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

2022

Master Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
By
Murad Al-Smady

DEVELOPMENT OF A SECURE MONITORING FRAMEWORK FOR OPTICAL DISAGGREGATED DATA CENTERS

In partial fulfillment
of the requirements for the master in
Cyber Security

Advisors: Salvatore Spadaro, Albert Pagès Cruz
Barcelona, Date 13/9/2022

Acknowledgment

I would really like to convey my gratefulness to Salvatore Spadaro, who serves as my supervisor. During the period that I was working on my thesis, he provided me with direction. As a result, I was able to collaborate with several very knowledgeable individuals. Then, I would like to express my sincere appreciation to Albert Pagès and Fernando Agraz, patients and experienced guides whom encouraged and uniquely directed me. I would like to use this opportunity to show my appreciation to my family, namely my lovely mother, Nehaya, and my father, Mansour. It was because of them that I was able to pursue my passions without feeling alone, even while I was thousands of miles away from home. In addition, I would like to express my gratitude to my fiancee, sweetheart Rahmeh, who has been an unwavering source of encouragement and a loyal friend to me while pursuing my education and living my life. In conclusion, I would like to express my gratitude to everyone I had the pleasure of interacting with throughout the Master's program, including both old and new friends. To begin, I'd like to thank my roommates Bilal and Hani and Mayssaa. Without the assistance of these extraordinary individuals, none of this would have been achievable.

Contents

List of Figures

List of Tables & block codes

Revision History And Approval Record

Revision	Date	Purpose
0	15/07/2022	Document creation
1	05/08/2022	Document revision
2	25/08/2022	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
AlSMADI, MURAD	murad.al-smady@estudiantat.upc.edu
SPADARO, SALVATORE	salvatore.spadaro@upc.edu
PAGÈS CRUZ, ALBERT	albert.pages-cruz@upc.edu

Written by:		Reviewed and approved by:	
Date	25/07/2022	Date	31/08/2022
Name	AlSMADI, MURAD	Name	SPADARO, SALVATORE; PAGÈS CRUZ, ALBERT
Position	Project Author	Position	Project Supervisor

Abstract

The mismatch between the variability of resources required by running applications and the fixed amount of resources in server units severely limits resource utilization in today's Data Centers (DCs). The Disaggregated DC (DDC) paradigm was recently introduced to address these limitations. The main idea behind DDCs is to divide the various computational resources into independent hardware modules/blades, which are mounted in racks, bringing greater modularity and allowing operators to optimize their deployments for improved efficiency and performance, thus, offering high resource allocation flexibility. Moreover, to efficiently exploit the hardware blades and establish the connections across them according to upper layer requirements, a flexible control and management framework is required. In this regard, following current industrial trends, the Software Defined Networking (SDN) paradigm is one of the leading technologies for the control of DC infrastructures, allowing for the establishment of high-speed, low-latency optical connections between hardware components in DDCs in response to the demands of higher-level services and applications. With these concepts in mind, the primary objective of this thesis is to design and carry out the implementation of the control of a DDC infrastructure layer that is founded on the SDN principles and makes use of optical technologies for the intra-DC network fabric, highlighting the importance of quality control and monitoring. Thanks to several SDN agents, it becomes possible to gather statistics and metrics from the multiple infrastructure elements (computational blades and network equipment), allowing DC operators to monitor and make informed decisions on how to utilize the infrastructure resources to the greatest extent feasible. Indeed, quality assurance operations are of capital importance in modern DC infrastructures, thus, it becomes essential to guarantee a secure communication channel for gathering infrastructure metrics/statistics and enforcing (re-)configurations, closing the full loop, then addressing the security layer to secure the communication channel by encryption and providing authentication for the server and the client.

1.0 Introduction

The increasing importance of data centers (DCs) in supporting and sustaining rapidly rising Internet-based applications has spurred academic and corporate interest in DC infrastructures [1], as well as related performance assessment and improvement. Traditional DCs are becoming overburdened as workload and platform domain sizes expand, resulting in low network and server utilization despite the fact that more resources are being used for the same capabilities. To solve these restrictions, the Disaggregated DC (DDC) paradigm was recently introduced. The core premise behind DDCs is to partition diverse processing resources into independent hardware modules/blades that are mounted in racks, [2]. As a result of the increased modularity, DC operators will be able to optimize their implementations for better efficiency and performance, offering high resource allocation flexibility. Subsequently, the scope of the Data Center Network (DCN) is expanded to include optical networking that has emerged as the leading choice for meeting the demands of contemporary DC infrastructures such as DDCs, and fabric that connects the hardware devices that will design and build the compute node [3]. Moreover, the control layer will be based on the Software-Defined Networking (SDN) principles that will enable the establishment of high-speed [4], low-latency optical connections between hardware components in the DC in response to the demands of higher-level services and applications. Operators are now advocating for a more flexible approach, enabling access to low-cost solutions and achieving high performance through network disaggregation. The scalable disaggregation and composition impacts include using fewer assets to serve the same demand, so servers become highly interchangeable, storage is pooled and thin provisioned. Considering this, the degree of pressure on new and future requirements is being reduced.

However, disaggregation must not compromise performance or utility, and it must be invisible to the workload. It is also required high-speed storage without scale or deployment constraints, as well as low-latency, congestion-free. The prerequisites for successful disaggregation are scalability, network performance, and ease of deployment. On the basis of the preceding information, the primary purpose of this thesis is to monitor and to utilize the resources of a DDC infrastructure to the greatest extent feasible, while also providing a secure framework for quality assurance loops.

1.1 Data Centers

Almost each organization, regardless of size or profession, requires a DC, which is a physical structure where enterprises retain their data and other applications that are essential to their operations. Nonetheless, in its most basic form, a DC is a physical structure utilized by enterprises to store data and vital applications. The architecture of DC's is based on a large number of nodes (i.e., servers) that are linked together by a network that interconnects storage and computing resources in order to enable the distribution of shared applications across the shared physical infrastructure.

A DC infrastructure is considered one of the most sophisticated systems [5] since it may comprise Network Infrastructure DC Components such as routers, switches, security devices such as firewalls, intrusion prevention systems, intrusion detection systems, Storage Infrastructure DC Components such as Direct-Attached Storage (DAS), Backup Storage, Storage Area Network (SAN), Server Infrastructure DC Components such as physical servers racks, blades, and cabling. Steady power is an important aspect of ensuring uptime in DCs, which essentially connect to the district electrical grid and most even have some type of power failover and backup source of electricity. These considerations, such as electricity and cooling for humidity levels and temperature regulation, are crucial in order to enable the efficient operation of the DC infrastructure since the operations performed will generate a lot of heat and, as a consequence of that, a system to control and regulate these factors (i.e. temperature, humidity) is required. Higher efficiency and lower expenses can be achieved while providing DC administrators' a comprehensive perspective of the DC to properly manage the operational loops of the infrastructure and implement services/applications thanks to the utilization of DC Infrastructure Management (DCIM) [6]. This would aid in managing business challenges that are readily associated with current technology constraints. Companies contemplating such solutions must be prepared for DCIM by building a strategy that covers the most critical concerns as shown in the Figure 1 while keeping focused enough to be practicable by adopting DCIM, delivering immediate and sophisticated knowledge that promotes energy efficiency and strategically plans for future capacities, such as power, cooling resources, and space. Then that will monitor a DC's assets, resource consumption, and procedure status. The data will then be analyzed and acted upon to meet commercial and corporate goals while maximizing performance. The DCIM system enables DC managers to carry out technical and organizational objectives and improvements, decrease waste and unneeded overprovisioning, identify additional capacity, reduce energy consumption and also enhance energy efficiency.

As the duties of maintaining and administering a DC infrastructure grow, there are several chances to reinforce and optimize the overall DC environment with increased efficiency. Once a DCIM is planned and incorporated in a DC infrastructure, major criteria for indicating resource and competency improvements include power and environmental tracking (monitoring), capacity and schedule planning, and management of assets; these are the most essential DCIM characteristics. DCIM might help DC administrators manage their infrastructures more efficiently and effectively by easing the process of selecting and evaluating a solution that meets many of the principles and suggestions mentioned in this thesis project. The correct DCIM solution ought to be responsive to future DC growth and a crucial instrument in achieving development as quickly as feasible.

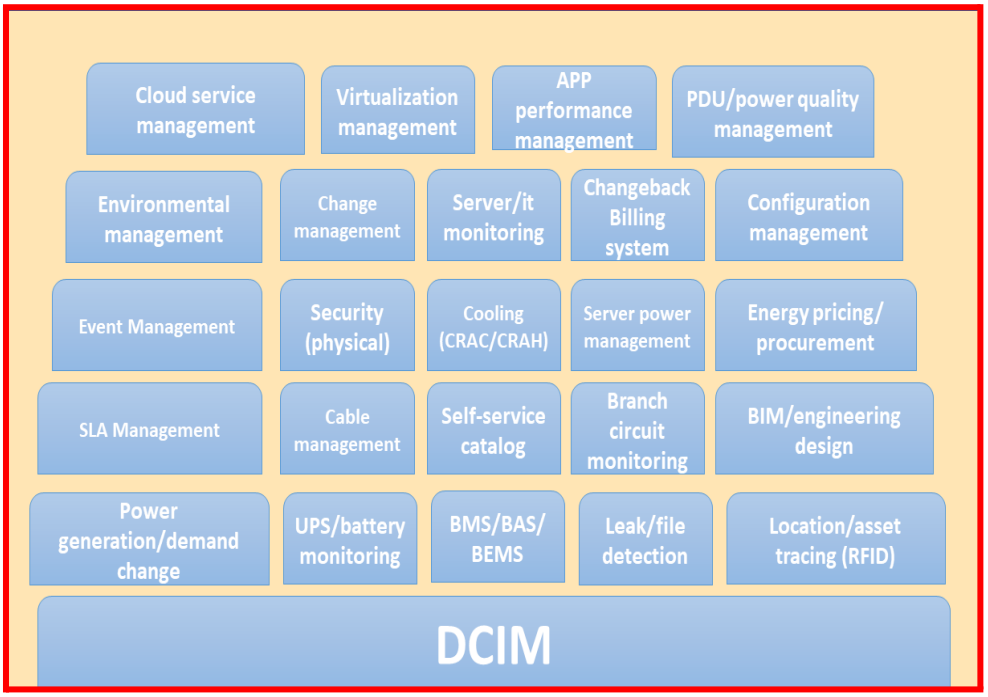

Figure 1: DCIM Business Process

1.2 Data Centers Infrastructure Types

1.2.1 Traditional Data Centers

The main idea is to host a significant number and medium-sized applications, typically running on their own specialized hardware infrastructure that would be separated and secured from other environments in the same facilities, often for a number of organizational divisions or businesses. The DC architecture is built on a tried-and-true layered methodology that has been tested and developed over numerous years in some of the largest DC deployments. The layered approach is the basis of a DC design that maximizes scalability, performance, flexibility, resilience, and

maintenance. The verified network architecture is made up of DCs that connect and communicate with one another via the corporate core network. Each DC is constructed on a multilayered architecture that incorporates routers as well as core, access and aggregation layer devices Each component operates inside its own separated layer, which has its own administration on a different level. This indicates that in order to handle each type of hardware equipment, a specialized team comprised of individuals who possess the necessary expertise is required. For instance, the management of storage, servers, and networking all need to be handled by at least three or four independent teams. In addition, there is still a need for some extra cross functional groups in order to deal with other infrastructure layers such as virtualization [7], security, and so on. Additionally, there has to be a group that is responsible for monitoring the warranties associated with various purchases made from various retailers.

Because most of these components of a typical infrastructure are put together as seen in Figure 2 the Operations, Deployment, and Administration become more complicated, and the supply of IT services and applications is slowed down. This type of setting is not only costly, but it is also tough to match and needs expertise, which are tough to locate and much harder to obtain. Since there are many different types of hardware, software, and firmware, the procedure has to be carried out by several groups. This makes the whole process exceedingly complicated, that in turn leads to a large amount of cooperation and laborious administration. Traditional infrastructures were already in existence long before the concept of virtualization was even conceived. During that time period, it was possible to simply characterize the situation as one application running on each dedicated physical device. The introduction of virtualization, on the other hand, brought about a paradigm shift in how traditional DCs were understood. The traditional infrastructure is composed of independent components, which often originate from a number of different suppliers and function only inside the confines of their respective silos. Application distribution is often built on multi-tiered techniques, with infrastructure installed and configured to serve certain requirements. In spite of the fact that this strategy offers a significant lot of adaptability, maintaining it is difficult and expensive, and it continues to present suitability and integration challenges. Despite this, the conventional infrastructure was able to meet workload needs well enough up to the introduction of new applications, which were far more complicated, dynamic, and resource-intensive. Not only did they demand higher agility and scalability, but they may also be regionally scattered, which made conventional structures more burdensome than they had ever been before.

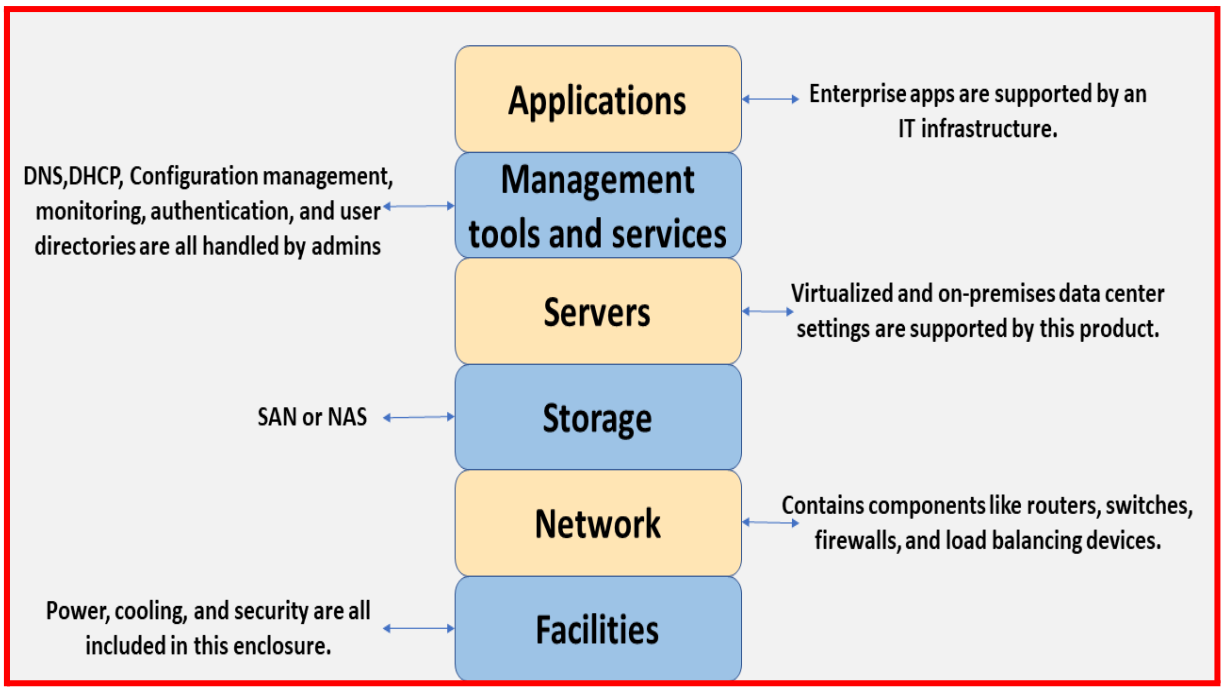

Figure 2: Typical Layer of IT infrastructure

1.2.2 Converged Infrastructure (CI)

CI in DCs solves many of the problems that are caused by the restrictions of old infrastructure and multi-tiered architectures depicted in the both of which are getting increasingly challenging and costly to manage [8]. The use of CI technologies may assist in improving the efficiency of IT operations, increase the use of the resources available in DCs, and reduce the costs associated with establishing and operating infrastructure. Storage and network resources are integrated to a varied degree in every CI; this provides a straightforward approach to the enhancement of resources and is also simpler to administer than traditional infrastructures. If the infrastructure requires to expand out, all it has to do is add more appliances. But despite this, each of the components may still be utilized for the task that it was designed for. For instance, the server and the storage space may be partitioned out and used as two different functional entities. This indicates that there has not been a significant leap in performance. On the other side, there are a great number of advantages that just cannot be ignored, such as the following:

- Straightforward to deploy: The task of the commercial team will thus be simplified as a result of this.

- Enhance teamwork performance: Implementing CI ensures that you can concentrate on enhancing your utilization of that system environment.

- Better management: Several systems come with management solutions capabilities that can handle the whole system.

- Decreased the complexity: This type of architecture has a smaller environmental footprint, involves minimal cabling, and could be installed much more quickly than traditional infrastructure.

Nevertheless, CI remains a hardware-centric approach, despite it having several attractive features. In the face of today's industrial problems, it was unable to give the necessary adaptability. Because of the granularity with which CI used to streamline operations, reduce duplication of effort, improve efficiency, and cut down on expenses associated with managing IT assets which allows IT settings to be adjusted, they are well-suited to businesses with stringent control requirements. Large corporations that are updating their complete IT infrastructure may also benefit from them since they eliminate the need to negotiate around for individual parts, for corporations with a consistent environment and the capacity to manage massive operations, traditional infrastructures remain a viable alternative.

1.2.3 Hyper-Converged Infrastructure (HCI)

The next step in the development of CDI is the introduction of HCI [9], that also expands the capabilities of CI through the addition of a software defined layer that manages to combine storage, network components, and computing power into pools of managed and controlled resources. HCI provides a higher degree of interconnection across the components, as well as a higher percentage of simplification and automation, that further will help to simplify implementation and administration even more. Each pool of servers is responsible for virtualizing the underlying computational resources and laying the groundwork for storage to be pooled among all nodes. A HCI product consists of individual units that are completely discreet and come pre-configured and improved for a particular task. Because of this structural system block and the fact that its physical structural components have been abstracted, it is much simpler to implement and scale up the computing resources than it does with traditional or CI architectures. This leads to processes that are carried out more quickly, with sufficient agility, flexibility, and with much less sophistication. Furthermore, many HCI systems are equipped with built-in safeguards against the loss of data as well as disaster response capabilities. However, it brings with it a number of obstacles and challenges to overcome. For instance, the computation and storage resources are consolidated onto a single unit. The only method to grow such systems is to add units/nodes in their whole, which may lead either to over-provisioning the computation resources. However, this is the only option to scale and expand these systems. Furthermore, HCI is often configured for certain workloads, which limits its mobility to serve specialized applications. Moreover, much as with CI, it might be challenging to steer clear of becoming

locked in with a certain vendor while using HCI. Policy-based configuration eliminates the need for sophisticated constructions in the design process. The growth of the Internet of things (IoT) sensors and devices necessitates companies to reconsider data collection, storage, and computer processing. Throughout many situations, this means moving processing and data analysis from the centralized DC to the periphery or edge. HCI simplifies remote deployment and administration compared to conventional infrastructure systems; Figure 3 shows the comparison between the different types of DC infrastructure.

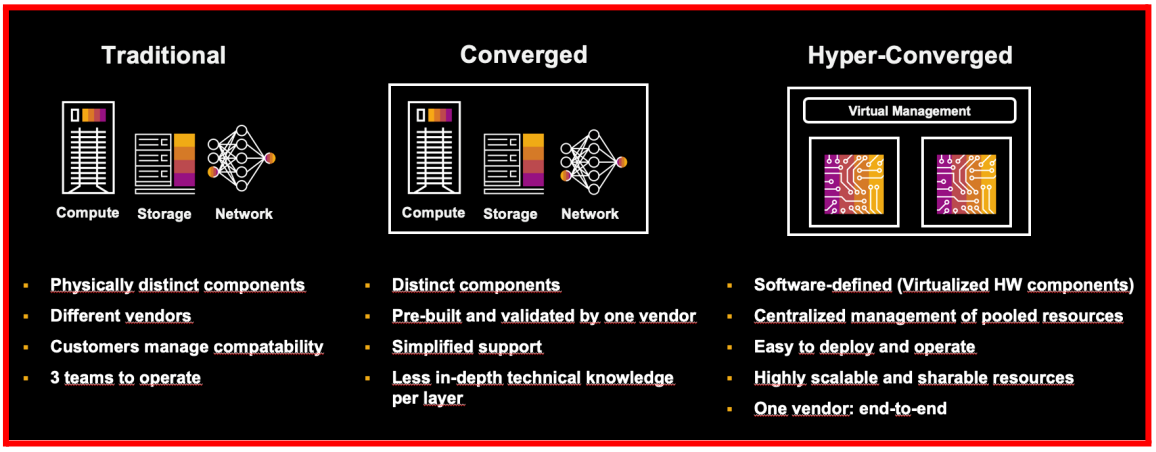

Figure 3: Traditional Infrastructure Vs Converged Infrastructure Vs HCI [34]

1.2.4 Composable Disaggregated Infrastructure (CDI)

The next step in the growth of DC convergence is recognized as CDI [10]. This step brings together the most beneficial aspects of HCI and CI, enhancing their combined features to provide a setting that is entirely controlled by software. The computation, storage, and network resources are all separated by CDI and then delivered as services that really are capable of being continuously and dynamically assembled and rearranged to suit the evolving requirements of individual applications. In addition, the infrastructure offers a complete application programming interface (API) for administration, which admins and programmers may use to orchestrate and automate various tasks, as well as provision resources. Composability utilizes resources in a really effective manner than either HCI or CI, so it is uniquely qualified to support a wide variety of tasks that are changing all the time. As illustrated, the first-generation of HCI and CI technologies can only really manage virtualized varieties of tasks. Containers are currently supported by a significant number of the more recent systems; nonetheless, mainly CDI is capable of running apps inside Virtual Machines (VMs), and containers as seen in Figure 4.

Figure 4: Compose server characteristics

CDI is indeed effectively able to support apps that are currently in use. Its extensive API makes administration easier and functions more streamlined, removing most of the implementation and enhancement overhead that has been associated with using alternative infrastructures. Meanwhile, CDI is still a very modern technology; as a result, there have been no benchmarks for it, and there isn't even a consensus on how to define it. Composable providers remain on their own to develop their CDI products, and each must adhere to its own distinct standards in order to do so. It not only enhances the potential risks of lock-in, but also causes incompatibility concerns across vendor solutions for an enterprise adopting CDI technologies from different suppliers. The automated features of CDI could indeed assist IT workgroups in streamlining their processes, as the API allows developers to interface with 3rd-party management solutions. CDI may help applications like Machine Learning (ML) and Artificial Intelligence (AI), which may need provisioning to accommodate numerous functionalities along with data volatility. Comparison of HCI and CDI is shown below in Figure 5. The processes of purchasing, deploying, and managing HCI systems have also seen a significant change in recent years. The creation of next generation HCI technologies, commonly referred to as HCI 2.0 [11], is perhaps the most illustrative example of that kind of transition.

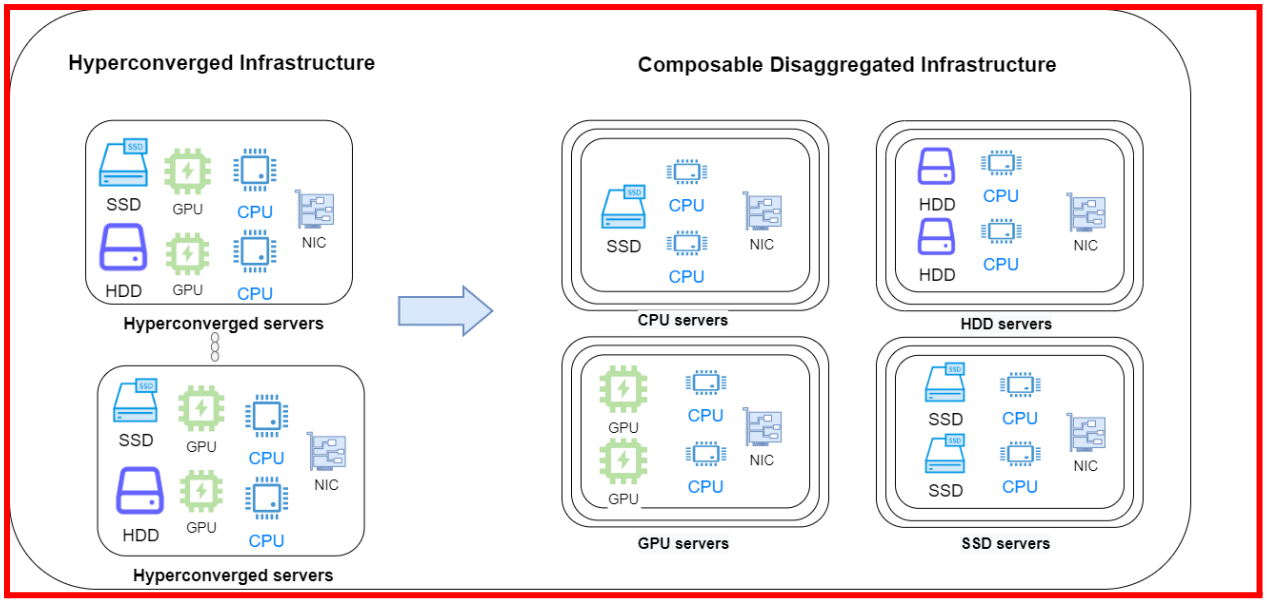

Figure 5: HCI Vs CDI

Hardware disaggregation is an essential aspect of HCI 2.0, and it is also one of the main qualities instead of aggregating/integrating the hardware resources at the cluster level. This has the consequence of allowing hardware resources to be assigned separately, hence preventing any resources from being stranded or being squandered. To put it another way, it provides an opportunity for DC infrastructures to ramp up both their compute and storage capabilities while still operating inside an HCI environment. It thus gives it more elasticity and frees up its agility. It opens the door for HCI to develop further, which in turn removes obstacles regarding performance and scalability.

To support new cloud-native workloads, DC infrastructure must provide a graphical processing unit (GPU), central processing unit (CPU), memory, and network resources formerly allocated for experimental and web server apps. Even with elevated power consumption as well as higher prices, the fixed hardware of the centric server leads to insufficiency of hardware resources. Decoupling all resources from the server and then replacing the onboard bus connector with network interconnection has recently been presented as a solution to these concerns. As a result, the definition of the data center network (DCN) has been expanded such that it now includes the fabric which will link the various pieces of hardware that will make up the computational nodes. Utilizing distinct hardware resource pools/groups allows DDCs to maintain both excellent performance and power efficiency while operating and elastic resource provisioning for such a variety of apps by providing the separation of the computational resources of servers into independent hardware modules for their individual exploitation. Despite this, there are quite a number of obstacles to overcome before DDC may be implemented. Consequently, a network that has both low latency and large bandwidth is essential for the disaggregated architecture. Furthermore, there are many more hardware nodes (i.e., blades) that need to be linked, which necessitates the use of an interconnected network that is expandable. However, the increased number of hardware blades makes it tough to discover the ideal mapping between upper

service/application requirements and resource utilization, as well as the changing needs of applications call for a real-time orchestration system in order to reallocate resources. The granular decomposition of DC resources and the grouping of those resources into adaptable groups have the opportunity to deliver such a substantial change in the environment. Even if storage is now separated into its own resource groups/pools in today's DCs, primary accelerators and memory are still physically connected to the server platters. The centralized technique will almost surely result in the most effective use of the accessible resources, but it also runs the danger of establishing a hot spot problem as well as a substantial performance bottleneck in the process. The real-time monitoring and learning from the impacts of actions made could be used to help automating the allocation of network resources that can meet new demands as well as continuously optimize current service requirements, such as a cloud computing platform. As a result, an essential part of the disaggregated network is (optical) performance monitoring, which is intended to provide the response that is required to ensure the quality of service and quality of transmission of the optical connections between hardware blades and composed servers [12]. The procedure for scheduling and provisioning resources begins further with requesting unit nodes to identify the kind of resource and the quantity that is needed. Then the quantity of necessary resources may be determined either directly by workload or intuitively also as a particular querying node runs out of resources. Both of these methods are valid. Once the request has been made, the resource provisioning mechanism will select either one of the resource groups with required resources in order to provision resources. After that, it will engage in communication with the resource management of the appropriate resource group in order to request the resource. If there are several requests for a particular resource flowing in at the same time, the resource manager for every pool of resources will devote its resource to the incoming request as well as handle any possible struggle that may arise. It thus may need the resource scheduler to examine resource utilization statistics to provide an appropriate workload-dependent prediction. Administrators are able to attain new heights of efficacy and adaptability by disaggregating a specific network function through its component pieces of hardware and software components and then utilizing the characteristics of software to deploy a certain OS (Operating System) onto open white boxes. In addition to this, the seek for the implementation of software-orchestration so that the control-layer might well be abstracted from the physical layer as well as the system could indeed acquire automation and software-based control functionalities [13]. Disaggregated networks might indeed be the ideal option to satisfy the requirements of the modern world; yet, one of the most significant obstacles is the effect that integrating disaggregated unit nodes towards the architecture of the network would have on the network's performance. A performance trade-off would be provided in exchange for interoperability. More precisely, once trying to navigate disaggregated endpoints, the transmission losses of the light-path rise related to the shortage of enhancement and optimization of the multiple-vendor endpoints, which further are unable to satisfy an enhanced spectral efficiency that would only be accessible through fine-tuned single-vendor systems. This is because those certain multiple-vendor endpoints are made by more than one corporation. There must certainly be other problems as well, such as the incorporation of the systems using accessible elements and their management, and most crucially, the pace and adaptability of the optical market in terms of issuing specifications and functioning tools. These difficulties must be overcome. Due to the increasing significance of disaggregated optical networking, doing

research on the aforementioned concerns is both necessary and fascinating. As per [14], the increased transmission inefficiencies that occur through the light-path as a result of integrating disaggregated gear together with the networks need the implementation of more re-generator units in comparison to a system that is supplied by a single-vendor. This would be due to the additional flow losses resulting in the lowering of the maximum data transmission distance without regenerating the optical signals (Transmission Reach). The cost of optical re-generators is often relatively expensive, which means that carriers need to determine whether or not such a financial setback is more than the anticipated advantages brought about by the disaggregated strategy. This is the most significant challenge.

The disaggregation approach was initially implemented in DC setups with the intention of enhancing the usage of computer resources in consideration of the rapid rise that the industry was experiencing at the time. As can be seen in Figure 6 (a), the present "server-centric" DCs are comprised of CPU, memory, and storage resources that are all integrated onto a motherboard. However, different services, including cloud computing and big data analytics, often need a variable percentage of resources, which causes poor resource utilization in "server-centric" DCs [18]. In this specific situation, the idea of rack-scale DDC has been the focus of considerable research in recent times, with the overarching goal of enhancing the effectiveness with which DCs make use of their available resources. As can be seen in Figure 1a, the present DCs make use of a Top of the Rack (ToR) switch to link the integrated servers located in a rack to one another. Blades in DDCs, each of which contains a single resource (such as a CPU, a memory module, or a hard drive storage space), are connected to one another by means of a switch that must be capable of supporting high-capacity transmission. Given this requirement, it is clear that optical technologies will need to be implemented. The showcases a rack of a DDC following the rack-scale approach is shown in Figure 6 (c). Because a resource blade can store more of that resource type than an integrated server can store of the same resource type, it offers greater flexibility when it comes to allocating resources. A particular example of a job that may be performed in a DDC can be shown in Figure 6 (b), which depicts virtual machines (VMs). The activities that are being carried out are able to see through DDC's underlying architecture. It indicates that the connections between the CPU, memory, and storage, which are typically implemented on the buses of the motherboard in a conventional server, were already handled through external optical connections between resource blades in the rack of the DDC.

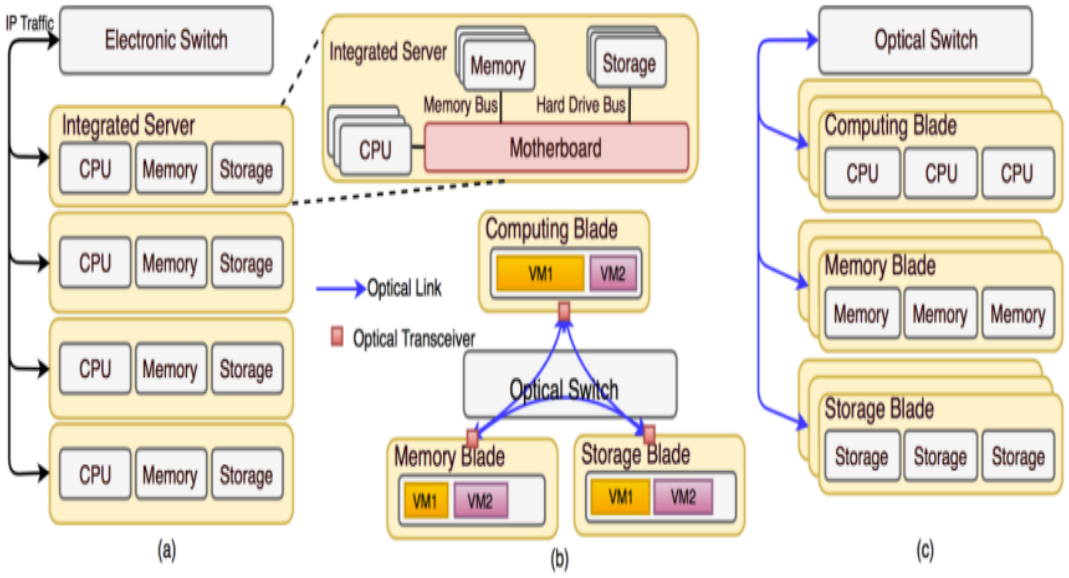

Figure 6: (a) A Rack in Traditional DC (b) Virtual Machine Deployment in DDC (c) Rack-scale, [17]

DC infrastructures are made up of lots of servers and workstations which together closely combine a tiny portion of the different resources required for a computing function [15]. Those computational capabilities have all been tied to a particular motherboard, which means that it is only possible for the apps operating on a particular server to make use of them. DCs are able to concurrently conduct challenging tasks and handle dozens of users at the same time. However, in order to make full use of their potential, DDCs are the alternate configuration to traditional integrated DCs. At this point, DC administrators already have comprehensive and precise control over resources over tasks as a result of the usage of a new configuration in which each resource type is constructed as a separate resource blade. A fundamental facilitator of DDC is a network that could satisfy both latency and bandwidth needs of communication within the server in a DDC situation, including different resource blades linked through a network fabric. Figure 7 shows how the computational resources are tied to specific hardware boards between the integrated amd the disaggregated blades, dedicated fiber links are established between the various pieces of hardware blades which relate to a certain rack.

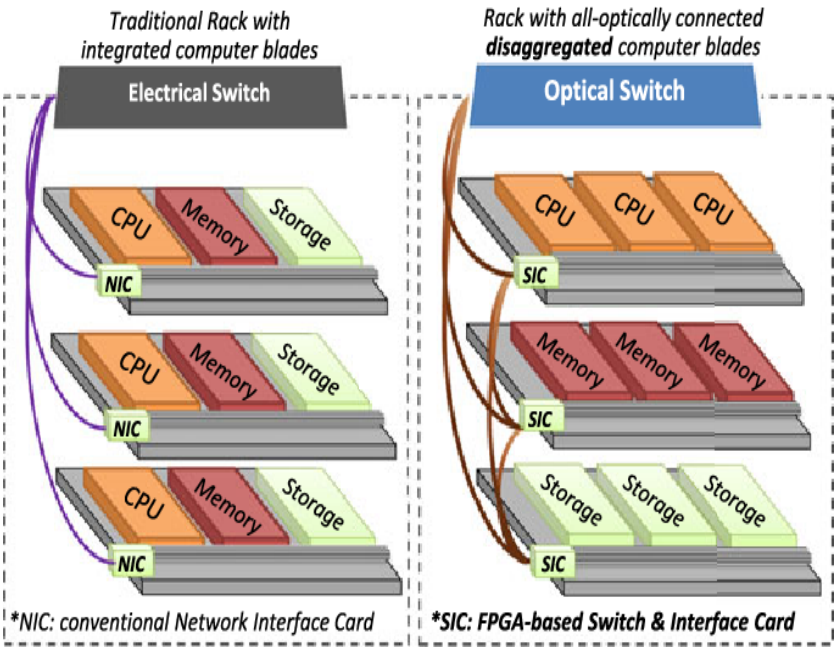

Figure 7: Integrate computer blades Vs disaggregated computer blades [15]

Applying disaggregation to future DCs carries with it a number of essential benefits, the most prominent of which is increased utilization effectiveness, decreased energy usage, and more agility and flexibility. To be more precise, we could consider a DDC infrastructure, which is characterized by the organization of computing resources in the form of hardware blades stored inside racks. This will proceed under the assumption that each rack has blades representing all three categories of computing resources (storage, memory, and CPU cores) in order to simplify our analysis.

An optical to electrical and electrical to optical conversion of the signals arriving from and departing to the hardware components of the blade. In addition, a collection of fiber connections is used to connect and link the many pieces of hardware blades that are housed inside a rack as well as those that are located in other racks. When interfacing amongst the different pieces of hardware that are housed inside the same rack, this will provide the ability to cut down on the amount of latency that is incurred as well as the bandwidth restrictions that are imposed on the connection. In terms of the communication between racks, a wavelength division multiplexing (WDM) optical network is used to connect all of the different racks together. Moreover, an optoelectronic top of rack (ToR) switch has been installed in each and every one of the racks, the type of switching depends on the architecture and technology of the DDC, some may employ circuit switching, but there are several proposal that advocate for the use of optical packet switching (OPS). The circuit-switched WDM inter-rack network fabric is what connects the hardware blades of one rack to the hardware blades of certain other racks, and this switch is the component responsible for making that connection. Communication between racks is handled by a series of optical switches. This allows for unhindered data transfer. Thanks to these networking elements, connections between the many pieces of hardware blades that make up the DCs architecture may be created with higher bandwidth and an extremely low latency [15]. The

allocation of a Virtual Data Center (VDC) is just one of the many services that can be deployed in DCs, not only disaggregated but also integrated ones; it provides for the provisioning, including both network resources and computation resources, as well as the requisition comprises of a group of virtual machines (VMs), each of which asks for a certain quantity of computing resources. Afterward, the virtual connections with the required bandwidth are created in between VMs so that they may communicate with each other. A specific rack's physical resources were used to transform these VMs. This would be attributable to the fact that translating or mapping them across many racks might result in undesirable latencies between the various computing resource units. In particular, we consider the situation in which the VMs that correspond to the same occupant are translated into various racks. This is done in order to offer increased robustness in the event that a hardware component fails. Concerning the virtual links, each one of them requires a particular bandwidth in regards to wavelength channels. Therefore, in order to complete the virtual link mapping, it's indeed essential to address a problem known as Routing and Wavelength Assignment (RWA) in a transparent optical network as well known as the inter-rack network fabric. This involves deciding the necessary lightpaths in order to fulfill the connectivity requirements that are stipulated mostly by virtual links. Because of the straight by-pass linkages between different hardware blades that are included within the same rack, there are still no bandwidth or latency constraints for such intra-rack interaction. Instead, the emphasis should be placed on selecting the network bandwidth for the inter-rack segment. It is important to keep in mind that the possibility of making such a premise exists due to the fact that we have been taking into consideration the fact that racks retain hardware blades representing all different kinds of resources and that the VMs are mapped and translated over thought the resources of the same rack. Dedicated fiber links are used to connect and interact with the various hardware blades, high-efficiency speed and low latency communications are accomplished within the components of a single VM. As a consequence, the components of the VM successfully function as a single constructed component, seeking to avoid any bottleneck for intra-rack communication channels, as can be seen in [15] and [16]. Consequently, the mapping and the translation of the virtual link correspond to determining the light paths between such the source and the destination ToRs of the racks upon whom the VMs of the virtual link were mapped. In addition, to accomplish this, it is necessary to locate a route that extends from source to destination and also a wavelength channel that becomes continuous across all of the fiber connections that make up the way. Furthermore, a lightpath and the same wavelength channel cannot indeed be allocated to the same physical connection at the same time due to wavelength clashing constraints. In order to maintain the separation between occupants, However, according to the capability of multiplexing offered by optical networks based on WDM, distinct lightpaths are able to operate in the same physical connections without interfering with one another. Memory storage disaggregation in distribution DCs and cloud data centers was addressed by the authors of [27] via the presentation of an SDN architecture. This technology is adaptable enough to be employed in both conventional distribution centers and cloud-based ones. This paradigm was investigated using an approach that took into account the whole of the system. Because it has a control plane that is centralized, it is endowed with the capability to continuously govern a number of decentralized resource groups. These resource groups are in charge of computation and memory storage, and they are accountable to the centralized control plane. These resource pools or groups are actualized in the real world as standard modules that may be fitted into rack

trays. These modules can be used to build computer systems. The key goal that they want to reach with this assignment as their primary concentration is to provide a cutting-edge disaggregated memory controller (DMC) architecture. This is the primary aim that they want to attain with this project. Memory sections will have the capability of being unilaterally allocated to computational nodes, and those computational nodes will have the ability to access those memory sections in line with the requirements imposed by the tasks. The overarching design of the building's architecture will make this possible. The exposing of the disaggregated storage memory that would take place as part of the workaround would be quite transparent from the perspective of the CPUs. This shows that the system is powerful enough to run unpatched app binaries while they are being stored in the disaggregated memory space. It is correct that this feature cannot be found in any other product or service on the market that is even somewhat equivalent to what they provide. According to their observations, disaggregation is a viable method for the future generation of DCs, as shown by their findings. In addition, they estimate that fabrication among installations of their architecture will significantly reduce remote memory store accessibility overhead when compared to their advanced technology demonstration. This may be done, for example, by significantly simply expanding the number and capabilities of transmitters, therefore boosting its operational clock rate.

The authors of [28] investigated and explored the concept of DDCs, which are projected to attain much-enhanced resource effectiveness as compared to DCs based on comprehensive servers. Consequently, even when the optical networks allow for higher transmission capacities and bandwidths, it is still not fair to believe that the capacity for intercommunication between resources is limitless. This is the case even though optical transmission is at a very elevated level. They demonstrate that the average benefits of total DDCs may be reduced or even rendered nonexistent with bottlenecks imposed by CPU-to-CPU and CPU-to-Memory communications that optical networks may still not be ready to meet. This is because the most cutting-edge optical fiber communication technology cannot provide a significant amount of bandwidth, which is the reason why this situation has arisen. As a result of this, it is self-evident that research into more cost-effective short-range optical communications that provide a wider bandwidth has to be carried out. In particular, the repercussions on latency and power consumption caused by the introduction of capability disaggregation are not taken into consideration and need to be examined further. In addition, the relevant techniques of allocation of resources for the applications/services deployed in the DDCs need to be explored. The disaggregated architecture offers a potential solution to solve the difficulty of accessing massive data between the weakly connected storage nodes and for better monitoring and management. at the same time. DCs that are built on this architecture make it possible to reorganize the DC in order to increase its operational efficiency and decouple innovation cycles among its components, while at the same time, the DCN transforms into the back-plane of the DC. The authors of the research paper [29] provided a qualitative review of the techniques and problems for utilizing disaggregated architecture for at-scale cloud data centers. In particular, they evaluate and contrast the data models for the different hardware devices that might be used in order to get access to the disaggregated resources, as well as the consequences for the network and the provisioning and administration of the resources. They concluded that a disaggregated

architecture with appropriate programming models and resource provisioning would likely achieve improved DC operating efficiency for heterogeneous workload environments. These environments can benefit from the improved elasticity of physical resources. This conclusion was based on a qualitative assessment as well as early experimental results. The conventional DC architecture was faced with several issues as a result of the fast rise of workloads associated with cloud computing, both in terms of the spectrum and the volume. The first challenge is the rapidly shifting configuration prerequisites because of workload restrictions. The second challenge is the widely different invention cycles of the components of the system. The third and final challenge is the maximum peak sharing of the systems and the subsystems through obtaining excellent efficiency.

The authors in this research paper [30] describe an intent-based SDN controller and orchestrator capable of managing the complexities brought upon by the optic DDC model separately, mostly from physical infrastructure. They demonstrate the mechanisms of control management and orchestration involved in the delivery of Cloud Data Center as a Service (CDCaaS). An emulated DDC infrastructure centered on rapid optical switching has been deployed to exercise the conceptualization process. Precisely, both technologies, known as Nanosecond Optical Switching (NOS) and OPSquare, are replicated to create the Composite Compute Nodes Networks (CCN-N) and CDC inter-connection network (CDC-N) accordingly. The SDN controller provides modules and processes that allow the separation of the data plane to make the architecture-independent from the many technological solutions that have been presented for the DDC infrastructure. An extended reachability graph has been developed to adopt best practices. Its network contains many kinds of information technology equipment (such as CPUs, GPUs, memory blocks, storage units, and network interface card (NICs)), and its linkages reflect the extent to which such parts may communicate. Whereas the nodes are parameterized depending on the type and computational characteristics, the connections simplify all possibilities to join two end-nodes together. As a result, the linkages provide a concise summary of the many lightpaths that may be created at the data plane level between the connected nodes. These connections are determined by a number of wavelength combinations that are available, the throughput linked with them, and a distinctive feature that may be assigned to each one sums increase in the cost of utilizing that lightpath. It is crucial to remember that such a connection characterization correlates to the various intentions that might be utilized to join the concerned nodes on the data plane. Afterward, a query made by two end-nodes linked to one another is converted into a service pattern before being delivered to the CDC-Man at the orchestrator via Representational State Transfer (REST). This is done as part of the experimental validation. Given these criteria and the knowledge about the virtualized data plane obtained from the SDN, the CDC computing method determines which components are the most relevant ones to be employed. The algorithm's results are then converted into a collection of SDN intentions. There is an intention specified for each CCN, as well as an intent defined to set up the interconnection of the CCNs. Using the Northbound interface (NBI), which is responsible for putting the Transport API protocol into action, these intentions are communicated to the Provisioning Manager (PM) of the SDN architecture. After the PM has processed the intentions, any

necessary low operational level configuration setup queries are transmitted to the SDN agents of such associated data plane components. As shown in their illustration, the low-latency gRPC solution is used to implement the South Bound interface (SBI) communication protocol to assist with the setup and monitoring of the data plane. A feedback loop is essential for the development of self-adaptive software. When it comes to self-adaptive software and autonomic computing, a well-known feedback loop called the Monitoring Analysis Planning Execution (MAPE) loop is often employed in the adaptation process, the monitoring process is accountable for the gathering of data describing the environment and any internal changes. The following actions comprise the loop:

1) The monitoring process is responsible for gathering and correlating software and environment data.

2) The analyzing process is accountable for studying adaptive symptoms through data monitoring.

3) The planning process is responsible for defining the necessary adjustments and their implementation strategies.

4) The implementation of the adaptation strategy is the responsibility of the executing process.

The SDN-enabled customizable DDC infrastructure depending mainly on NOS has been experimentally proven by the authors in [31], who state that they did so. The data plane incorporates two CPU endpoints, two memory endpoints, and one NOS with four ports. All hardware nodes are connected using the Peripheral Component Interconnect Express (PCIe) Generation-3 bus. The CPU and memory of the micro-board are used as hardware end-nodes, as well as the field programmable gate array (FPGA) is utilized to construct the computational modules. In order to facilitate quick clock and data restoration, the switch controller ensures that the processing clock frequency of every one of the hardware end-nodes is synced. Polling requests are sent out at regular intervals by the SDN controller in order to keep track of various network characteristics. These packets are sent by the control plane to set up the DiCNs, together with the acknowledgment signal sent by the SDN agent. Therefore, as a result of the automated resource provisioning, the model that has been created is able to reconfigure the allotted nodes in a flexible manner in accordance with the metrics of the network. As a consequence of surveillance and dynamic resource reallocation, the disaggregated data plane network was capable of achieving an end-to-end latency improved performance that was 21 percent higher than before. This was accomplished by accessing memory nodes more quickly. There are many benefits and effects of thr scalable disaggregation and composition such as fewer assets to serve the same workload as servers become highly interchangeable, storage pooled and thin provisioned, it provides the automation, relieves line of sight pressure on emerging and future requirements and also no forklift upgrades (independent asset, deployment life cycles between servers, GPUs, and storage).

The authors of [32] discuss the benefits of resource disaggregation and recommend accurate methodologies of integer linear programming (ILP) as well as a method based on heuristics to determine the minimum amount of computational resources required to fulfill a recognized VDC demand founded on the base of a common DC infrastructure. When it comes to assigning VDC events on the basis of a shared infrastructure facility, the alternatives that they have provided suggest that DDC designs may undoubtedly assist in cutting down on the number of computing resources that are required, which need the concurrent allocation of network and computing resources. However, conventional DC designs may result in the need to overprovision server components in order to satisfy the translation of a collection of requirements. As a result, causes an increase in the capital expenditures (CAPEX) expenditures connected with the related spending. For example, it is feasible to conduct experiments with average declines of about 46 percent for VM profiles that are well balanced, although the reductions may rise to as high as 60 percent in the case of highly specialized VMs. These findings provide evidence that DDCs are able to circumvent the limitations imposed by previously developed designs in terms of making effective use of available computing resources. The prerequisites for successful disaggregation such as the disaggregation must not come at the expense of performance and utility, it must be invisible to the workload, as who is consuming it is different that who is providing it, and also prevent fabric fatigue and exacerbating skills and staffing shortages.

1.3 Software-Defined Networking (SDN)

It is important to have an autonomous control framework that allows the (re)configuration and monitoring of DDC resources as part of QA (Quality Assurance) operations, the SDN is one of the leading technologies for managing cloud environments like DCs and DDCs, It is a control paradigm that promotes a flow-based network architecture and the decoupling of the control and data planes [19]. When SDN was already established, it was influenced by the upheaval that transpired in the computer world industry in prior generations. The data planes and control planes are detached, while conventional networks use switches and routers to perform functions that place strain on them. The component of the control plane that is responsible for making decisions is isolated from the data plane and relocated to a central control point known as the SDN controller, which is responsible for handling and managing the network as a whole in a unified manner. Since this controller enables programmability on the network, it is classified as a "Software Defined." Through the use of a clearly-specified API, the SDN control plane as shown in Figure 8, shows that there are several abstraction levels, interfaces, and separate planes. It is able to exert effective direct control on the condition of the data-plane components such as routers, switches, and other network devices. Every single network operator is allowed to choose their own unique design principles and then construct their networks in accordance with those choices.

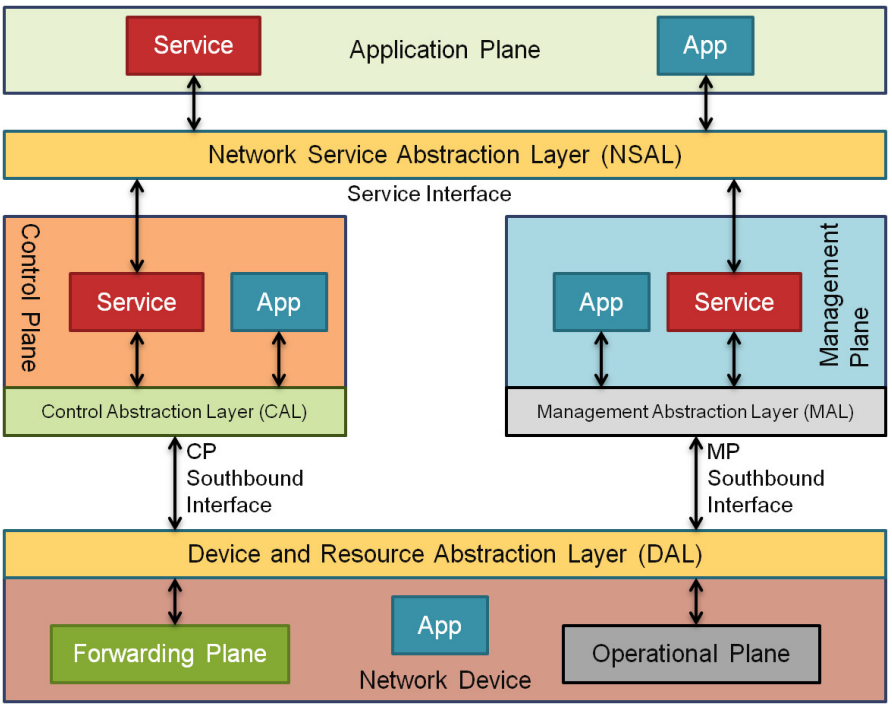

Figure 8: The SDN layered architecture [35]

The fundamental concept underlying SDN is that networks are composed of separate data and control planes, and so the division between these two planes needs to be specified by means of an open interface. The control plane is able to decide how well the network should operate, whereas the data plane is in charge of actually putting that behavior into action. In the simplistic form, the control plane specifies how the network should function, that is, the forwarding of the data streams. The challenge of transmitting packets along those paths is the job of the data plane. Network nodes make routing and forwarding decisions in a flow-based fashion, instead of destination-based as used in legacy traditional networks. For instance, one of the jobs of the control plane is to find the best path that packets should take through the network. No one disputes that separating the network control and data planes is beneficial. First and foremost, SDN's control and data plane interfaces must be well-defined and open. The technical name for such adaptability is segmentation, which enables separate entities to manage separate aspects of the system, such as particular planes. Therefore, disaggregation implies that a network operator ought to be able to afford to buy their control plane and their data plane from different vendors. As a natural result of the disaggregation process [20], inexpensive packet forwarding devices, often known as "bare-metal switches," are created from the data plane components, and all of the intelligence is handled as software and operating on the control plane. Once the control and data planes have been separated, the next step is to figure out the implementation of the control plane; every switch in the network must act independently and communicate with the other switches in the network to build a global routing table. The rate of innovation that SDN promises may not necessarily be realized thereby, despite the fact that doing so may cut costs by reducing complexity (due to the fact that just the necessary control units need to be put over onto the device). This is owing to the operator's continued participation in the slow standardization processes indicated by the protocols now in use. In addition to this, it is unable to supply the emerging networking representations that were anticipated by the developers of SDN. Another advantage of control plane standardization is one that will become more apparent as we go forward with implementing SDN use cases. APIs for the network may be accessed from a central site thanks to a control plane that is logically centralized. It has been there for previous architectural generations, but the concept of placing programmable APIs on particular switches and routers has not had much of an effect on the industry. Figure 9 shows the comparison between a traditional network and a SDN network.

A centralized API that is accessible across a full set of switches and routers, on the other hand, has made it possible to implement a wide variety of novel use cases. These include the virtualization of networks, the automation of networks, and the verification of networks. Although the objective of IT teams is to develop and administer IT infrastructures and applications, this work should ultimately fulfill critical business objectives for the organization they are a part of. Some of these factors are as follows:

- Expansion and Growth.
- Effectiveness.
- Expansion and Growth
- Flexibility and adaptability
- Security and reliability.

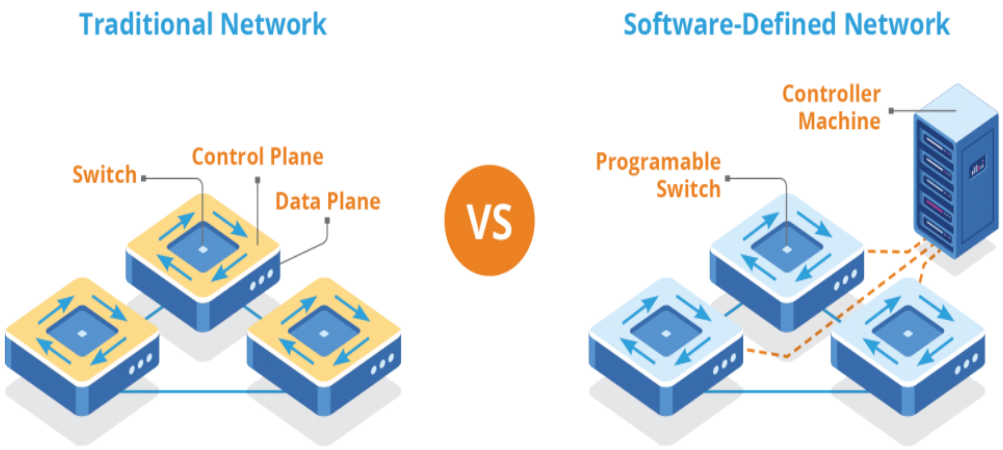

Figure 9: Traditional Network Vs SDN Network [21]

To satisfy the needs of current IT infrastructures, traditional DCN displayed significant shortcomings and posed several operational issues. These issues, as well as other additional needs produced by corporations from diverse sectors, orientated the approach toward SDN. A substantial gap was discovered between advances accessible inside the storage area and application servers and those relating to the field of telecommunication network infrastructures. It ranged from virtualization to automating and rapid migrations, to list a couple. The gap has indeed been filled by offering new networking functions (NF) and remedies to old and recurring difficulties. Clients have the minimal option in picking networking components devices such as hardware and software for the particular unique use-case. Earlier, network OS were closely linked to their underlying hardware and lacked widely recognized interfaces. This would be the situation when additional extracted features had already been directly connected with the whole network OS stack. With its flexibility, SDN offers choice and innovation at multiple tiers of the networking model. This is made feasible by the underlying allowing users to opt and then choose

the hardware they want to run on it. When it comes to networking, there are a lot of specifics that need to be revealed and absorbed by the average computer user. This tends to grow into specifics such as which Virtual Local Area Network (VLAN) each host should be connected to network node devices. Defining network requirements, which could then be configured programmability and redistributed by the control plane. Achieving agility, which refers to the capacity for automated and speedy network setup, is one of the most often requested features by the workgroups responsible for business applications in an effort to reduce their time to market in order to better their command and control over the underlying infrastructure. It takes a significant amount of time to carry out the (mostly) manual network configuration processes that are in use nowadays. This issue was resolved by the implementation of programmability networks and simpler intent-based interactions to specify the routing behavior of the network. Customization and distribution could now be accomplished in a matter of a few minutes rather than days, and much less human intervention was necessary for problem-solving after a modern orchestration layer was put in place. In the same way that users want the ability to handle their individual system infrastructures, they desire the ability to handle and manage their respective virtual networks, which were themselves implemented on a common physical infrastructure. The necessity for virtual routing protocols (VRFs) and multiprotocol border gateway protocol (MP-BGP) was eliminated thanks to SDN's implementation of virtualized networking services. A fundamental need for managed Internet Service Providers (ISPs) is the ability to share the same infrastructure with multiple users at the same time. In the instance of VLANs and Virtual private networks (VPNs), this was something that had previously been established. Because of overlay capabilities like Virtual Extensible LAN (VXLAN), SDN could also accommodate multi-tenancy in a comprehensive manner. It might be unable to keep up with the massive expansion in the size of the DC as the server port volume continues to expand at its previous rate. Included a restriction on the amount of MAC addresses that may be used, the inactivity of a large number of connections, the transportation of multicast packets, and so on. It was no longer able to expand the network infrastructure in an easy and meaningful manner in response to changing demands. It has become possible to easily add additional switches and provide its configuration in a matter of a few minutes, thanks to the use of "normalized" common switches and the issuing of network settings from centralized SDN controllers. In order to accommodate the rise in downlink throughputs, it became necessary to make effective use of each and every connection that was made accessible by the switches. It was previously common knowledge that the spanning-tree architecture, which is utilized extensively in local networks, would deactivate a portion of the connections. Multi-chassis EtherChannel (MEC) then subsequently Equal Cost Multi-Path (ECMP) with Charles_Clos architectures which is a type of non-blocking, multistage switching architecture that reduces the number of ports required in an interconnected fabric, were essential pathways to manage the many multi-pathing situations as a result of the spectacular development in server density. SDN was able to bring with it a number of important abstraction features, one of the most important being virtualization. Virtualization has been around for a long time, manifesting as server virtualization, VLANs, and overlay networks. As businesses sought more control over their networks, they began applying virtualization ideas to their DCs, and then to the WAN and LAN. As virtualized DCI became the standard, conventional VLAN ideas were updated to enable increasingly complex networks. As virtualization use cases extended throughout the network, enterprises eventually adopted SDN

principles in their data centers, WANs and LANs. This was made possible by allowing several separate virtual networks, which allowed servers to maintain up with the demands of computation and storage. The push toward virtualization that had previously occurred within the application server business was beginning to make its way into the networking market as well. SDN worked to expand on several levels and went through a lot of diverse iterations as it progressed. Software-defined data centers, also known as SDDCs, allow users to establish their own independent, unique DCs by combining the aforementioned application-driven infrastructure and services with orchestration and control-management layer. Without any need to acquire specialized hardware, common network equipment may be provided and independently structured in a virtual network by utilizing the right software. In most cases, the Infrastructure as a service (IaaS) providers are willing to take care of it; as a result, companies are also responsible for ensuring the devices' continued upkeep and security. In the event that particular equipment is no longer mandatory, the SDDC allows for their prompt removal at any moment. The most up-to-date types of SDDC take into account not just the influence of application decomposition into micro-services as well as the impending rise of docker machines (i.e. container-based deployments). SDN enabled programmable optical DCN architecture that includes the SDN-enabled control plane and the optical data plane, It allows several OS with different configurations to run concurrently on a physical machine. In the data plane, instead of using hardwired connections between nodes, a DCN architecture with an Architecture on Demand (AoD) node is used. This node consists of an optical backplane (i.e. a Polatis fiber switch with a large port number) to which various switching modules (i.e. OPS, Wavelength Selective Switch) and passive devices (i.e. Mux/DeMux, splitter). This AoD design allows for flexible I/O and module layouts to be built on-demand by establishing the necessary cross-connections in the optical backplane. It is possible to configure the backplane to connect to an OCS (optical circuit switch). An SOA-based modular architecture with widely distributed control enabling fast port-count-independent reconfiguration time has been devised and prototyped for the OPS [4]. A hybrid ToR switch may filter incoming server traffic and re-route it through alternative connection types (OPS/OCS). An major use of open interfaces exporting programmability is the control and administration of disaggregated optical networks. When it comes to modeling, however, optical networks present unique difficulties owing to the absence of agreed-upon hardware models, which are essential to the creation of an interoperable system based on disaggregated hardware. Having the standard and agreed-upon device and network models streamlines development and integration in regards to cross-vendor interoperability, a crucial necessity in a disaggregated network. Thus, ToR conducts traffic aggregation (e.g., traffic going to the same ToR/Rack) and application-aware traffic categorization to establish either OPS or OCS connection. the ToR switch's packet processing routines, When data begins flowing in, the SDN controller uses matching criteria stored in a look-up table (LUT) to sort it into various buffers. Once the sending mode (OPS or OCS) has been determined, the packets in each buffer will be sent on to the OPS/OCS mapping module, where additional associated connection setups, particularly for OPS connections, will be carried out (i.e., optical packet payload size and labels). An optical label is applied to the optical packet payload at the FPGA output for traffic that has been classed as OPS. This label is produced by a specialized opto-electronic label generator that is interfaced with the FPGA. OF agents are built for the AoD, OPS, and FPGA-based ToR in order to provide SDN control over this hybrid optical data plane. A variety of interfaces allow

the agents to communicate with the underlying devices and carry out tasks including gathering information about those devices' capabilities and qualities and setting them up for monitoring. Multiple Kubernetes technologies [22] are used throughout the process of container management and deployment inside clusters. As the administration of access and backbone optical networks grows in significance, research into the implementation of SDN is intensifying. Given the concentrated nature of optical networks, it makes even more sense to implement the centralized control envisioned by SDN.

1.4 Transport Layer Security / Secure Socket Layer (TLS/SSL)

The SDN Controller is a logically centralized entity in charge of translating needs from the SDN Application layer down to the SDN data paths and giving an abstract picture of the network to the SDN Applications (which may include statistics and events). The SDN Control Logic is composed of multiple SBI Agents. The definition of a conceptually centralized entity does not dictate or prevent implementation features such as the federation of numerous controllers, hierarchical controller connection, communication interfaces between controllers, or virtualization or slicing of network resources; each interface is implemented by a driver-agent pair, the agent representing the “southern” or infrastructure facing side and the driver representing the “northern” or application facing side, The southbound interface in SDN is the communication between the SDN controller at the middle control layer and the lower networking elements at the data layer. The data layer consists of physical or virtual network switches and ports. The SDN controller takes the desired state of the network and translates it into specific commands and configurations that are then pushed to the network devices over the southbound interface. The southbound communication must be conducted via an encrypted channel to prevent an unauthorized controller from taking control of the network or an unauthorized switch or network device from gaining access to sensitive network data, the southbound communication must be conducted through a secured channel to prevent an malicious controller from taking control of the network or an unauthorized network device from gaining access to sensitive data. SSL is the abbreviation for Secure Sockets Layer, an encryption protocol used to safeguard data while it is being sent over the internet. Netscape was first developed in 1995 [23] to protect users' privacy, authenticate users, and preserve the integrity of the information sent over the World Wide Web. SSL is the ancestor of TLS, a more modern and safe encryption system, In the browser's URL bar, the letters "Hypertext Transfer Protocol Secure (HTTPS)" will appear in place of the words "HTTP" if the website uses the SSL or TLS encryption protocol. By using both symmetric and asymmetric encryption, it ensures that the data's integrity, as well as the privacy of the data while it is in transit, is safeguarded. Using both protocols is essential to establishing secure connections between a client and a server. After the secure session has been created, symmetric encryption is used for the process of exchanging data with the recipient. Even though this helps to prevent and minimize security breaches, fraudsters continue to use these encrypted channels to spread malware and steal information because they know that they can get around typical security assessment methods that do not decrypt traffic data and use it to their advantage. Whenever it comes to detecting possible hazards buried within

traffic, security inspection technologies such as next-generation firewalls (NGFW), data-loss prevention (DLP) systems, intrusion detection and protection systems (IDS/IPS), web gateways, and others perform very effectively. Nevertheless, they do not perform accurate decoding of the transmission in order to save time before evaluating it. Over this, security evaluation tools are unable to identify encrypted dangers, and it is conceivable for harmful software or data to get through unnoticed or prevented, depending on the context. SSL Decryption, known as SSL Visibility, is the process of decrypting traffic data on a global level and forwarding it to other inspection tools to determine risks coming into apps as well as risks coming out of clients to the web, In Figure 10 shown the connection with HTTPS and with HTTP. The purpose of carrying out this procedure is to strengthen the level of security, an application example of TLS/SSL security gRPC as used in this work for southbound connection's security.

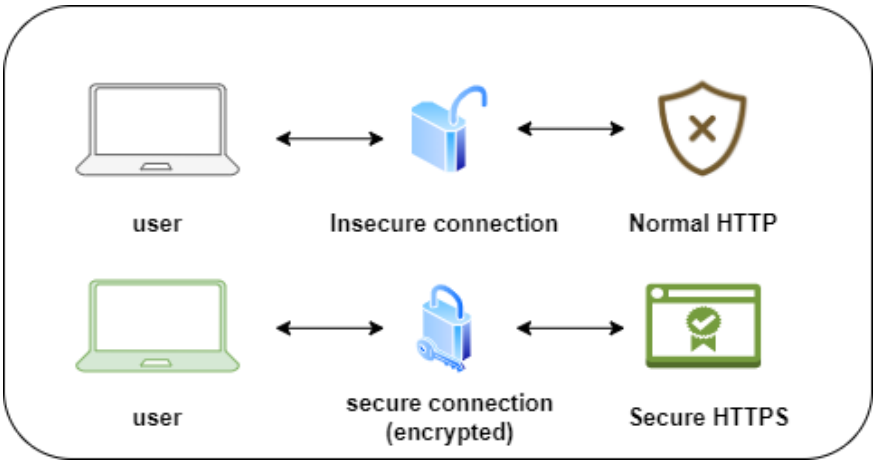

Figure 10: HTTP vs HTTPS

Encryption is performed before transmission using the SSL protocol through the web so that users may firmly believe that their information is kept private [24]. This ensures that anybody who attempts to acquire the data would only catch a jumbled-up collection of letters that is very difficult to decipher or decrypt. Among any two devices that are interacting with one another, SSL will start an authentication procedure known chiefly as a handshake in order to verify that specific devices are always, in fact, they say they are. In addition to this, data is digitally signed in terms of providing data integrity, which ensures that the data has not been altered since it is sent to the person or organization that it was meant for. There were several generations of SSL, each one making the protocol more secure than the previous one. Traditionally, information on the Web was sent in plaintext (unencrypted), meaning anybody might read it if they managed to capture the communication. SSL was developed to ensure the confidentiality of user information. Encrypting all sent data can assure that any third party that has access to the data information will only be able to view a scrambled mass of letters. Additionally, it prevents particular cyber-attacks: it verifies web servers, which would be vital since cyber-criminals will frequently attempt to build up counterfeit sites to fool users and steal sensitive data [25].

Additionally, it prohibits attackers from altering data while it is in transmission, which is another way they attempt to take the information. In a typical implementation of TLS as shown in Figure 11, the server is the one who has a TLS certificate as well as a public-private key set, whereas the client is the one who does not. The TLS protocol normally operates in the following manner:

1) Client establishes connection with server.
2) The server displays the TLS certificate it was issued with.
3) The client does a certificate validation check on the server.
4) The client and the server communicate against each other over an encrypted TLS connection.

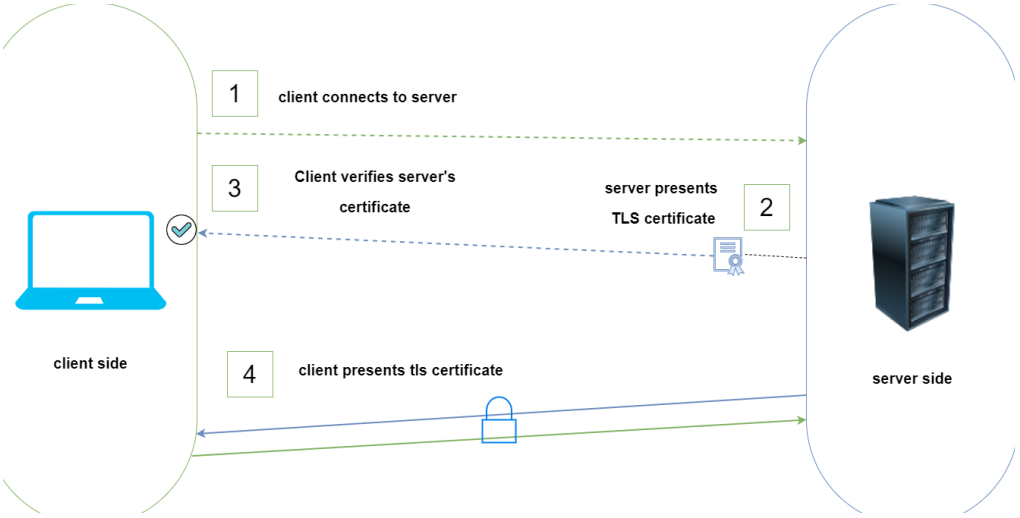

Figure 11: Implementation of TLS

1.4.1 Mutual TLS (mTLS)

Whenever two parties of a communication system check each other's identification, this process is referred to as mutual authentication. This would be in contrast to traditional authentication, where only one part verifies the other. Since the procedure is carried out in all the ways, mutual authentication is often referred to as "2-way-authentication. [26]" Hence, mutual authentication validates both participants in a digitized communication system. For instance, when employing mutual authentication, a user client and a server take appropriate measures to independently verify and confirm each other's identities, rather than only the client certifying the server. This is in contrast to traditional authentication, in which only the client verifies the server's credentials. Connections from one device to another, such as those between IoT nodes, frequently use mutual authentication as well. However, this practice may also be used by other protocols and in various settings. There are several techniques for mutually authenticating the endpoints of a communications channel, which will be explained by the following:

1) Authentication using a public key: This technique depends on public-key encryption. A secret key is used in encrypting data, which results in the data becoming unreadable. In order to encrypt data using a public key, not only one key but two are required: a private key and a public key. It is possible to decode data that was encrypted using the public key by using the private key, and it is also possible to decrypt data that was encrypted using the private key by using the public key. In the process of public critical mutual authentication, both parties involved in communication broadcast a public key, and both parties are required to demonstrate that they own the private key that corresponds to their public key. This is analogous to someone displaying an identity (ID) card such that their name can be verified. Each party transmits to the other a fragment of data known as a digital signature, which is then encrypted using their respective private key before being sent. The receiving party uses the public key to decode the signature. If the signature can be successfully decrypted using the public key, then the correct private key was used to encrypt it, and the entity that supplied the signature can be trusted.

2) Authentication through Certificate: This method is comparable to authentication by public key; however, rather than merely a public key, both counterparties get a public key certificate in their possession. Whoever granted the certificate and public key, to whom the certificate relates, whenever the certificate terminates, that may be used to verify the identities of those engaged are all contained in the certificate, as well as other relevant data., even when both parties possess a TLS certificate, then that is possible to utilize that digital certificate with this form of mutual authentication.

3) Username and password authentication: This mutual authentication technique still utilizes a digital certificate again on the server-side. The server gives the client a certificate, and it is the client's responsibility to verify the digital certificate. Standard username and password authentication are used on the client-side of things. This means the client will transmit its login and password pair to the server, verifying the credentials.

On the Web, authentication can only go in one direction. This is called one-way authentication. When a user accesses a website, if a certain website utilizes HTTPS, the user's device will verify the identity of the web application server by validating the TLS certificate, which is stored on the web server. For example, someone would be logging into their account on a mobile application; in this situation, the mobile application is the one authenticating the user. The transmission of data takes longer and requires more processing power due to the need for mutual authentication, which, on the other hand, removes some security weaknesses and makes it significantly harder to move out certain kinds of threats. Additionally, it demands preparation since both parties involved in the communication channel need to own a credential, such as a public-private key pair or a public key certificate. Because of this, it is challenging for the ordinary user to perform

mutual authentication, which is one of the reasons why mutual authentication is often not a component of TLS when someone is utilizing a web app. The vast majority of IoT devices require a connection to a distant location for conducting their intended tasks. They may need to link to other IoT devices as well. IoT devices ought to communicate with one another through an insecure network. Mutual authentication allows verifying that the data they obtain is correct and coming from a valid source, thus minimizing the likelihood that an adversary has hacked their connections somehow. However, API security depends on authentication by using TLS/SSL to secure the gRPC communication channel between the SDN controller and the agents at the data plane, verifying that APIs' queries originate from reliable sources. One technique to ensure that an API is not allowing queries from cyberattacks, whereas an API user is not receiving faked API answers, is to implement a system known as mutual authentication between the two parties. Also, security based on the principle of zero-trust believes that any individual or device may be a potential danger. Mutual authentication guarantees that only genuine users are interconnected to the network, server, and web application by requesting authentication from both parties of a connection at the same time. On the other hand, users can know that they are connecting to the appropriate network, web application, or server. However, with mTLS, as seen below in Figure 12, both the client and the server have a certificate, and both the client and the server authenticate one another using their respective public and private key pairs. When compared to traditional TLS, mTLS includes a few more stages for verifying :

1) Client establishes connection with server
2) The server delivers the certificate it uses for TLS.
3) The client performs a certificate check on the server.
4) Client delivers its TLS certificate
5) The server makes sure the client's certificate is valid.
6) Access is granted by the server.
7) Information is traded between the client and the server through an encrypted TLS connection.

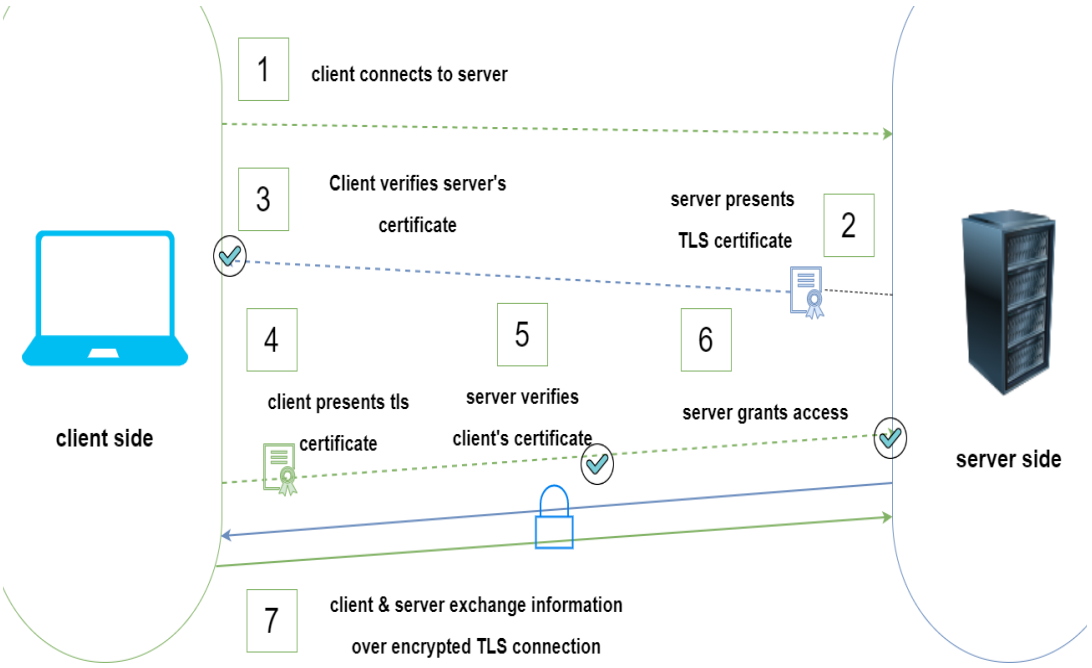

Figure 12: Implementation of mTLS

2 Secure Monitoring and Management Framework

The mismatch between the variety of resources needed by running applications and the set quantity of hardwired resources (e.g., number of central processing unit CPU cores, memory capacity) in server blades greatly limits resource usage in contemporary DCs. The paradigm of function disaggregation is presented in this context, in which integrated server blades having all sorts of resources are replaced with resource blades carrying just one specialized function. Each rack-scale DDC architecture has completely separated hardware components for the CPU, memory, storage, NIC, and accelerator. Optical connections in a rack allow for communication between blades that each contain just one kind of resource, as opposed to the more general server blades that hold a wide variety of resources. These architectures may be classified as either an all-optical or a hybrid interconnect depending on the kind of interconnects utilized in the rack. The thesis project's primary emphasis is on the development of a monitoring framework for optical DDC in a secured way that would allow the establishment of low latency and high-speed optical interconnections between hardware component nodes in a DDC infrastructure with optical DCN in responding to the requirements of upper layer services and applications. In particular, the work intends to accomplish the high-security characteristics of the monitoring framework by including a trustworthy layer and adopting the usage of TLS encryption to encrypt communications between the client-SBI, the server-SBI, and the server-controller-SBI. This will be done in order to protect sensitive data. Users often mention multi-vendor compatibility for provisioning as an appealing feature of SDN; however, that is not always true even though there is an assumption of benchmarks behavior. When trying to build marketplaces for their wares, vendors may resort to using proprietary methods and protocols, the

success of which is contingent either on the pervasiveness of the vendors' goods or the allure of the applications they power. In the SDN market, applications were initially intimately connected to the controller due to the apparent usage of non-standardized APIs, and this keeps happening nowadays. Due to the apparent relationship between the agent and the controller, then the agents adapt the tenancy scheme into a format that is easily consumable by non-controlled components. Typically, this entails changing the tenancy scheme into Subnetworks. This technique could have a reduced capacity for handling packets, resulting in a throughput hit. In order to exert control over commercially available devices, which do not often come pre-fitted with SDN capability out of the box, the ad-hoc method was used to develop the Agents. The source code for the extended SDN Agent has been released and is available to be tested. To be more specific, this means that each and every SDN Agent makes it easier to configure, maintain, and monitor functions in order to provide the fundamental characteristics that are required by the optical DDC architecture. Concentrating on the particular APIs that are exposed by the optical network devices makes it possible to activate these functions. Thus, the project will center on the delivery of a secure monitoring framework for optical DDC to enable the establishment of high-speed and low-latency optical connections between hardware components in the DC in response to the needs of higher-level services and applications, as well as the implementation of secure monitoring that relies on the SDN agents. During the course of continuing communication, several potential security risks, including those to privacy, confidentiality, integrity, and availability, may emerge. Consequently, the communication channel must be secured by the using of SSL/TLS technologies against eavesdropping and tampering should the messages being sent be intercepted and used maliciously.

At the level of the orchestrator; the service are modelled and represented, then converted into different data models conveying the specificities of each managment element, such as the SDN controller, which deals with connecivity services. It includes all of the specifications that the client has specified for the managed services. In addition, in order to compile aggregated monitoring information, the Monitoring Abstraction Manager analyzes the information of the infrastructure devices, Figure 13 depicts the architectural design including the agents, one per blade, as a part of the full architecture of intent-based SDN control with DDC infrastructure orchestration, which is obtained from the monitoring manager, the SBI with the gRPC protocol does not only communicates with the NOS nodes, but also to any infrastructure element that has a running SDN agent, like the CPU, memory and NIC blades. In particular, the DDC Monitoring module compiles the monitoring data that is connected to the assets that are utilized. This data is then used to continue providing high-level performance information of the DDC infrastructure, which in turn enables periodic maintenance to be taken by the orchestrator. After that, the NICs are linked to one another via the composed DC inter-connection network (CDC-N), as well as the switch components that form such a network ought to install SDN agents for the controller in order to set the pathways between the CCNs.

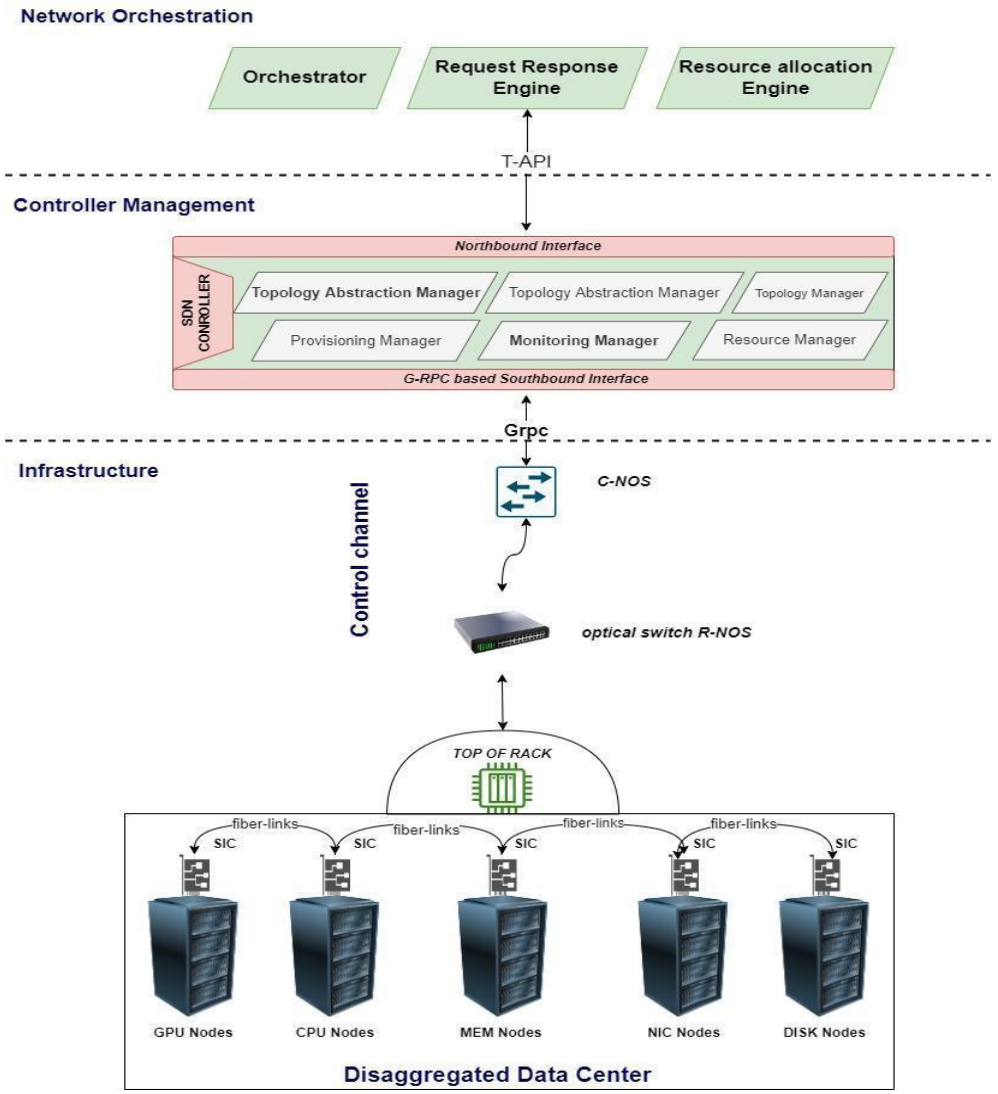

Figure 13: Architecture for intent-based SDN control and orchestration of DDC infrastructure

In order to isolate the control and orchestration from the framework of the infrastructure facilities, the linkage that is obviously necessary to the physical DDC fabric, always present in SDN-based architectures is the need for agents and the channels via which they communicate with the SDN controller and the underlying infrastructure. This layer consists of the SDN agents, the SBI as well as the Resource Manager of the SDN controller. This was done with the intention of decoupling the control and orchestration from the innovation of the physical infrastructure. Furthermore, the SDN controller includes an implementation of the Topology Abstraction Manager. This manager analyses the information gathered by the Resource Manager via the SBI in order to compute the DDCs that are associated with the client services that have been requested. In this part, we will provide the following information that was provided up until this point, the main objectives of the thesis project are to develop, design, and implement

the control layer of a DC infrastructure, which is premised on the advanced technologies of disaggregation and makes use of optical technological developments frameworks for the DC network fabric infrastructure. These objectives will be accomplished by utilizing the information that has been presented prior to this point. The use of the technology that allows for the disaggregation of data will be used to attain this goal. As was said earlier, the control layer is based on the fundamentals of SDN. This layer, which conforms to the tendencies that are now dominant in the network industry, is built on top of the underlying infrastructure.

2.1 SDN agents implementation

Interconnected data like metrics, tracking, and notifications constitute the backbone of every monitoring infrastructure. They could show how their systems are doing, provide light on patterns of use or behavior, and clarify the results of any adjustments. These systems may provide alerts to an operator if the metrics deviate from the predetermined norms, and the operator can then utilize the information that has been brought to the surface to investigate the root cause. Metrics are the raw, observable, and collectible measures of resource utilization or behavior across all of your systems. These may be high-level summaries of use connected to the particular functionality or job of a component, such as the number of requests processed per second or the number of web servers in a cluster, or they can be low-level usage summaries supplied by the OS. Metrics may be provided in two ways: either in proportion to a total capacity, or as a rate that represents the "business" of a component. The metrics your OS currently exposes to you to indicate the use of underlying physical resources are often the best place to start. Disk consumption, CPU utilization, NIC utilization, RAM utilization, swap usage, and other similar metrics are all readily accessible and may be related to a monitoring system with no effort. Furthermore, many software packages, including web servers, database servers, and others, also have their own metrics that may be sent along. Metrics collection and dissemination serve the function of monitoring, when studied together, metrics provide important information about the activity and health of the systems. These are the primary input into your monitoring system, from which an accurate picture of the state of the environment can be gleaned, automated actions can be triggered, and human operators can be notified of any necessary modifications. Metrics are the fundamental values used to analyze patterns across time, establish relationships between seemingly unrelated variables, and track improvements or declines in output, consumption, or mistakes.

The metrics in your system are the numbers that may be collected, aggregated, and analyzed via monitoring to get a deeper understanding of the features and behavior of the individual parts of your system. A monitoring system is what stores, aggregates, visualizes, and triggers automatic reactions when values satisfy certain criteria, and it collects data from all around the environment. The distinction between metrics and monitoring is similar to that between data and information. In contrast to data, which consists of unprocessed facts, information is the product of data analysis and organization to provide context that adds value. Data from metrics is aggregated and displayed in different ways to help people draw conclusions. Various interconnected tasks are performed by monitoring systems. First and foremost, they have to be able to take in and retain information that has already been collected. While values at any one moment in time have their place, it is often more informative to see them in the context of previous values in order to better understand how things have changed and what the overall trends are. This necessitates the capability of a monitoring system to handle data in a longitudinal fashion, which may need the sampling or aggregation of historical data. Secondly, data visualizations are frequently available in monitoring systems. Metrics may be presented and comprehended in the form of individual numbers or tables, but humans are considerably more adept at seeing patterns and making sense of how parts relate to one another when the data is presented in a visually compelling fashion. Configurable graphs and dashboards are common ways in which monitoring systems visually depict the parts they track. This allows for a glance at a screen to convey an understanding of the interplay between several complicated factors or changes inside a system.

Data from several sources may be consolidated and correlated by using a monitoring system. When analyzing the metrics, managers must be able to spot similarities and differences across servers and types of resources. Whether an application encounters an increase in error rates, for instance, the administrator should be able to utilize the monitoring system to determine if this corresponds with the saturation of a linked resource, monitoring systems are often utilized as a foundation for creating and deploying alerts. The following SDN-Agents were built at this time in Python. These SDN-Agents are tailored to the various blade types that we need to keep an eye on and tweak, and they provide us with access to data for the relevant metrics.

2.1.1 CPU-Agent

In the CPU-Agent code as shown below in code block 1, we implemented CPUMonitor class to monitor and calculate the load of the device by using of psutil library and their useful functions to calculate the CPU Load by utilizing /proc/stat' to display the CPU utilization, After that, we may save the values we've obtained in a csv. We could bring them up later in and deduct them from the latest statistics in the next iteration.

```
class CPUMonitor(Monitor):
    def __init__(self, device):
        Monitor.__init__(self, device)

    def run(self):
        while self.stop is False:
            self.device.load = psutil.cpu_percent(interval=1, percpu=True)[0]
            print("[cpu_load] {}".format(self.device.load))
            time.sleep(self.device.monitoring_freq)

    def run_old(self):
        last_idle = last_total = 0
        while self.stop is False:
            with open('/proc/stat') as f:
                lines = f.readlines()
                #fields = [float(column) for column in f.readline().strip().split()[1:]]
                fields = [float(column) for column in lines[self.device.local_cpu_id].strip().split()[1:]]
            idle, total = fields[3], sum(fields)
            idle_delta, total_delta = idle - last_idle, total - last_total
            last_idle, last_total = idle, total
            self.device.load = 100.0 * (1.0 - idle_delta / total_delta)
            print("[cpu_load] {}".format(self.device.load))
            time.sleep(self.device.monitoring_freq)
```

Code block 1: CPU-Agent

2.1.2 Memory-Agent

In the Memory-Agent code, as shown below in code block 2, we implemented the memMonitor class to monitor and to get the RAM information summary statistics of the used percentage of ram and the available percentage of ram by using the psutil library and their useful functions. Thanks to that, it is possible to get real-time RAM and CPU monitoring data and to tally up the amount of RAM currently being utilized and to determine the amount of accessible memory as well.

```
class memMonitor(Monitor):
   def __init__(self, device):
      Monitor.__init__(self, device)

   def run(self):

      vvm=psutil.virtual_memory()
      x=dict(psutil.virtual_memory()._asdict())

      while self.stop is False:
         print ('                                                ')
         print ('--------------RAM Information summary--------------')
         print ('                                                ')
         def forloop():
            for i in x: # i == each parameter in memory data
               print (i,"--",x[i]/1024/1024/1024) #Output will be printed in GBs
         forloop()
         print ('                                                ')
         print ('--------------RAM Utilization summary--------------')
         print ('                                              ')
         print('Percentage of used RAM :',psutil.virtual_memory().percent,'%')
         print('Percentage of available RAM :',psutil.virtual_memory().available * 100 /
psutil.virtual_memory().total,'%')
         self.device.usage = psutil.virtual_memory().percent
```

Code block 2: MEM-Agent

2.1.3 NIC-Agent

In the NIC-Agent code, as shown below in code block 3, we implemented NICMonitor class to monitor and get the useful information statistics of NIC by using the psutil library and their useful functions, As a consequence, this code retrieves the statistics and data linked to the NIC, such as throughput, the number of bytes and packets transmitted and received, the overall number of failures when receiving and transmitting, and the overall number of inbound/outward packets that have been lost.

```
class NICMonitor(Monitor):
    def __init__(self, device):
        Monitor.__init__(self, device)

    def run(self):
        while self.stop is False:

            if len(sys.argv) > 1:
                INTERFACE = sys.argv[1]
            else:
                INTERFACE = 'eth0'

            last_received = psutil.net_io_counters().bytes_recv
            last_sent = psutil.net_io_counters().bytes_sent
            last_total = last_received + last_sent

            while True:
                bytes_received = psutil.net_io_counters().bytes_recv
                bytes_sent = psutil.net_io_counters().bytes_sent
                bytes_total = bytes_received + bytes_sent

                new_received = bytes_received - last_received
                new_sent = bytes_sent - last_sent
                new_total = bytes_total - last_total

                mb_new_received = new_received / 1024 / 1024
                mb_new_sent = new_sent / 1024 / 1021
                mb_new_total = new_total / 1024 / 1024

                RX_RATE = mb_new_received
                TX_RATE = mb_new_sent
                TOTAL_RATE = mb_new_total
                print(f"{RX_RATE:.2f} MB received, {TX_RATE:.2f} MB sent, {TOTAL_RATE:.2f} MB total")

                last_received = bytes_received
                last_sent = bytes_sent
                last_total = bytes_total
                time.sleep(1)


                    #final result

                self.device.rate_rx = RX_RATE
                self.device.rate_tx = TX_RATE
                self.device.total_rate = TOTAL_RATE
                time.sleep(self.device.monitoring_freq)
```

Code block 3: NIC-Agent

2.2 Secure gRPC With TLS/SSL

The SBI is the interface between the SDN controller and the infrastructure devices. Its primary function is to allow the communication between the controller and the infrastructure devices, with the purpose of configuring them according to upper service requirements as well as gathering metrics and statistics about their status and current utilization as part of the QA loops.. In practice, most current APIs adhere to the same HTTP standard and are implemented by mapping different models to that standard. There are now more options available to an API designer as a result of the trend toward RPC API designs that include one or two principles from the HTTP model while yet remaining inside the RPC framework. gRPC is a contemporary Remote Procedure Call (RPC) framework [33] that is open source and high performance, it is an implementation of RPC APIs that relies on the HTTP 2.0 protocol. Since they are built on totally different conceptual frameworks, it stands to reason that gRPC and HTTP can't coexist. To do this, gRPC adopts the Remote Procedure Call (RPC) architecture, where procedures serve as the addressable entities and data is buried behind them. When using HTTP, the process is inverted. The behaviors are concealed behind the data; the creation, modification, and deletion of resources determine the system's behavior. It has pluggable functionality for load balancing, tracing, health monitoring, and authentication, which enables it to link services in and across data centers in an effective manner. It may also be used in the last phase of distributed computing, known as the "last mile," to link browsers, mobile apps, and devices to backend services. The implementation of the "minimum security demands" that is identified is one of the most crucial preconditions for the project thesis. These products aren't built with high-security usage in mind, but they nevertheless have to meet certain basic standards for authentication and encryption (cryptography). For instance, it makes perfect sense that a web application that keeps sensitive information such as passwords or credit card numbers would use a salted hash to encrypt the data saved to disk on a record-by-record level. A distributed system must also have the capacity to manage encrypted blobs, encrypt all inter-node communication, and authenticate and sign all communications. The implementation of this measure will result in an increase in the system's overhead costs; however, these costs will be significantly less than the costs associated with a security breach. Furthermore, if the system's minimum security requirements are met, then the overhead costs will be regarded as acceptable business costs. Because it was developed to be compatible with a wide range of authentication schemes, gRPC makes it simple and straightforward to communicate securely with other computer systems. You may simply utilize the endorsed techniques such as SSL/TLS both with and without Google token authentication, and you can also plug in your own authentication system by modifying our given code. Both of these options are simple to implement. gRPC's built-in support for authentication and its encouragement of the usage of SSL/TLS for authenticating and encrypting communications are two of its many attractive features. The initial stage ought to encrypt the connection using TLS/SSL from the server, succeeded through an authenticated, collaborative TLS utilizing a certificate authority; this became essential since all replicas in the system served as between servers and clients. Chains of trust, also known as transitive trust, are essential to the operation of TLS and SSL. If a certain certificate authority has earned my or an application's confidence, then will also trust the certificates that it has issued. This trust is assumed by

browsers and OS. Almost any browser and ant OS really does have a Trustworthy Roots certificate repository, which it uses to verify the trustworthiness of HTTPS servers on the internet. Because whenever you register an SSL/TLS server certificate from other public certificate authorities, web browsers, and OS will completely trust the truthfulness of that server certificate. We're developing our own certificate authority (CA) so clients could trust our server's certificate. Regarding server certificates, we must ensure that the server name we connect to matches the server name in the certificate. Client and Server have faith in the identity of the certificate authority, therefore, in one another. During the TLS handshake that takes place between the client and the server, connection to a server can only be formed after a client has successfully connected, then the client presents the server with its own identity certificate. The client validates the certificate presented by the server by ensuring that it was created and signed with the help of the certificate authority. The server, on the other hand, does the identical action and verifies that the client is providing a certificate that has been signed and produced by the trusted ca. The sections that follow provide examples more about how the authentication and authorization mechanisms mentioned previously appear in the python programming language. As a result, the server has established its reliability, but the client has not. Fortunately, TLS enables both parties to be verified. By confirming that both parties involved in a network connection possess the appropriate private key, mutual TLS (mTLS) guarantees that the entities at either end of the connection are who they guarantee to be. Further verification is provided by the metadata included inside each of their unique TLS certificates. In a Zero Trust security model, mTLS is often used as a verification method for users, devices, and servers located inside an enterprise. Additionally, it may contribute to the safety of APIs. It helps to guarantee that traffic among a client and a server is reliable and trustworthy in both ways. Users that log in to an organization's network or apps benefit from this extra layer of protection that is provided by this. In addition to this, it validates connections with client devices, such as Internet of Things (IoT) gadgets, that do not need a login procedure.

2.2.1 Securing Server/Client Using SSL/TLS Authentication :

At this part, we implemented the authentication for the server (both agent_api_server and controller_sbi_server) parts by importing the webserver certificate and the webserver key to the server code as shown below in the code blocks 4, 5, then implemented the authentication for the client (both agent_sbi_client and controller_sbi_client) parts by importing the certificate authority (CA) as shown below in code blocks 6, 7:

- agent_api_Server

```
def __init__(self, agent):
    '''
    Constructor
    '''
    self.agent = agent
    self.server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
```

```
        with open('rootca/webserver.key.pem', 'rb') as f:
            private_key = f.read()
        with open('rootca/webserver.crt.pem', 'rb') as f:
            certificate_chain = f.read()
        self.server_credentials = grpc.ssl_server_credentials(((private_key, certificate_chain), ))

    def run(self):
        sbigrpc.add_AgentServiceServicer_to_server(self, self.server)
        self.server.add_secure_port('[::]:50052' , self.server_credentials)
        self.server.start()
```

Code block 4: agent_api_Server

- controller_sbi_server

```
class ControllerSbiServer(sbigrpc.ControllerServiceServicer):

    def __init__(self, parent):
        '''
        Constructor
        '''
        self.parent = parent
        self.server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))

        with open('rootca/webserver.key.pem', 'rb') as f:
            private_key = f.read()
        with open('rootca/webserver.crt.pem', 'rb') as f:
            certificate_chain = f.read()
        self.server_credentials = grpc.ssl_server_credentials(((private_key, certificate_chain), ))

    def SendHello(self, request, context):
        print("Received Hello Request ")

        hello_msg = sbi.HelloReply(status='OK')
        print("Sending Hello Reply")

        return hello_msg

    def run(self):
        sbigrpc.add_ControllerServiceServicer_to_server(self,self.server)
        self.server.add_secure_port('[::]:50051', self.server_credentials)
        self.server.start()
```

Code block 5: controller_sbi_server

- agent_sbi_client

```
class SbiClient(object):
    def __init__(self,agent):

        self.agent = agent
        #url = 'localhost:50051'
        self.url = self.agent.controller_ip + ':' + str(self.agent.controller_port)
        with open('rootca/cacert.pem', 'rb') as f:
            self.creds = grpc.ssl_channel_credentials(f.read())

    def sendMessage(self):
        ip_address = self.agent.agent_ip
        port = str(self.agent.agent_port)
        print("Agent = {}:{}".format(ip_address,port))
        print("Controller = {}".format(self.url))

        with grpc.secure_channel(self.url, self.creds) as channel:
            stub = controllergrpc.ControllerServiceStub(channel )


hello_request=controller.HelloRequest(id=self.agent.device.id,type=self.agent.device_type,ipaddress=ip_address,
port=port)
            response = stub.SendHello(hello_request)
        print("Server says: " + response.status)
```

Code block 6: agent_sbi_client

- Controller_sbi_client

```
class SbiClient(object):

    def __init__(self):
        '''
        Constructor
        '''
        with open('rootca/cacert.pem', 'rb') as f:
            self.creds = grpc.ssl_channel_credentials(f.read())

    def sendMessage(self):
        print("Choose a specific message: GetFeatures, GetStatistics, ConfigureFlow")

    def sendGetFeaturesMessage(self,device_id):
        with grpc.secure_channel('10.0.2.9:50052',self.creds) as channel:
            stub = agentgrpc.AgentServiceStub(channel)
            response = stub.GetFeatures(agent.FeaturesRequest(id=device_id))
        print("Received response from: " + response.type)
        return response
```

```
    def sendGetStatisticsMessage(self,device_id):
        with grpc.secure_channel('10.0.2.9:50052', self.creds) as channel:
            stub = agentgrpc.AgentServiceStub(channel )
            response = stub.GetStatistics(agent.StatisticsRequest(id=device_id))
        print("Received response from: " + response.id)
        return response

    def sendConfigureFlowMessage(self,device_id,device_type):
        with grpc.secure_channel('10.0.2.9:50052',self.creds) as channel:
            stub = agentgrpc.AgentServiceStub(channel )
            if device_type == "NIC":
                response = stub.ConfigureFlow(agent.ConfigurationRequest(id=device_id,
nicBody=agent.NICConfiguration(flowInfo=agent.OPSFlowConfiguration(label="1",wavelength="1550",load="
0.5"))))
            elif device_type == "NOS" or device_type == "RNOS" or device_type == "CNOS":
                response = stub.ConfigureFlow(agent.ConfigurationRequest(id=device_id,
nosBody=agent.NOSConfiguration(in_port="in1",out_port="out1",flowInfo=agent.OPSFlowConfiguration(label
="1",wavelength="1550",load="0.5"))))
        print("Received response from: " + response.id)
        return response
```

Code block 7: controller_sbi_client

- Generating the certificates with gencert.sh file as shown below in code block 8 and apply the mutual TLS:

  The server has indeed provided the client with its certificate. Mutual TLS requires both the server and client exchanging certificates. The gencert.sh shell-script has been updated to produce and sign a certificate for the client. In its most basic form, this shell-script consists of the following five parts:

  - ❖ Firstly, establish the private key for the Certificate Authority (CA) and the certificate that it has signed for itself.
  - ❖ Secondly, generate a private key for the web server as well as a certificate signing request (CSR).
  - ❖ Thirdly, make use of such a CA's private key in order to sign the web server's CSR and then get the certificate.
  - ❖ Fourthly, produce the client's private key as well as the CSR.
  - ❖ Finally, to receive the signed certificate then sign the client's CSR using the CA's private key and return it.

```
rm *.pem


# 1st.
openssl req -x509 -newkey rsa:4096 -days 365 -nodes -keyout ca-key.pem -out ca-cert.pem -subj
"/C=ES/ST=Barcelona/L=Barcelona/O=UPC/OU=TFM/CN=THESIS/emailAddress=moradsmadi1997@g
mail.com"

echo "CA's self-signed certificate"
openssl x509 -in ca-cert.pem -noout -text

# 2nd.
openssl req -newkey rsa:4096 -nodes -keyout server-key.pem -out server-req.pem -subj
"/C=ES/ST=Barcelona/L=Barcelona/O=upc/OU=tfm/CN=THESIS/emailAddress=moradsmadi1997@gmai
l.com"

# 3rd.
openssl x509 -req -in server-req.pem -days 60 -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -out
server-cert.pem

echo "Server's signed certificate"
openssl x509 -in server-cert.pem -noout -text

# 4th.
openssl req -newkey rsa:4096 -nodes -keyout client-key.pem -out client-req.pem -subj
"/C=ES/ST=Barcelona/L=Barcelona/O=upc/OU=tfm/CN=THESIS/emailAddress=moradsmadi1997@gmai
l.com"
# 5th.
openssl x509 -req -in client-req.pem -days 60 -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -out
client-cert.pem

echo "Client's signed certificate"
openssl x509 -in client-cert.pem -noout -text
```

Code block 8: gencert.sh to generate the certificates

3 Results And Test

In order to showcase the operation of the developed SDN agents and how they are able to gather monitoring data from the infrastructure elements in a timely and secure fashion, this section details a series of experimental results highlighting these aspects.. These results may be thought of as the end product of the previous section. These discoveries came about as a direct consequence of the work that was discussed in the introduction. When you have the psutil package implemented in the Python environment, you will be able to collect statistics on how your system machine's resources, such as its CPU, RAM, and NIC, are being used, which will allow you to make informed decisions about how to best allocate your resources. Because of this, you are now able to easily combine into our own Python program code the capacity of studying and monitoring the utilization of the several computational blades at the physical infrastructure. The tests allow for the validation and verification that the metrics from the actual hardware can be monitored using the developed agents, while also providing assurance that the communication channel must be secured against eavesdropping and tampering.

3.1 Testing CPU Statistics

Figure 14 depicts the steps involved in obtaining CPU statistics and data information after constructing the CPU_Agent, which includes receiving agent-related information such as the agent's and controller's IP addresses and sending messages to the controller that include these metrics along with information about the CPU_LOAD and CPU_USAGE. There's also a graphical depiction of CPU utilization, along with the percentage of CPU use as a function of time (in milliseconds), the CPU utilization is retrieved from an idle state.

```
┌──(ddc-control-plane-xDnjB-X-)─(root㉿ca)-[/home/…/Desktop/TFM GIT PROJECT/ddc-control-plane/src]
└─# python -m unittest test.test_cpu_agent

Creating Agent
Agent - 10.0.2.9:50052
Controller - 10.0.2.9:50051
Received Hello Request
Sending Hello Reply
Server says: OK
[cpu_load] 4.0
.Sending Get Features Message
Received Get Features
Received response from: CPU
[cpu_load] 2.1
.Sending Get Statistics Message
Received Get Statistics
[get_usage] 2.1
Received response from: CPU-1
.
----------------------------------------------------------------------
Ran 3 tests in 8.089s

OK
```

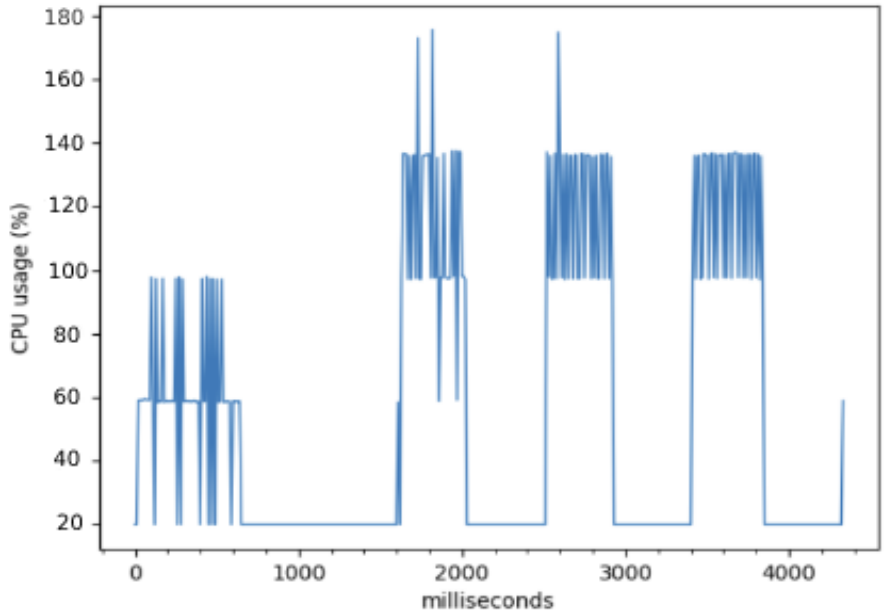

Figure 14: CPU Statistics

3.2 Testing Memory Statistics

As can be seen in Figure 15, after the MEM_Agent has been created, statistics and data information pertaining to the Memory_RAM are obtained. Then, the agent receives information about itself, such as the agent's and the controller's IP addresses. Finally, the agent sends messages to the controller that include a summary of the RAM's information and its utilization, the amount of memory used in megabytes and the amount of time it took to use that much memory (in seconds):

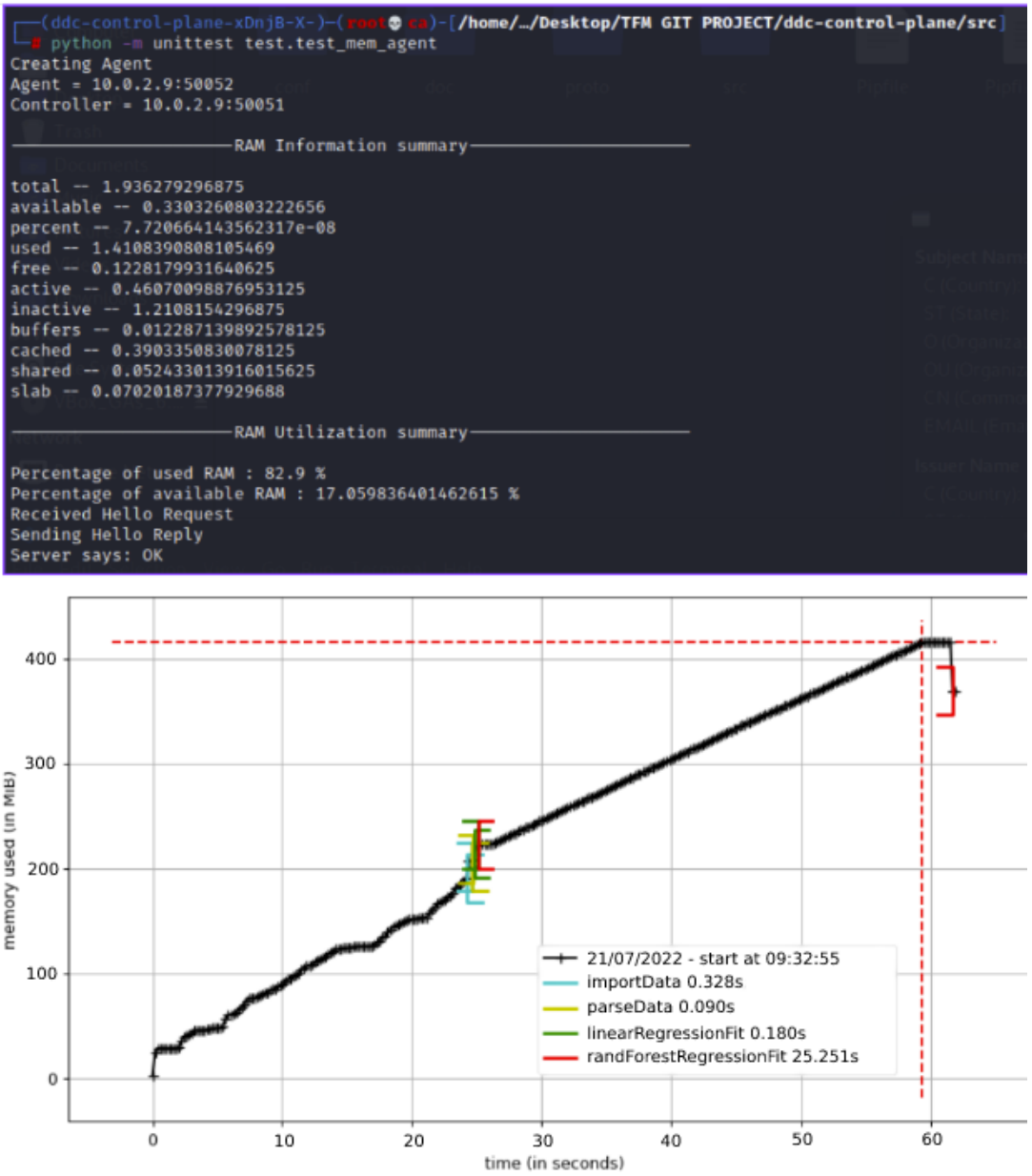

Figure 15: Memory Statistics

3.3 Testing NIC Statistics

As can be seen in Figure 16, running the test_nic_agent python file to retrieve the statistics and the data information of the NIC after creating the NIC_Agent, then sending the configuration flow message, then receiving the information related to an agent such as the IP addresses of the agent and the controller, as well as receiving and sending the messages to the controller also with the statistics of the NIC such as the throughput, the number of bytes and packets transmitted and received as seen in the chart below, the overall number of failures when receiving and transmitting, and the overall number of inbound/outward packets that have been lost, also showing a graphical representation for the NIC usage (Bytes sent and received), the NIC utilization is retrieved from an idle state.

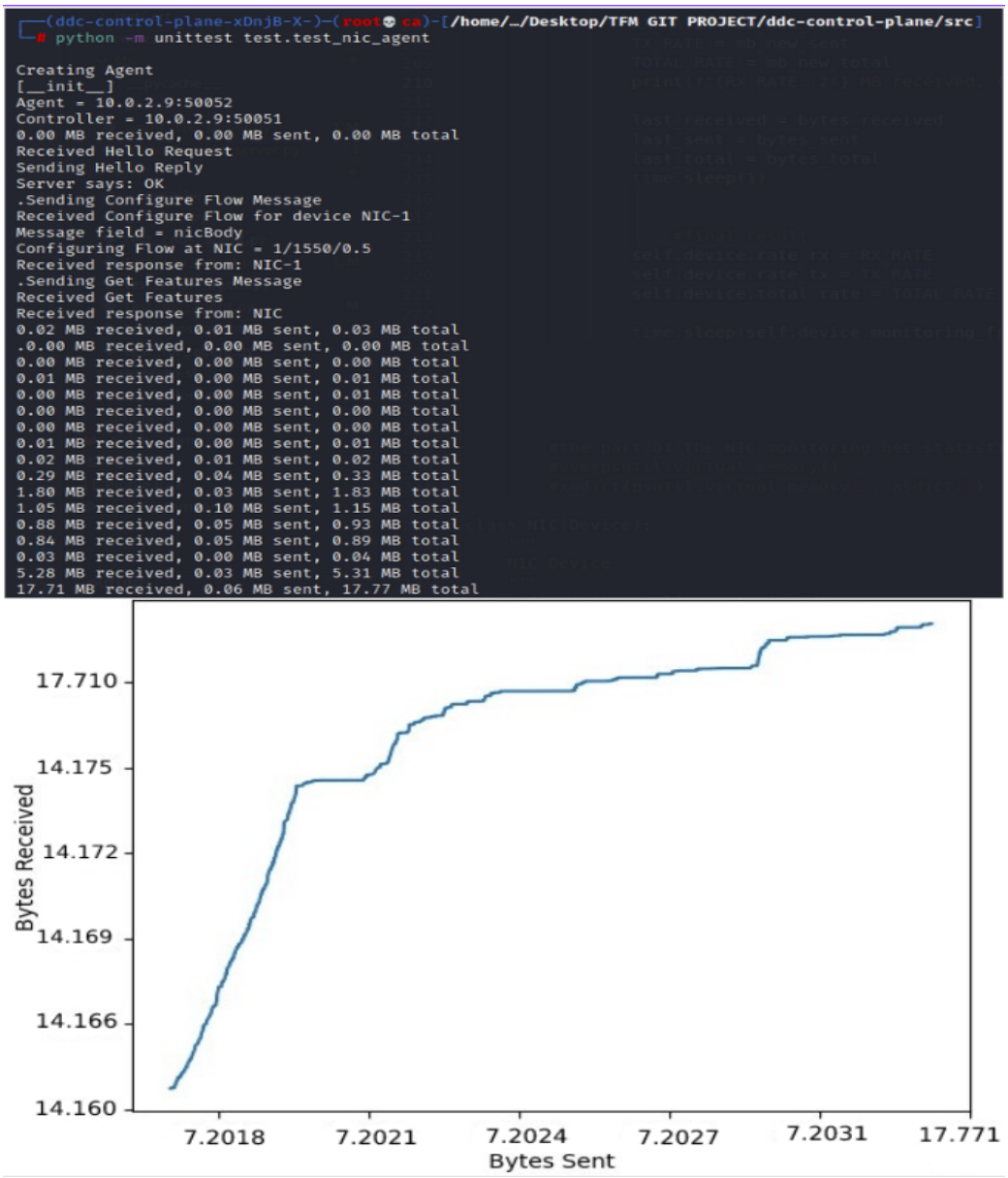

Figure 16: NIC Statistics

3.4 Security Part Test

Now let's take a look at the security part of securing the gRPC and securing the communication channel as described before, as seen in Figure 17, using wireshark to view and examine the network traffic of the grpc communication channel without using SSL/TLS, as seen it is possible to see and sniff the data (such as the configuration of the SBI controller serive) clearly without any kind of encryption; the information that contained in the gRPC message is in a plain text, it is necessary to prevent an unauthorized controller from taking control of the network or any third-party device from gaining access to sensitive network data so it is obligatory all of the communication through the southbound must be conducted via an encrypted channel, gRPC makes it simple and straightforward to communicate securely with other computer systems.

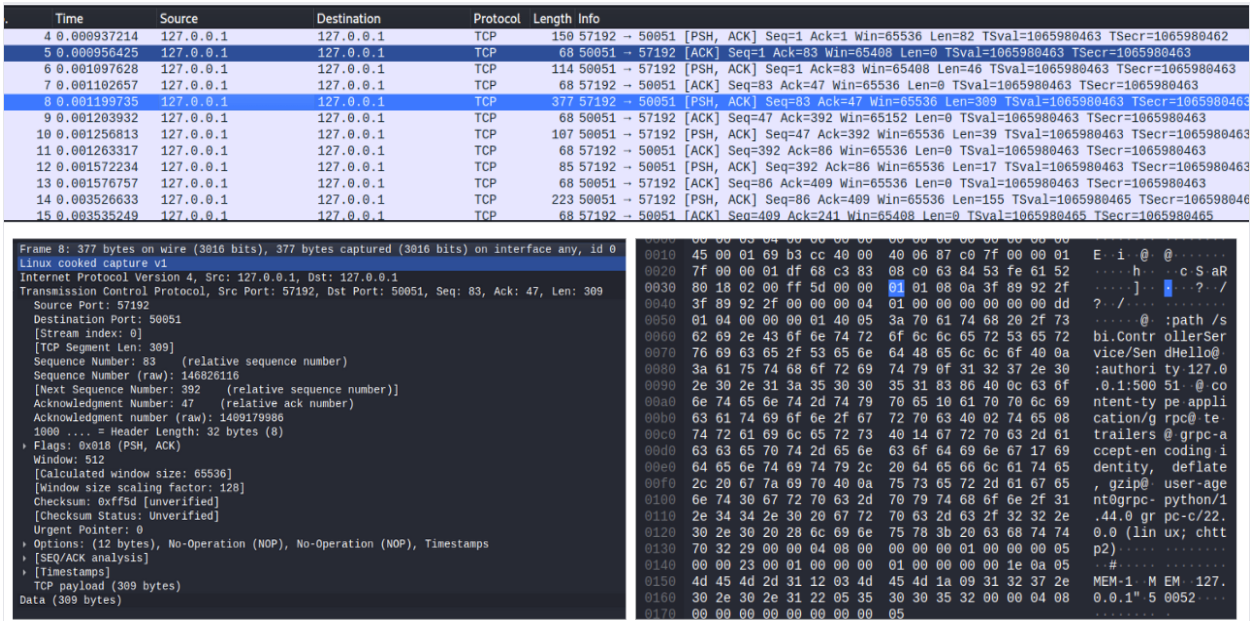

Figure 17: Capturing the packets without encryption

Then applied the implementation of the encryption by using SSL/TLS to secure gRPC, Figure 18 shows the data information of the captured packet, but it is encrypted via TLSv1.3 (can't interpret the data header of the packet); it helps to guarantee that traffic among a client and a server is reliable and trustworthy in both ways and also to prevent any malicious or third-party device from gaining access or sniffing any confidential network data.

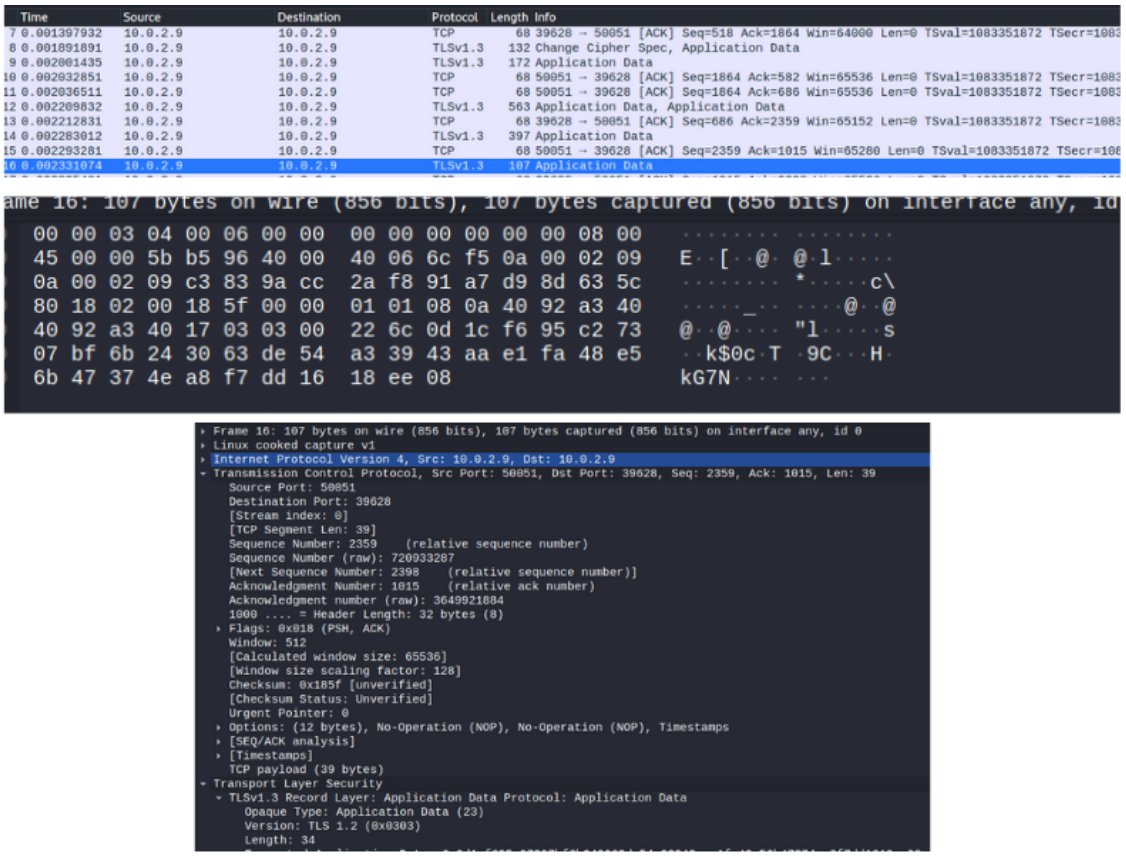

Figure 18:Capturing the packets with security (encryption)

4 Conclusions And Future Development:

As a conclusion to this thesis project, which centered on the DDC paradigm and the development of a secure monitoring framework for optical DDC, it also characterized the most significant characteristics of each kind of DCIs and compared them to each other, as was said in an earlier sentence. In particular, focused on developing a secure monitoring and control framework as well as the implementation of SDN agents; the control layer would be based on the principles of SDN. As a result, This will facilitate the architecture with high-speed and low-latency optical connections between the various hardware components in the DC in order to meet the requirements of higher-level services and applications. The Implementing server authentication (agent_API_server and controller_SBI_server) by importing the webserver certificate and the web server key into the server code, and then client authentication (both agent_SBI_client and controller_SBI_client) by importing the CA in the same manner as the server, are all necessary steps to ensure the security of the gRPC protocol and its associated communication channel. It is essential for disaggregation to be scalable, have high network performance, and have an easy deployment process; these are the requirements for a successful disaggregation. The primary objective of this thesis is to implement the control of a DDC infrastructure layer that is founded on the SDN and makes use of optical technologies for the DC network fabric, then securing the communication channel that is between the SBI and the SDN agents against eavesdropping and tampering; this will enable the execution of QA loops, for which a secure monitoring framework is needed. It makes it possible to gather the statistics of the critical metrics that are included in the agents by creating SDN-Agents (CPU_Agent, MEM_Agent, NIC_Agent); these SDN-Agents are tailored to the various blade types that we need to keep an eye on; they also provide with access to data for the relevant metrics. Moreover, the last section provided an analysis of the outcomes generated after implementing the secure monitoring by running the code to display real-time statistics and data information. After that is done, it will be able to gather statistics on how your system machine's resources, such as its CPU, RAM, and NIC, are being utilized. This will allow for securing the monitoring framework for optical DDC, and it becomes essential to guarantee a secure communication channel for gathering infrastructure metrics/statistics and enforcing (re-)configurations, closing the full loop, then addressing the security layer to secure the communication channel by encryption and providing authentication for the server and the client.

In the future, there will be additional work that needs to be done in order to expand this work by integrating with the monitoring manager of the SDN controller as the primary aim, ML/AI usage with deep learning approaches, making educated choices on deploying the resources at the most significant level possible, which could improve the effectiveness of the results.

ACRONYMS

DC	Data Centers
DDC	Disaggregated Data Centers
DCI	Data Centers Infrastructure
SDN	Software Defined Networking
DCN	Data Center Network
DAS	Direct Attached Storage
SAN	Storage Area Network
DCIM	Data Center Infrastructure Management
CI	Converged Infrastructure
IT	Information Technology
HCI	Hyper-Converged Infrastructure
CDI	Composable Disaggregated Infrastructure
API	Application Programming Interface
CPU	Central Processing Unit
GPU	Graphical Processing Unit
DCN	Data Center Network
NIC	Network Interface Card
SIC	Switch Interface Card
WDM	Wavelength Division Multiplexing
ToR	Top Of Rack
CDC	Colorless, Directionless, and Contentionless
VDC	Virtual Data Center

VMs	Virtual Machines
RWA	Routing and Wavelength Assignment
SDN	Software-Defined Networking
NF	Networking Functions
OS	Operating System
VLAN	Virtual Local Area Network
VRF	Virtual Routing Protocols
ISP	Internet Service Providers
VPN	Virtual Private Networks
VXLAN	Virtual Extensible LAN
MEC	Multi-chassis EtherChannel
ECMP	Cost Multi-Path
TLS	Transport Layer Security
SSL	Secure Socket Layer
IoT	Internet Of Things
NOS	Nanosecond Optical Switching
HTTPS	Hypertext Transfer Protocol Secure
ML	Machine Learning
AI	Artificial Intelligence
RAM	Random Access Memory
PM	Provisioning Manager
VPN	Virtual Private Network
IaaS	Infrastructure as a service
SBI	Southbound Interface

References

[1]. Fadaee, Mostafa & Haghighat, Fariborz & Nasiri, Fuzhan. (2020). Data center maintenance: applications and future research directions. Facilities. ahead-of-print. 10.1108/F-09-2019-0104.

[2]. Li, Chung-Sheng & Franke, Hubertus & Parris, Colin & Chang, Victor. (2015). Disaggregated Architecture for At Scale Computing. 10.5220/0005551300450052.

[3]. Yan, Fulong & Miao, Wang & Raz, O. & Calabretta, Nicola. (2017). OPSquare: A Flat DCN Architecture Based on Flow-Controlled Optical Packet Switches. Journal of Optical Communications and Networking. 9. 291. 10.1364/JOCN.9.000291.

[4]. W. Miao, F. Agraz, S. Peng, S. Spadaro, G. Bernini, J. Perelló, G. Zervas, R. Nejabati, N. Ciulli, D. Simeonidou, H. Dorren and N. Calabretta, "SDN-enabled OPS with QoS guarantee for reconfigurable virtual data center networks", IEEE/OSA Journal of Optical Communications and Networking, vol. 7, no. 7, pp. 634-643, July 2015.

[5]. J. Bigelow, S. (2022, May 18). "How to design and build a data center." Retrieved July 7, 2022,[Online].Available: https://www.techtarget.com/searchdatacenter/How-to-design-and-build-a-data-center

[6]. Gillis, A. S. (2022, April 21). Data Center Infrastructure Management (DCIM) . SearchDataCenter. Retrieved July 7, 2022, from https://www.techtarget.com/searchdatacenter/definition/data-center-infrastructure-management-DCIM?_gl=1%2Aqvd55r%2A_ga%2ANTQ4NjMxMjYuMTY1NDU1OTkxMw..%2A_ga_TQKE4GS5P9%2AMTY1NzE1NTQ1Ni45LjEuMTY1NzE1NjA1OC4w&_ga=2.111853005.851595955.1657155457-54863126.1654559913

[7]. Sheldon, R. (2020, June 24). Pros and cons of composable architecture vs. traditional storage. SearchStorage. Retrieved July 7, 2022, from https://www.techtarget.com/searchstorage/feature/Pros-and-cons-of-composable-architecture-vs-traditional-storage

[8]. Kranz, G. (2017, October 25). What is converged infrastructure? . SearchDataCenter. Retrieved July 7, 2022, from https://www.techtarget.com/searchdatacenter/definition/converged-infrastructure?_gl=1%2At4hsi8%2A_ga%2ANTQ4NjMxMjYuMTY1NDU1OTkxMw..%2A_ga_TQKE4GS5P9%2AMTY1NzE1NTQ1Ni45LjEuMTY1NzE1NzE2OS4w&_ga=2.114512718.851595955.1657155457-54863126.1654559913

[9]. Bigelow, S. J. (2020, August 17). What is hyper-converged infrastructure? guide to HCI. SearchDataCenter. Retrieved July 7, 2022, from https://www.techtarget.com/searchdatacenter/definition/What-is-hyper-converged-infrastructure-Guide-to-HCI?_gl=1%2Ax78t43%2A_ga%2ANTQ4NjMxMjYuMTY1NDU1OTkxMw..%2A_ga_TQKE4GS5P9%2AMTY1NzE1NTQ1Ni45LjEuMTY1NzE1NzI4OS4w&_ga=2.13996926.851595955.1657155457-54863126.1654559913

[10]. J. A. Miller, "Words to go: Composable disaggregated infrastructure ," SearchDataCenter, 17-Jul-2019. [Online]. Available: https://www.techtarget.com/searchdatacenter/feature/Words-to-go-Composable-disaggregated-infrastructure-glossary?_gl=1%2A1o9mrzc%2A_ga%2ANTQ4NjMxMjYuMTY1NDU1OTkxMw..%2A_ga_TQKE4GS5P9%2AMTY1NzE1NTQ1Ni45LjEuMTY1NzE1Nzg0MC4w&_ga=2.140792187.851595955.1657155457-54863126.1654559913. [Accessed: 07-Jul-2022].

[11]. B. Posey, "Disaggregated hyper-converged infrastructure vs. traditional HCI," SearchDataCenter, 17-Mar-2020. [Online]. Available: https://www.techtarget.com/searchdatacenter/feature/Disaggregated-hyper-converged-infrastructure-vs-traditional-HCI?_gl=1%2A1jtmrvj%2A_ga%2ANTQ4NjMxMjYuMTY1NDU1OTkxMw..%2A_ga_TQKE4GS5P9%2AMTY1NzE1NTQ1Ni45LjEuMTY1NzE1ODE2MS4w&_ga=2.88300370.851595955.1657155457-54863126.1654559913. [Accessed: 07-Jul-2022].

[12]. C. Lluca, "Resource allocation in disaggregated optical networks," Pàgina inicial de UPCommons, 07-Nov-2019. [Online]. Available: https://upcommons.upc.edu/handle/2117/171950. [Accessed: 07-Jul-2022].

[13]. H. Adams, "Trends in Metro Optical Networks," March 2018. [Online]. Available: https://www.juniper.net/assets/us/en/local/pdf/industry-reports/2000696-en.pdf *[Accessed: 07-Jul-2022].*

[14]. J. Santos, N. Costa and J. Pedro, "Cost-Effectiveness Assessment of Transport Networks based on Disaggregated Optical Plataforms," in 2017 Optical Fiber Communications Conference and Exhibition (OFC), Los Angeles (USA), January 2017.

[15]. Y. Yan et al., "All-Optical Programmable Disaggregated Data Centre Network Realized by FPGA-Based Switch and Interface Card," in Journal of Lightwave Technology, vol. 34, no. 8, pp. 1925-1932, 15 April15, 2016, doi: 10.1109/JLT.2016.2518492.

[16]. G. M. Saridis, E. Hugues-Salas, Y. Yan, S. Yan, S. Poole, G. Zervas and D. Simeonidou, "DORIOS: Demonstration of an all-optical distributed CPU, memory, storage intra DCN interconnect", Proceedings of Optical Fiber Communications Conference and Exhibition 2015 (OFC 2015), March 2015.

[17]. C. Yuxin, D. Marilet and L. W., "Resource Disaggregation Versus Integrated Servers in Data Centers: Impact of Internal Transmission Capacity Limitation," in 2018 European Conference on Optical Communication (ECOC), Rome (Italy), Semptember 2018.

[18]. A. Pagès, J. Perelló, F. Agraz, and S. Spadaro, "Optimal VDC Service Provisioning in Optically Interconnected Disaggregated Data Centers," IEEE Communications Letters, vol. 20, pp. 1353-1356, 2016.

[19]. Cabaj, Krzysztof & Wytrębowicz, Jacek & Kuklinski, Slawomir & Radziszewski, Paweł & Dinh, Khoa. (2014). SDN Architecture Impact on Network Security. 10.15439/2014F473.

[20]. D. Comer and A. Rastegarnia, "Toward Disaggregating the SDN Control Plane," in IEEE Communications Magazine, vol. 57, no. 10, pp. 70-75, October 2019, doi: 10.1109/MCOM.001.1900063.

[21]. S. Tunas , "Software define data center," Sapta Tunas Teknologi, 2020. [Online]. Available: https://www.saptatunas.com/software-define-data-center/. [Accessed: 07-Jul-2022].

[22]. Ubuntu, Ed., "What is kubernetes," Ubuntu, 2022. [Online]. Available: https://ubuntu.com/kubernetes/what-is-kubernetes. [Accessed: 07-Jul-2022].

[23]. Rhodes, Brandon & Goerzen, John. (2014). TLS/SSL. 10.1007/978-1-4302-5855-1_6.

[24]. Chris, "SDN and Network Disaggregation Simplified: Packet pushers," SDN and Network Disaggregation Simplified | Packet Pushers, 16-Mar-2017. [Online]. Available: https://packetpushers.net/simplified-approach-sdn-network-disaggregation/. [Accessed: 07-Jul-2022].

[25]. Cloudflare, Ed., "What is SSL (secure sockets layer)? | ," What is SSL? [Online]. Available: https://www.cloudflare.com/learning/ssl/what-is-ssl/. [Accessed: 07-Jul-2022].

[26]. Cloudflare, Ed., "What is MTLS? | mutual TLS - cloudflare," What is MTLS? [Online]. Available: https://www.cloudflare.com/learning/access-management/what-is-mutual-tls/. [Accessed: 07-Jul-2022].

[27]. D. Syrivelis et al., "A software-defined architecture and prototype for disaggregated memory rack scale systems," 2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2017, pp. 300-307, doi: 10.1109/SAMOS.2017.8344644.

[28]. R. Lin, Y. Cheng, M. D. Andrade, L. Wosinska and J. Chen, "Disaggregated Data Centers: Challenges and Trade-offs," in IEEE Communications Magazine, vol. 58, no. 2, pp. 20-26, February 2020, doi: 10.1109/MCOM.001.1900612.

[29]. Li, Chung-Sheng & Franke, Hubertus & Parris, Colin & Chang, Victor. (2015). Disaggregated Architecture for At Scale Computing. 10.5220/0005551300450052.

[30]. F. Agraz, A. Pagès and S. Spadaro, "Data Plane Technology-agnostic Control and Orchestration Architecture for Optical Disaggregated Data Centers," 2022 Optical Fiber Communications Conference and Exhibition (OFC), 2022, pp. 1-3.

[31]. X. Guo et al., "Experimental Demonstration of SDN-enabled Reconfigurable Disaggregated Data Center Infrastructure," 2021 European Conference on Optical Communication (ECOC), 2021, pp. 1-4, doi: 10.1109/ECOC52684.2021.9605888.

[32]. A. Pagès, R. Serrano, J. Perelló, and S. Spadaro, "On the benefits of resource disaggregation for virtual data centre provisioning in Optical Data Centres," Computer Communications, 04-Apr-2017. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0140366417303808. [Accessed: 07-Jul-2022].

[33]. gRPC (Ed.). (2022, March 30). Securing gRPC using Python. gRPC. Retrieved July 10, 2022, from https://grpc.io/docs/languages/python/quickstart/

[34]. Heisig, M. (2018, December 26). SAP Hana on hyperconverged infrastructure. SAP Blogs. Retrieved August 15, 2022, from https://blogs.sap.com/2018/10/16/sap-hana-on-hyperconverged-infrastructure/

[35]. Haleplidis, E. (2017, September). Overview of RFC7426: SDN layers and architecture terminology. IEEE Software Defined Networks. Retrieved August 20, 2022, from https://sdn.ieee.org/newsletter/september-2017/overview-of-rfc7426-sdn-layers-and-architecture-terminology