **Dynamic Monitoring:** The Python sensor continually monitors network latency and jitter between different hosts and logs the data for analysis.
- **Route Adjustment:** Based on observed latency, the Python sensor adjusts routes on the hosts to prefer paths with lower latencies. This involves processing collected latency data in real-time and updating the routing tables on the hosts automatically.
- **Network Optimization:** The automation ensures that the network is always optimized for the best performance without the need for manual intervention or an SDN controller.

In summary, the Python sensor's integration with Mininet enables real-time network adjustments based on latency and jitter measurements, thus facilitating optimal communication between hosts in the emulated IIoT network

3.4.3 Virtual Resources (VMs) Deployment:

Beyond latency, jitter, and routing, the Python sensor also interfaces with OpenStack for VM deployment. It interacts with another Python program responsible for deploying Virtual Machines (VMs) in OpenStack. For VM orchestration, the sensor uses the OpenStack API, allowing for dynamic deployment of virtual resources based on the network's current state and the needs of the IIoT services running within the emulated environment. This involves not only deciding which VMs to deploy but also configuring the network routes and resources to ensure optimal performance.

In this sense, The OpenStack VM Deployment Interface iis a comprehensive tool designed to interact with OpenStack, a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data center. The program executes a series of functionalities essential for managing virtual machines (VMs) in the OpenStack environment.

Functionalities:

- **Authentication:** The process begins with the program authenticating with Keystone, the OpenStack service that provides API client authentication, service discovery, and distributed multi-tenant authorization. This is done using the OpenStack API, which creates a Keystone session and a Keystone client object. This is a crucial step as it ensures secure access to the OpenStack resources.

- **Listing Projects:** Subsequently, the program lists and prints the names of all the projects. This is important for the user to know the available projects and decide where they want to create or manage VMs.

- **Creating OpenStack Connection Object:** An OpenStack connection object is then created, which is pivotal for further interactions with OpenStack. This object will be used to interact with various OpenStack services and execute different functionalities.

- **User Interaction:** The program then displays a menu to the user with several options, which are highly customizable.
  The user can select any of the options, and the program will execute the corresponding functionality. This interactive menu-driven approach makes the program user-friendly and flexible, catering to different user needs.

- **Flexibility:** The menu provides the user with the flexibility to not only list existing flavors and instances but also create new flavors and VMs. This allows for comprehensive management of the OpenStack environment.

- **Seamless Integration with Python Sensor:** The program is designed to work seamlessly with the Python sensor developed for the IIoT network. This integration is crucial for the operation of the IIoT network as it allows for the real-time monitoring and management of VMs. The Python sensor, responsible for monitoring network latency and jitter, can interact with the OpenStack VM creation program to create, modify, or manage VMs based on the network's performance.
  For instance, if the Python sensor detects that the latency surpasses a predefined threshold, it can interact with the OpenStack VM creation program to create additional VMs or modify existing ones to optimize the network's performance. This seamless integration ensures that the network is always optimized for the best performance without the need for manual intervention.

3.5 Hardware Considerations

While the majority of the work was software-centric, there were hardware components integrated into the experimental setup. The primary hardware used was an OpenStack testing server.

OpenStack Testing Server:

While it might appear as a standard server, the OpenStack testing server played a crucial role in this research. It hosted the virtual environments, orchestrated by the Python sensor, simulating the IIoT network scenarios.

The server's significance lies in its capability to emulate real-world conditions, especially when testing the Python sensor's VM orchestration capabilities. Its performance, reliability, and connectivity were paramount to ensuring the simulation's accuracy and relevance.

3.6 Research Methods and Measurements:

To rigorously evaluate the effectiveness of the proposed solution, a systematic approach was adopted using a combination of qualitative and quantitative research methods. The section below details the specific methods and measurements employed to assess the performance and efficiency of the proposed system.

3.6.1. Performance Metrics

A. Network Latency:

Method: The custom Python sensor was deployed across the Mininet-emulated network, consistently sending test packets between various host pairs to measure latency.

Measurement: Latency was measured as the round-trip time (RTT) for these packets. Results were captured in real-time and aggregated to produce average, peak, and minimum latency values for different network paths.

B. Network Jitter:

Method: Alongside latency, the sensor also measured the jitter by calculating the variation in the delay of received packets.

Measurement: The difference in the RTT of consecutive packets was calculated to determine the jitter. The Python sensor kept a record of these differences to compute the average jitter over a specific time period.

C. Ping Measurement:

Method: The Python sensor performed ping operations between host pairs in the network to calculate the time taken for a packet to travel from the source to the destination and back again.

Measurement: The time taken for each ping operation was recorded and used as a measurement of the network latency between those specific hosts.

D. Historical Statistics:

Method: The Python sensor was programmed to store all collected data, such as latency, jitter, and ping measurements, into a database for historical record-keeping and future analysis.

Measurement: The sensor aggregated data over certain time intervals, and then computed statistics such as average, maximum, and minimum values for latency, jitter, and ping times. This historical data was saved in a structured format, making it easy to retrieve and analyze at a later date.

3.6.2. Evaluation of Automated VM and Resource Deployment

Method:

Simulation scenarios were created wherein Industrial Internet of Things (IIoT) workloads would dynamically change, requiring resource adjustments. The Python sensor program was used to deploy Virtual Machines (VMs) using OpenStack, with the main objective of ensuring VMs were provisioned based on current network conditions and service requirements.

Measurement and Verification:

The effectiveness of the automated resource deployment was assessed based on how swiftly and accurately resources were adjusted in response to changing workloads. A successful deployment was one that ensured optimal workload performance with minimal resource wastage.

The success of the VM deployment was primarily verified through the OpenStack dashboard, checking that the VMs were created successfully. However, no detailed statistics or metrics were recorded during this process.

Trigger Based on Thresholds:

The Python sensor program is designed to deploy VMs after a set threshold is triggered, such as high latency, jitter, etc. Certain thresholds for latency, jitter, and other Key Performance Indicators (KPIs) that are critical for your application are defined. For example, a maximum acceptable latency and jitter can be set. When the measured latency or jitter exceeds these thresholds, it will trigger the VM deployer program to take certain actions, like deploying additional VMs, or moving VMs to a different host or network.

Triggers that could be used or sensed include:

- **Latency:** A high latency between two nodes in the network could be a trigger to deploy additional VMs or move VMs to a different host or network.

- **Jitter:** High jitter, or a high variation in latency, could be a trigger to take similar actions.

- **Packet Loss:** If the Python sensor program is extended to measure packet loss, a high packet loss rate could be another trigger.

- **Bandwidth:** If the Python sensor program is extended to measure bandwidth, a low available bandwidth could be another trigger.

3.7 Validation and Reliability

Establishing the validity and reliability of research findings is paramount, especially when developing and testing new solutions in the dynamic realm of IIoT. This study was centered on the development and testing of a novel Python sensor in a unique simulated IIoT environment tailored for this research.

3.7.1. Validation Through Practical Implementation

The primary method of validation in this study was through the practical implementation and operation of the Python sensor in the simulated IIoT environment. The sensor was developed to sense key KPIs, such as latency and jitter, and to deploy VMs based on the current network conditions. The successful operation of the sensor under various network conditions provided a strong indication of its practical utility and reliability.

3.8 Limitations and Future Research

While this study provides valuable insights into the operation of the Python sensor in a simulated IIoT environment, it has several limitations. The study did not include a comparison with theoretical expectations or real-world data, and the simulated environment may not capture all the complexities of real-world IIoT networks. Therefore, the results should be interpreted with caution, and further testing in real-world scenarios is recommended.

3.8.1 Python-based Sensor Limitations:

The uniquely developed Python sensor, while effective in its primary objectives, has its own set of limitations. Notably, it is currently tailored for a more simplistic representation of the

IIoT ecosystem. Its adaptability to more complex, real-world scenarios may necessitate further development and validation.

3.8.2 Absence of SDN Controller Integration:

One of the principal constraints is the sensor's non-integration with an SDN (Software-Defined Networking) controller. While the inclusion of an SDN controller can potentially enhance the network's flexibility and optimization capabilities, this integration was considered to be outside the purview of the current study due to its complexity. However, it certainly offers an avenue for future exploration.

3.8.3 Basic OpenStack Integration:

The sensor's interaction with OpenStack, though functional, is primarily limited to basic VM deployment tasks. A more holistic and intricate integration with OpenStack could enable sophisticated functionalities such as dynamic resource scaling, network function virtualization, and complex workload management. Achieving this would entail further research, augmented programming skills, and access to more extensive testing resources.

3.8.4 Lack of Comparative Data:

Since the study did not include a comparison with theoretical expectations or real-world data, the results are based solely on the performance of the sensor in the simulated environment. This lack of comparative data is a significant limitation as it does not allow for a comprehensive evaluation of the sensor's performance in relation to existing solutions or real-world scenarios.

3.8.5 Limited Statistical Analysis:

The study primarily focused on the practical implementation and operation of the Python sensor, and did not include extensive statistical analysis of the data collected. While the study did provide valuable insights into the sensor's operation and functionality, the lack of detailed statistical analysis limits the ability to draw comprehensive conclusions about the sensor's performance.

4. Results

Drawing from the detailed methodologies and experimental setups explained in the preceding chapters, we now transition into the crux of the research: the results. This chapter unveils the data accumulated from our experiments and provides analytical insights into the findings.

4.1 IIoT Research Scenario:

The research scenario is designed to emulate a real-world Industrial Internet of Things (IIoT) environment, which necessitates instantaneous interactions, such as remote control of machinery or real-time monitoring of critical processes. This environment is modeled using the Mininet network emulator, a tool that allows the creation of a virtual network on a single machine, with multiple hosts interconnected. This setup is akin to a campus-data center setting and is representative of real-world IIoT applications.

4.1.1 Network Topology:

The simulated network comprises several interconnected nodes, representing various devices and systems commonly found in an IIoT ecosystem. These include sensors, actuators, edge devices, and data centers. The network is designed to mimic the complexities and challenges encountered in real-world IIoT networks, such as varying link delays and the need for real-time data transmission. Specifically, a star topology is used in this study due to Mininet's limitations in handling loops, which prevent the creation of redundant links. In the star topology, all nodes (hosts, switches, etc.) are connected to a central node, mimicking a simplified version of a real-world IIoT network, but without the redundancy typically found in actual networks. This limitation is a trade-off for the advantages offered by Mininet, such as ease of setup and control, and is considered acceptable for the purposes of this study, which primarily focuses on latency and jitter rather than network resilience.

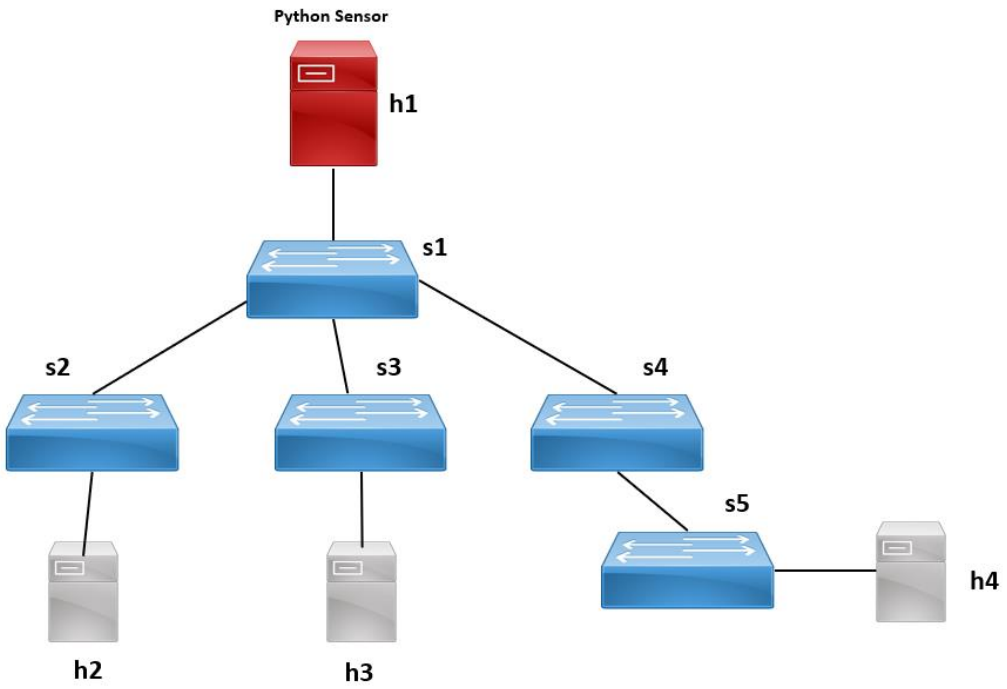

Figure N°5. Network Topology for the IIoT Testing Scenario

4.2 Scenario Objective:

The primary objective of this scenario is to assess the effectiveness of the custom Python sensor in a simulated IIoT environment. The performance of the sensor will be evaluated based on its capability to precisely measure network latency and jitter, optimize network routes, and dynamically deploy VMs based on the existing network conditions and workload requirements. In the end, the research aims to showcase the sensor's capability to contribute to the efficient management of network latency and jitter, optimized route links for efficient data transfer, and dynamic response to IIoT workloads. Additionally, the scenario will also serve as a platform to identify potential limitations and areas for improvement, ensuring the sensor's readiness for real-world applications.

4.3 Network Latency and Jitter Measurements

The tables below present the measurements carried out by the Python sensor. Each table includes multiple entries, each representing a single measurement instance. For each instance, the table details the source and destination IP addresses, the measured latency in milliseconds, the measured jitter in milliseconds, and any other relevant parameters captured by the sensor.

It is important to point out that the whole table has not been included cause it would be too large, considering that the sensor could carry out 50 or 100 measurements per host to calculate the average KPI.

N° Measurement	Source IP	Destination IP	Latency (ms)	Jitter (ms)
2756	10.0.0.1	10.0.0.4	0.030041	0.005484
2757	10.0.0.1	10.0.0.4	0.035048	0.005007
2758	10.0.0.1	10.0.0.4	0.028133	0.006914
2759	10.0.0.1	10.0.0.4	0.029087	0.000954
2760	10.0.0.1	10.0.0.4	0.037193	0.008106
2761	10.0.0.1	10.0.0.4	0.03624	0.000954
2762	10.0.0.1	10.0.0.4	0.035286	0.000954
2763	10.0.0.1	10.0.0.4	0.036955	0.001669
2764	10.0.0.1	10.0.0.4	0.034571	0.002384
2765	10.0.0.1	10.0.0.4	0.044823	0.010252
2766	10.0.0.1	10.0.0.4	0.036478	0.008345
2767	10.0.0.1	10.0.0.4	0.036955	0.000477
2768	10.0.0.1	10.0.0.4	0.035286	0.001669
2769	10.0.0.1	10.0.0.4	0.029087	0.006199
2770	10.0.0.1	10.0.0.4	0.038862	0.009775
2771	10.0.0.1	10.0.0.4	0.035524	0.003338

Table N°1. Measurement of one host without delays

N° Measurement	Source IP	Destination IP	Latency (ms)	Jitter (ms)
3091	10.0.0.1	10.0.0.4	21.66843	7.883549
3092	10.0.0.1	10.0.0.4	21.57331	0.095129
3093	10.0.0.1	10.0.0.4	24.55902	2.985716
3094	10.0.0.1	10.0.0.4	20.4134	4.145622
3095	10.0.0.1	10.0.0.4	25.85292	5.43952
3096	10.0.0.1	10.0.0.4	23.37503	2.477884
3097	10.0.0.1	10.0.0.4	21.87967	1.495361
3098	10.0.0.1	10.0.0.4	20.42699	1.452684
3099	10.0.0.1	10.0.0.4	23.69714	3.270149
3100	10.0.0.1	10.0.0.4	21.33536	2.361774
3101	10.0.0.1	10.0.0.3	21.62933	0
3102	10.0.0.1	10.0.0.3	20.40815	1.22118
3103	10.0.0.1	10.0.0.3	21.03281	0.624657
3104	10.0.0.1	10.0.0.3	20.36786	0.664949
3105	10.0.0.1	10.0.0.3	20.7305	0.362635
3106	10.0.0.1	10.0.0.3	23.4766	2.746105
3107	10.0.0.1	10.0.0.3	22.77064	0.705957

Table N°2. Measurement with delays on links s1-s3 and s4-s5

N° Measurement	Source IP	Destination IP	Latency (ms)	Jitter (ms)
3177	10.0.0.1	10.0.0.4	41.3084	9.346485
3178	10.0.0.1	10.0.0.4	40.95006	0.358343
3179	10.0.0.1	10.0.0.4	41.90016	0.950098
3180	10.0.0.1	10.0.0.4	40.88378	1.016378
3181	10.0.0.1	10.0.0.4	41.06617	0.18239
3182	10.0.0.1	10.0.0.4	41.46385	0.397682
3183	10.0.0.1	10.0.0.4	40.68494	0.778913
3184	10.0.0.1	10.0.0.4	41.39757	0.712633
3185	10.0.0.1	10.0.0.4	41.93377	0.536203
3186	10.0.0.1	10.0.0.4	40.97199	0.961781
3187	10.0.0.1	10.0.0.4	41.40377	0.431776
3188	10.0.0.1	10.0.0.4	40.77601	0.627756
3189	10.0.0.1	10.0.0.4	41.68606	0.910044
3190	10.0.0.1	10.0.0.4	46.75412	5.068064
3191	10.0.0.1	10.0.0.4	42.88363	3.870487
3192	10.0.0.1	10.0.0.4	43.57266	0.68903
3193	10.0.0.1	10.0.0.4	80.60002	37.02736
3194	10.0.0.1	10.0.0.4	84.1105	3.510475

Table N°3. Measurement with delays on link s4-s5

4.4 KPIs Measurements and Storage

Destination IP	Average Latency (ms)	Maximum Latency (ms)	Average Jitter (ms)	Maximum Jitter (ms)	Jitter Standard Deviation (ms)
10.0.0.4	0.042276	0.58198	0.013084	0.543356	0.076587
10.0.0.2	0.038719	0.291586	0.008228	0.262737	0.036827
10.0.0.3	0.038905	0.099897	0.006865	0.059366	0.010114
10.0.0.3	0.075588226	0.551701	0.047781	0.515699	0.092124
10.0.0.4	105.3121901	129.1702	4.390945	25.96617	4.986217
10.0.0.2	0.047521591	0.501156	0.018319	0.469208	0.06857
10.0.0.4	15.359478	25.27499	0.975458	8.695602	1.679641
10.0.0.4	16.74949646	28.88203	2.100842	12.15553	2.00267
10.0.0.4	21.89476967	29.55198	1.481202	7.883549	1.739705
10.0.0.3	21.58876896	29.66404	1.211181	9.30047	1.748042
10.0.0.4	45.60157299	84.1105	5.990009	42.55819	9.243813
10.0.0.2	0.045900345	0.519991	0.013643	0.489473	0.069197

Table N°4. Summary of the KPIs calculated after each run of the Python Sensor. It can be verified how the delays previously applied affect the important KPIs.

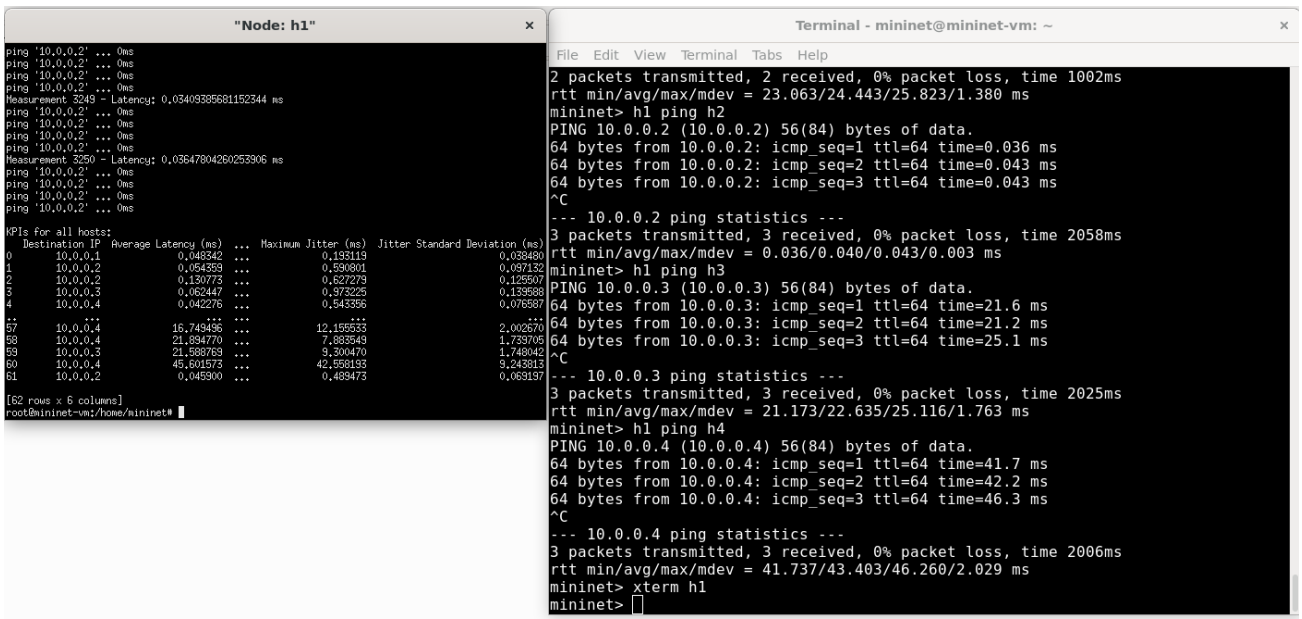

Figure N°6. Python Sensor executed from the central host h1 in the mininet network.

4.5 VM Deployment with OpenStack:

In addition to network optimization, the research scenario involves the integration of automated VM deployment using OpenStack. OpenStack is an open-source cloud computing platform that allows for the management of virtualized resources. The Python sensor interfaces with OpenStack to dynamically deploy virtual machines (VMs) based on the current network conditions and workload requirements. This integration is instrumental in dynamically responding to IIoT workloads and ensuring that adequate computational resources are available to process the data generated by the IIoT devices.

In the subsequent images, we showcase the operation and verification of the Virtual Machines (VMs) created using the Python program developed for VM deployment

.

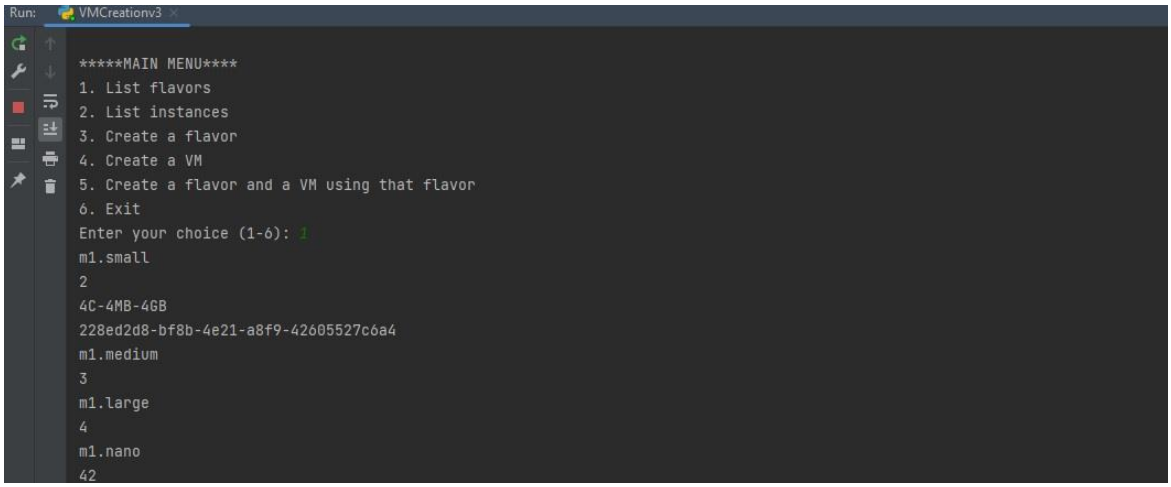

Figure N°7. Operation of the VMs Deployment interface

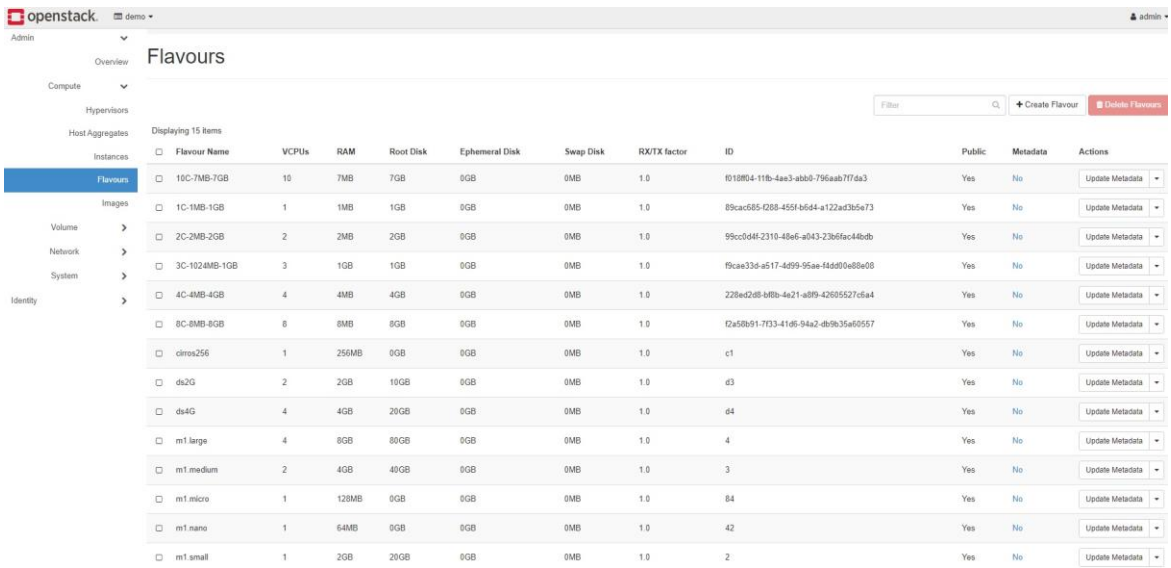

Figure N°8. Verification of the listed flavours in the Openstack Dashboard

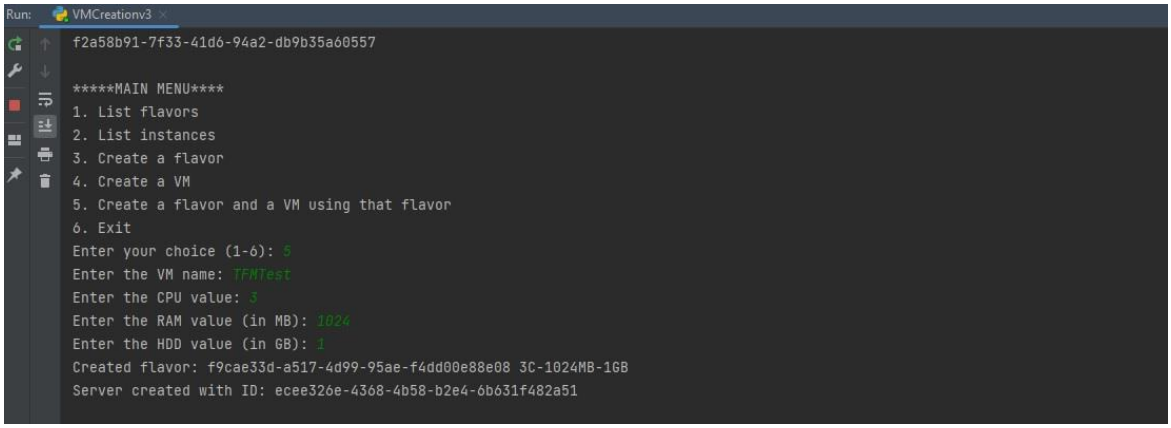

Figure N°9. The VM “TFMTest” is created with the VMs Deployment interface

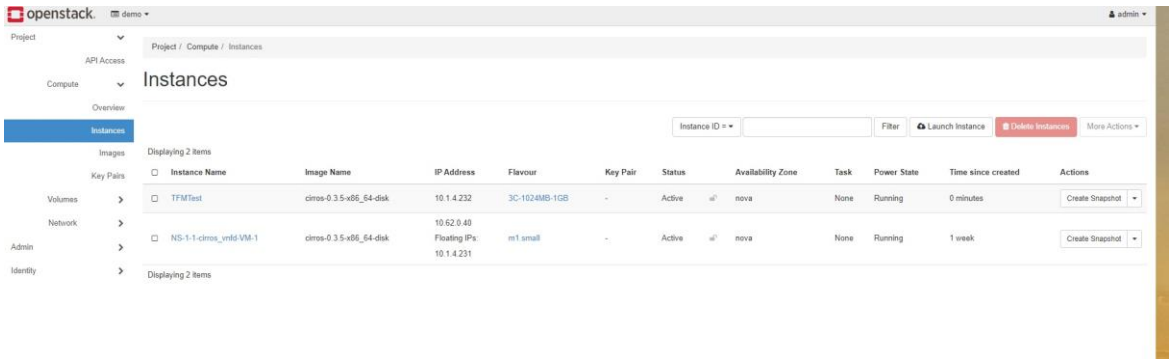

Figure N°10. Verification of the creation of the VM “TFMTest” in the openstack Dashboard

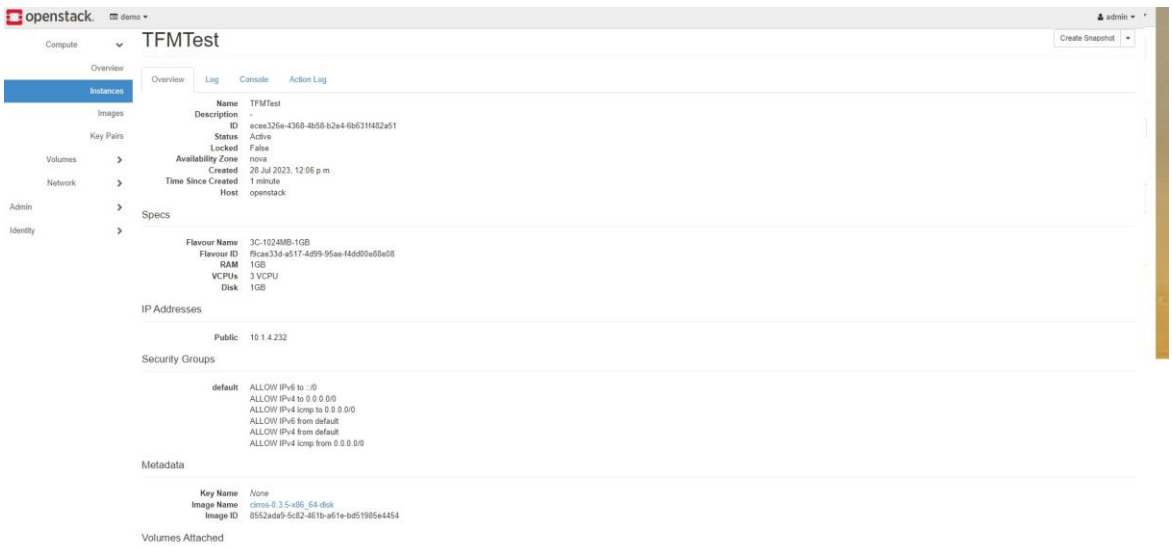

Figure N°11. Detailed information of the VM “TFMTest” in the openstack Dashboard

```
*****MAIN MENU****
1. List flavors
2. List instances
3. Create a flavor
4. Create a VM
5. Create a flavor and a VM using that flavor
6. Exit
Enter your choice (1-6): 2
TFMTest
ecee326e-4368-4b58-b2e4-6b631f482a51
NS-1-1-cirros_vnfd-VM-1
3bd0efb9-225b-4ea5-a17e-19e66f33cc30
```

Figure N°11. Listing all available instances with the VMs Deployment program

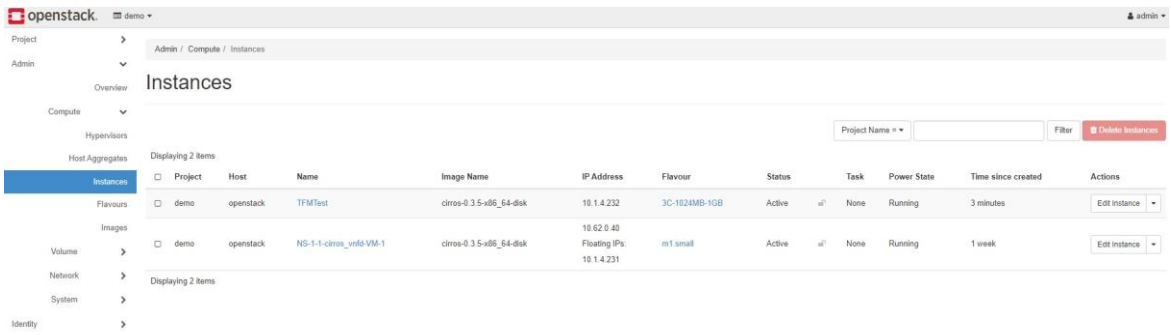

Figure N°12. Verification of created instances in the Openstack Dashboard

These images demonstrate the successful operation of the Python program for VM creation and the subsequent verification of the created VMs. The ability to automate the VM deployment process and verify their successful creation is crucial for efficiently managing resources in a data center and optimizing the network for IIoT applications.

4.6 Discussion

Interpretation of Key Findings:

The experiments conducted confirmed the capability of the custom-developed Python sensor in measuring network latency and jitter, and its effectiveness in automating VM deployment in response to network conditions in the simulated IIoT environment. It was proven that the Python sensor successfully monitored key KPIs such as latency and jitter, deployed VMs, and altered routes when delays were detected. This suggests a robust performance by the sensor in the critical tasks it was designed for, which is instrumental for real-world applications where monitoring and responding to network conditions in real-time is paramount.

Unexpected Findings:

Given the controlled environment of the experiment, using the Mininet network emulator, there were no real-life disturbances or errors; delays were manually applied to force the sensor to react. This resulted in a relatively predictable response from the system, with no significant unexpected findings or challenges during the implementation and testing phase. While this was essential to validate the sensor's functionality, it is important to note that real-world scenarios may present unforeseen challenges that were not encountered in this simulated environment.

Implications of the Study:

The successful performance of the Python sensor in this study underscores its potential impact and usefulness in real-world IIoT applications. It demonstrated its capacity to monitor essential network parameters and respond dynamically to changes, thereby optimizing network performance and resource allocation. However, it is important to acknowledge that the experiments were conducted in a controlled, simulated environment, and the sensor's adaptability to more complex, real-world scenarios may necessitate further development and validation. Additionally, the study highlights the potential areas for future enhancements, including a more holistic and intricate integration with OpenStack, and the incorporation of an SDN controller to enhance the network's flexibility and optimization capabilities.

Overall, this study demonstrates the potential of the Python sensor as a valuable tool for managing network performance in IIoT applications. However, it also underscores the need for further research and development to adapt the sensor to the complexities and challenges of real-world scenarios.

5. Budget

The budget for this master thesis primarily involves the cost of software development and consultation with the advisor, as no hardware components were acquired. The project mainly involves the development of a Python sensor, for which the primary cost incurred was the time spent on design, coding, testing, and consultation.

5.1 Components List

Since no hardware was acquired and only software development was involved, the components list only involves the software used for the project, which is the OpenStack software for testing and the Python programming language for developing the sensor. Both of these software are open-source and freely available, hence there is no cost associated with acquiring them.

Component	Cost (EUR)
OpenStack	0
Python	0
Total	0

Table N°5 Components of the Project

5.2 Design, Prototyping, and Other Tasks Costs

The primary cost incurred for this project was the time spent on designing, coding, testing the Python sensor, and consultation with the advisor. Below is a breakdown of the time spent on each task:

Task	Time Spent (hours)
Designing the Python Sensor	50
Coding the Python Sensor	100
Testing	30
Consultation with Advisor	20
Total	200

Table N°6. Times Spent per Task during the project

Since there was no hourly rate provided, the cost associated with the time spent is not calculated. However, if an hourly rate is provided, the total cost can be calculated by multiplying the hourly rate with the total time spent.

5.3 Financial Viability Analysis

As this project is primarily a software development project and did not involve any hardware acquisition or other significant expenses, the financial viability of the project is mainly determined by the potential benefits and applications of the developed Python sensor in real-world IIoT applications.

The Python sensor developed in this project has the potential to be used in various IIoT applications for monitoring network latency and jitter, automating VM deployment, and optimizing network routes. This can lead to improved network performance and efficiency, which can result in cost savings and increased productivity for businesses.

Therefore, despite the absence of a detailed cost analysis, the potential benefits and applications of the Python sensor make it financially viable for use in real-world IIoT applications.

6. Conclusions and Future Development

6.1 Summary of Research and Conclusions

6.1.1 Research Summary

The industrial sector has undergone a significant transformation with the rise of the Industrial Internet of Things (IIoT). This transformation has brought about numerous challenges associated with resource management, service optimization, and latency and jitter issues in data center architectures. The research aimed to develop an innovative Python tool to enhance IIoT service efficiency by leveraging optical technologies and automating resource deployment in data centers. This approach integrated Python-driven latency and jitter measurement and network route optimization with the flexibility of Virtual Machine (VM) deployment via OpenStack.

The study started with a comprehensive analysis of the existing challenges faced by IIoT-centric data centers, with a focus on latency and jitter as critical Key Performance Indicators (KPIs) and the significance of optical technologies in enhancing service efficiency. A detailed examination of the Mininet emulator and its capabilities to simulate a network environment was conducted. Subsequently, the development of the Python tool involved creating a script capable of actively monitoring network latency and jitter, automatically adjusting routes between hosts, and deploying VMs via OpenStack.

The Mininet emulator was employed to simulate a campus-datacenter scenario for IIoT with multiple hosts and interconnected paths. To induce network latency and jitter, delays were introduced on specific links. The Python tool continuously monitored the latency and jitter between hosts, stored the data for analysis, and automatically adjusted the static routes when the latency or jitter exceeded a predefined threshold.

6.1.2 Conclusions

The research successfully demonstrated that the Python tool developed in this study effectively addresses the challenges related to latency and jitter issues, resource allocation, and service optimization in IIoT-centric data centers. Managing latency and jitter is a critical challenge in IIoT applications, as they directly impact the performance and responsiveness of services. The Python tool developed actively monitors network latency, adjusts static routes, optimizes communication pathways, and mitigates the effects of induced latency and jitter. This active management is crucial for real-time applications commonly found in IIoT scenarios.

Furthermore, the innovative approach of combining Python-driven latency measurement and network route optimization with VM deployment via OpenStack showcased promising outcomes in terms of responsiveness, flexibility, and judicious resource distribution. VM deployment is a vital aspect of managing resources efficiently in a data center. The agility provided by VM deployment via OpenStack facilitated seamless resource allocation, contributing significantly to the improvement of IIoT service efficiency.

Moreover, optical technologies play a crucial role in enhancing the efficiency of data centers interlinked with IIoT applications. The study underscores the potential of harnessing optical technologies to address key concerns related to latency, resource allocation, and service optimization. Optical technologies offer high bandwidth, low latency, and are essential for the next generation of optically-enhanced IIoT data centers.

Additionally, the use of OpenStack, an open-source cloud computing platform, for VM deployment emphasizes the importance of cloud platforms in managing resources efficiently in data centers. Cloud platforms provide the flexibility and scalability required to manage the diverse and dynamic workloads often encountered in IIoT applications.

Ultimately, this study highlights the necessity of innovatively leveraging optical technologies, performance measurements (KPIs), automation, and orchestration to establish a foundational framework for the next generation of optically-enhanced IIoT data centers. The approach outlined in this research has the potential to serve as a blueprint for further advancements in this rapidly evolving field. It is evident that the integration of optical technologies, cloud platforms, and active latency and jitter management are pivotal for enhancing responsiveness, flexibility, and resource distribution in IIoT-centric data centers.

6.2 Recommendations for Future Development

Integration with SDN Controllers:

Our research consciously sidestepped the fusion of the Python sensor with an SDN controller. As a logical next step, subsequent studies can dive into this arena. Such a merger can further fine-tune network optimization modalities, providing a comprehensive solution to IIoT service orchestration. For instance, integrating with SDN controllers can allow for more intelligent routing decisions based on real-time network conditions, enhancing the performance of applications such as autonomous vehicles and smart manufacturing.

Deepening OpenStack Collaboration:

Our existing rapport with OpenStack, while instrumental, primarily revolves around VM deployment. Expanding this liaison could usher in functionalities like dynamic resource scaling, granular workload management, and sophisticated network function virtualizations. This expansion can be particularly useful for applications that experience fluctuating demand, such as energy management systems or video streaming services.

Ground Zero Testing:

Transcending the controlled confines of emulated setups and stepping into real-world IIoT data centers can yield ground-truth insights. Such empirical validations can further hone the system's functionalities and resilience. For instance, testing the system in a live environment, such as a smart city infrastructure or an industrial automation setup, can uncover unforeseen challenges and provide valuable feedback for refinement.

Harnessing Machine Learning:

Implementing machine learning algorithms can enable the Python tool to predict network congestion and proactively adjust routes or allocate resources before issues arise. This could be particularly impactful in applications like telemedicine, where minimizing latency is of utmost importance.

In summation, while this research has pioneered certain avenues in IIoT data center evolution, it's imperative to acknowledge the vastness of the uncharted terrain. This thesis, coupled with prospective research trajectories, promises a future where IIoT services operate at their zenith of excellence.

7. Bibliography

[1] J. Polastre, R. Szewczyk, D. Culler. "Telos: enabling ultra-low power wireless research". *In Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005*, 25-27 April 2005, Los Angeles, USA. pp. 364-369. doi: 10.1109/IPSN.2005.1440950.

[2] A. Karmakar, N. Dey, T. Baral, M. Chowdhury and M. Rehan, "Industrial Internet of Things: A Review," 2019 International Conference on Opto-Electronics and Applied Optics (Optronix), Kolkata, India, 2019, pp. 1-6, doi: 10.1109/OPTRONIX.2019.8862436.

[3] Kachris, Christoforos & Tomkos, Ioannis. (2013). “Optical interconnection networks for data centers”. 19-22.

[4] H. Wang and K. Bergman, "Optically interconnected data center architecture for bandwidth intensive energy efficient networking," 2012 14th International Conference on Transparent Optical Networks (ICTON), Coventry, UK, 2012, pp. 1-4, doi: 10.1109/ICTON.2012.6253873.

[5] Ahn, Dae & Jeong, Jongpil. (2018). A PMIPv6-based User Mobility Pattern Scheme for SDN-defined Smart Factory Networking. Procedia Computer Science. 134. 235-242. 10.1016/j.procs.2018.07.166.

[6] L. Da Xu, W. He, S. Li. "Internet of Things in Industries: A Survey". IEEE Transactions on Industrial Informatics, vol. 10, no. 4, pp. 2233-2243, Nov. 2014. doi: 10.1109/TII.2014.2300753.

[7] M. Handley, A. Moore, and I. Steenkiste. "Packet Purse and Packet Wallet: Real-time Network QoS Provisioning with Generalized Edge-to-Edge Budgeting". In Proceedings of IEEE INFOCOM 2000, Tel Aviv, Israel, March 26-30, 2000. pp. 1478-1487.

[8] Lerner, A. "Getting Started with OpenStack". O'Reilly Media, 2013.

[9] J. Rosenberg, L. Peterson. "The Mininet Network Simulator". In Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Helsinki, Finland, August 13-17, 2012. pp. 365-366. doi: 10.1145/2342356.2342417.

[10] B. Sullivan. "OpenStack Operations Guide: Set Up and Manage Your OpenStack Cloud". O'Reilly Media, 2014.

[11] A. Botta, W. de Donato, V. Persico, A. Pescapè. "Integration of Cloud computing and Internet of Things: A survey". Future Generation Computer Systems, vol. 56, pp. 684-700, Mar. 2016.

[12] Kreutz, D., Ramos, F. M. V., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. "Software-defined networking: A comprehensive survey." Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, January 2015.

[13] Mijumbi, R., Serrat, J., Gorricho, J. L., Bouten, N., De Turck, F., & Boutaba, R. "Network function virtualization: State-of-the-art and research challenges." IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 236-262, First quarter 2016.

[14] Drutskoy, D., Keller, E., & Rexford, J. "Scalable Network Virtualization in Software-Defined Networks." IEEE Internet Computing, vol. 17, no. 2, pp. 20-27, March-April 2013.

[15] Y. Zhao, and R. Govindan. "Understanding packet delivery performance in dense wireless sensor networks." Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing, 2005.

[16] M. Reisslein, X. Yuan, and M. Masdari. "Cognitive-based edge computing to reduce latency and conserve bandwidth in Industrial Internet of Things networks." IEEE Network, 2018.

Glossary

1. IIoT: Industrial Internet of Things refers to the integration of internet of things technologies into manufacturing and other industrial processes to enhance efficiency, data analysis, and automation.

2. Latency: The time taken for data to travel from the source to the destination in a network. It is usually measured in milliseconds (ms).

3. Jitter: The variation in the delay of received packets in a network. It is the deviation from true periodicity of a presumably periodic signal.

4. VM: Virtual Machine is a virtualized environment on a physical computer that mimics a physical computer. It can run an operating system and applications as if they were running on a physical computer.

5. OpenStack: An open-source software platform for cloud computing, which is used to control large pools of compute, storage, and networking resources throughout a datacenter.

6. Mininet: A network emulator that can create a virtual network of hosts, switches, controllers, and links on a single machine.

7. Python: A high-level programming language that is used for general-purpose programming. It is designed to be easy to read and write.

8. KPI: Key Performance Indicator is a measurable value that demonstrates how effectively a company or a process is achieving key business objectives.

9. Sensor: A device that detects or measures physical properties and records, indicates, or otherwise responds to it.

10. Actuator: A type of motor or device that is responsible for moving or controlling a mechanism or system.

11. Optical Technologies: Technologies that are based on the properties of light and how it interacts with various materials.

12. Cloud Computing: The delivery of various services over the internet. These services include storage, databases, servers, networking, software, analytics, and intelligence.

13. API: Application Programming Interface is a set of protocols, routines, and tools for building software and applications.

14. Data Center: A facility composed of networked computers and storage used to organize, process, store, and disseminate large amounts of data.

15. Static Route: A route that is manually configured and entered into the routing table.

16. Resource Allocation: The process of assigning available resources to the needed tasks.

17. Service Optimization: The process of making a service as effective, functional, or useful as possible.

18. Network Emulator: A tool that can mimic the behavior of a real network by using software and hardware to simulate network elements, such as routers, switches, and links.

19. Resource Management: The process of planning, organizing, and allocating resources in the best possible manner.

20. Network Route Optimization: The process of finding the most cost-effective path for data packets to travel from the source to the destination in a network.