2024 EuCNC & 6G Summit - OPE 1570991305
2024 European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit): Operational & Experimental Insights (OPE)

2024 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit) | 979-8-3503-4499-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/EuCNC/6GSummit60053.2024.10597110

On the Automated Scaling of User Plane Function for 5G: An Experimental Evaluation

Sokratis Christakis*‡, Nikos Makris*†, Thanasis Korakis*† and Serge Fdida‡
*Dept. of Electrical and Computer Engineering, University of Thessaly, Greece
†Centre for Research and Technology Hellas, CERTH, Greece
‡Laboratoire LiP6/CNRS, Sorbonne University, Paris, France
Email: schristakis@uth.gr, nimakris@uth.gr, korakis@uth.gr, serge.fdida@lip6.fr

***Abstract*—The advent of emerging technologies like autonomous vehicles, AR/VR, IoT, smart cities, etc., have raised the bar in terms of dealing with enormous data transfer, high speeds, and low response times. 5G has been designed and is optimized unceasingly to try to meet these demands. The quick evolution of 5G Radio Access Network (RAN) technologies has ushered in a new era of connectivity and communication. This has caused the 5G Core Network to face significant problems, as on many occasions, the RAN's ability to transmit data overwhelms the core network's ability to handle it. As a result, network congestion occurs, leading to reduced network speeds, significant delays, and occasional disruptions, which have a profound effect on low-quality user experience. This issue has triggered the scientific community to investigate ways to enhance the performance of the 5G core network, to match with the evolving demands of RAN technologies. In this work, we present an innovative algorithm for dynamic management and optimization of 5G network resources within a Kubernetes cluster environment. The algorithm's functionality revolves around monitoring metrics of User Plane Function (UPF) and making real-time decisions on deploying multiple UPFs within the cluster to ensure enhanced network performance and cost optimization.**

***Index Terms*—Kubernetes, 5G Network, UPF, Core Network, Scaling, Dynamic Algorithm**

I. Introduction

5G is the 5th generation of cellular networks designed to meet the large increase in data and connectivity of today's modern society, with billions of connected devices. The innovations and technologies that make it up, as well as its inventive architecture, give the ability to connect billions of devices, with very short response times (latency) and extremely high speeds that can reach up to 100 times the speeds offered by its predecessor 4G/LTE. 5G has helped make the most of technologies like cloud computing and changed the way we perceive technology, allowing us to perform calculations and access cloud services with impressive efficiency. Already from the 5th generation, pieces of the Core Network run as self-contained Network functions, natively in the Cloud (Cloud Native) either as virtual machines or as micro-services.

This has been enabled through the definition of the Service Based Architecture (SBA) for the 5G Core Network, allowing the different parts of the core network to run completely distributed, with tools previously used for service management hosted in the Cloud. As a result of the above, tools such as Kubernetes / Redhat Openshift [1], OpenStack [2] are used as a basis for the design of new ones, aimed exclusively at the management of the Core Network (e.g. ONF AETHER, ONF CORD, Google Nephio [3]). The use of such tools unleashes new possibilities for the management of these functions, which until now did not exist, such as dynamic scaling (Horizontal and Vertical scaling), transfer (Migration) and scheduling in other data centers. Furthermore, these services can indirectly manage various aspects of the telecommunication network. This includes tasks like selecting the appropriate edge for hosting the User Plane Functions for Multi-access Edge Computing, slicing the telecommunications network, and others.

The core network constitutes the central computing and management core of the 5G network. It performs key 5G network functions related to data security, resource management, and traffic control. In addition, it is responsible for the registration of the various devices in the wireless network through authentication actions, while being the interface through which the data ends up on the Internet. From a practical point of view, the component responsible for transferring data from the 5G wireless network to the Internet, is the User Plane Function (UPF). More specifically, the user equipment (UE) connects to the closest gNB and after registering with the provider, it sends data over the wireless channel. The gNB relays the data to the UPF, which in turn forwards the data to any Data Network (DN). The reverse path is followed in the opposite direction.

In this work, we build upon this cloud native approach for the 5G Core Network, and design, implement and evaluate a scheme for the dynamic scaling of the UPF functions at the edge of the network. We employ an experimentally driven approach, deploying the OpenAirInterface (OAI) platform as a gNB and cloud-native 5G Core Network. The scaling process is managed through metrics collected in real-time from the network operation and which reflect the load and energy efficiency aspects of the network. Our contributions are summarized as follows:

- Efficient Resource Management: Kubernetes and deploying the 5G network as cloud-native functions enables dynamic resource allocation, optimizing UPF's resource usage.
- Dynamic Congestion Handling: Scaling UPF based on metrics prevents congestion, ensuring better performance.

The research leading to these results has received funding from the European Horizon 2020 Programme for research, technological development and demonstration under Grant Agreement Number No 101008468 (H2020 SLICES-SC). The European Union and its agencies are not liable or otherwise responsible for the contents of this document; its content reflects the view of its authors only.

Authorized licensed use limited to: University of Thessaly. Downloaded on February 24,2025 at 08:35:33 UTC from IEEE Xplore. Restrictions apply.

- Cost Optimization: Resource scaling reduces operational costs by allocating resources as needed.
- Scalable and Adaptable: This approach offers flexibility and scalability to meet changing network demands efficiently.
- Actual Experiment Conditions: Implementing these experiments under actual network conditions ensures practical applicability and effectiveness.

The rest of the paper is structured as follows. In Section II, details are given on research and studies related to this work. In sections III and IV, we provide details on the system architecture and our experimental findings. Finally, in Section V, we conclude and discuss some possible future work.

II. Related Work

With the advent of cloud-native computing and networking, several works have focused on building functionality on top of new features that frameworks around this ecosystem can provide. One of such features is the support of auto-scaling, as a means to accommodate the demand for the services. Scaling is broken down into three different approaches: 1) Horizontal Scaling, where more replicas of the service are spawned, and traffic is balanced between them, 2) Vertical Scaling, where more/fewer resources are applied to the running service to meet the demand (adding/removing CPU cores/memory), and 3) Cluster Auto-scaling, where the cluster hosting the services offers more/less resources. Several of these features are implemented in well-established frameworks used for cloud-native research and industrial environments. The most notable ones that support all three types of scaling are Kubernetes (K8s), ETSI Open Source MANO [4], and Hashicorp Nomad [5]. Different works exploit such functionality towards defining new algorithms/policies, as well as services that can auto-scale to meet the service demand. In [6], authors investigate how the Mobility and Management Entity shall auto-scale in 4G networks, towards meeting high swarms of users entering the cell. Authors in [7] provide a vendor-agnostic system architecture that enables the elasticity of cloud-native services utilizing automated scaling and resource provisioning.

As such decisions rely on metrics collected directly from the platforms hosting the services, they can be easily annotated with timestamps and in turn, create a time-series of metrics. Such time-series can be fed to Neural Networks, to predict their future values, and hence proactively apply decisions for the network. Authors in [8], [9], [10] employ such an approach, with differentiating factors the models used to train and predict the future values. In [11] and [12], authors employ a similar approach for auto-scaling the Authentication Management Function (AMF) entity of the 5G network. An analysis of the data, training models, accuracy of predictions, and benefits of scaling are provided.

Nevertheless, the majority of such works only address the control plane part of the telecommunications network (5G AMF/4G LTE) or apply to stateless services. In this work, we progress beyond existing state-of-the-art and propose the scaling process for the 5G User Plane Function (UPF), that is in charge of breaking out the traffic to Data Networks. Works [13] and [14] suggest a proactive solution, which relies on deploying UPFs utilizing machine learning techniques. Our work focuses on deploying UPFs dynamically and only when necessary, by observing real-time metrics. Authors in [15] suggest a well-structured way of scaling VPP-based UPFs based on VPP rate, which represents the number of packets that can be processed simultaneously. Our work aims to developing an algorithm that can be utilized in any 5G network testbed and can operate with any type of UPF implementation (SPGWU, VPP, EBPF etc.). In [16], authors explore the benefits that are brought by the reconfiguration of the UPF component in relationship to the different connections with the rest of the 5G network entities. Works [17] and [18] apply the auto-scaling function provided by Kuberentes to the UPF function of the network, and evaluate their approach using simulations. In addition, in [19], the authors suggest a 4G/LTE-based model that scales Data Plane resources based on Bit rate Aware Auto Scaling (BAAS). We strictly follow an experimentally driven approach (beyond simulations) applied in real networks, to validate the architecture and quantify the benefits of the proposed approach. The custom-built auto-scaler mitigates the high triggering times of such tools.

III. System Architecture

In this section, we will provide a detailed overview of the key components and the system configuration. To start with, we employ a Kubernetes cluster consisting of one master node and 2 worker nodes, which helps us manage our 5G network infrastructure easily, ensuring scalability and efficient resource management. For the nodes to communicate seamlessly in our cluster, we deploy Flannel overlay management network. As noted, the 5G core network components operate as independent functions, benefiting from SBA's capacity to allow operation in a fully distributed manner. Having this ability, Kubernetes leverages Docker containers to deploy our 5G Netrork in a micro-services environment. Multus CNI is aslo deployed to increase network versatility by enabling multiple network interfaces within a single pod.

TABLE I: Master Specifications

Node / Specs	Master Node
CPU	Intel(R) Core(TM) i7-8700 CPU
Cores	12
RAM	16GB
Operating System	Ubuntu 18.04.6 LTS
Docker Version	19.03.6
Kubernetes version	v1.20.0
Tasks Executed	5G-Core Functions, UERANSIM, Prometheus/Grafana

It is worth mentioning that our system operates within a real experimental infrastructure called NITOS [20] (part of the SLICES RI [21]). NITOS provides a controlled and reliable environment for testing our 5G Network, confirming that our work is applicable to real-world scenarios and environments. The detailed specifications of each cluster node can be retrieved from Tables I,II.

As mentioned earlier, our 5G Network deployment lies in the OpenAirInterface (OAI) platform [22], which implements 5G networks with the use of software. In that matter, we

Authorized licensed use limited to: University of Thessaly. Downloaded on February 24,2025 at 08:35:33 UTC from IEEE Xplore. Restrictions apply.

deploy the 5G core components, including the Network Repository Function (NRF), the User Data Repository (UDR), the Unified Data Management (UDM), the Authentication Server Function (AUSF), the Access and Mobility Management Function (AMF), the Session Management Function (SMF) and the UPF. For the Radio Access Network (RAN) part, we deploy UERANSIM [23], an open-source 5G RAN simulator, that is designed to simulate the behavior of 5G RAN, giving us the ability to perform advanced 5G experiments. When the UE successfully connects to the core network, a Packet Data Unit Session (PDU) is created between the UE and the UPF, allowing data traffic. This configuration can be observed in Figure 3a and depicts a monolithic 5G network architecture. In our work, we extend this setup (Figure 3b) by incorporating multiple User Plane Functions (UPFs), which are responsible for managing user data traffic. More specifically the 5G model that we deploy is capable of supporting multiple UPFs, while keeping the number of the other network functions the same (1 AMF, 1 SMF etc.). Each UPF is associated with a unique Data Network Name (DNN), which plays a vital role, as it allows us to astutely connect the UEs to the specific UPF that matches the conditions it necessitates. In a state where a UPF is experiencing high load, nearing its capacity, and the link is at the brink of degradation, we utilize the DNNs to reroute User Equipment (UE) to a different UPF that has the capacity and resources to maintain the link's stability and prevent a potential network breakdown.

TABLE II: Worker1 & Worker2 Specifications

Node / Specs	Worker1 & Worker 2
CPU	Intel Core Processor (Skylake, IBRS)
Cores	4
RAM	8GB
Operating System	Ubuntu 18.04.6 LTS
Docker Version	19.03.6
Kubernetes Version	v1.20.0
Tasks Executed	UPF2 , UPF3, Prometheus/Grafana

In addition, we employ Prometheus [24] and Grafana [25] to efficiently monitor and visualise our cluster's performance. Prometheus acts as data tracker, collecting metrics (CPU consumption, packet reception etc.) through NodeExported from the Kubernetes cluster, to provide end-to-end monitoring. These metrics are then passed to Grafana, which serves as a visualization and data extraction tool. Grafana generates sophisticated graphs and dashboards while giving the ability to store and process the collected data for thorough evaluation.

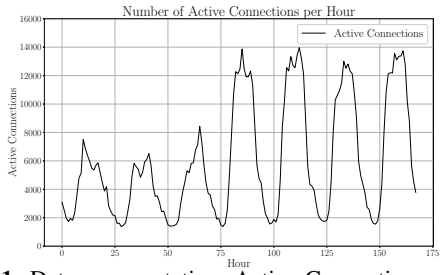


Fig. 1: Data representation: Active Connections per Hour

In order to emulate a real-world scenario, we have used an online available dataset [26]. The datasets provided, contain information about telecommunication activities in Milan and the Province of Trentino. Due to the extensive size of the dataset, we performed an extraction, obtaining a subset of 170 data points that describe the active connections per hour (Figure 1). This subset contains a wide range of the amount of connections per hour, spanning from low to high values, in order to offer a thorough representation of real world scenarios. To be able to utilize the data in our experiments, we converted it to throughput data. We assume that the number of connections correlate to the load and the usage of the network. To elaborate further, we spotted the highest value in the data that reflects the maximum number of active connections and mapped that value to the maximum Mbps (i.e. 500Mbps) that our system can handle before the PDU session (tunnel) drops. Then, we scaled the rest of the values with respect to the highest value. The produced dataset used for the experiments represents the total throughput generated per hour and can be seen in Figure 2.

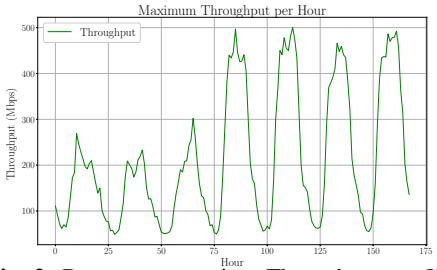
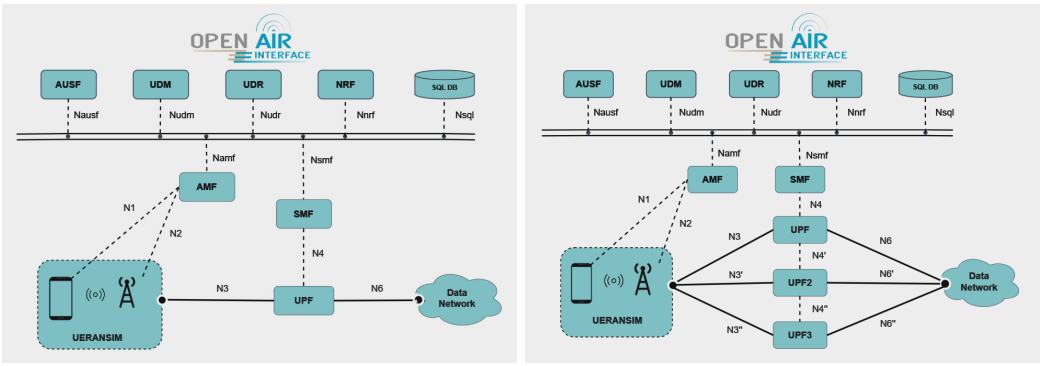


Fig. 2: Data representation: Throughput per Hour

As noted earlier, the main idea of this work is the design and implementation of a dynamic auto-scaling algorithm. The algorithm starts by deploying the monolithic 5G architecture within our cluster, initially featuring a single UPF. Subsequently, it reads the sequence of throughput values from our dataset and initiates traffic with the use of the iperf command for 20 seconds for each value. It is worth elaborating that each throughput value is read from the dataset, split randomly to the UEs, and generated by each UE. This traffic generated by UEs is initially directed towards the data network through the original UPF. Furthermore, the algorithm uses threads to continuously monitor the CPU consumption of the UPF and establishes two predefined limits (indicative and not definitive values), $LimitA$ (i.e. 67%) and $LimitB$ (i.e. 87%). If the CPU consumption exceeds $LimitA$, the algorithm responds by deploying a new UPF function and assigning it to another node within the cluster. It is important to mention that every new UPF that gets deployed, is using a unique DNN in order to give us the flexibility to associate the various UEs on demand. When the CPU consumption surpasses $LimitB$, the algorithm deploys two new UPF functions to match the escalating network demands. In the event of a new UPF deployment, a dynamic process takes place, where some of the UEs, initially associated with the original UPF, change their UPF association to the new one. They then begin transmitting data to evenly distribute the network traffic. For example, if the CPU consumption surpasses $LimitA$ but remains below

Authorized licensed use limited to: University of Thessaly. Downloaded on February 24,2025 at 08:35:33 UTC from IEEE Xplore. Restrictions apply.



(a) Monolithic Architecture **(b)** Dynamic Algorithm Architecture

Fig. 3: Monolithic vs Dynamic Algorithm Architecture

Algorithm 1 Dynamic Algorithm

```
Initialize:
- Create a single UPF (User Plane Function)
- Set CPU utilization LimitA and LimitB
- Create variables for tracking deployed UPFs and UE assignments
function MonitorCPU
    Initialize a thread for CPU monitoring
    while Algorithm is running do
        - Measure current CPU consumption of the UPFs
        - Update CPU monitoring data
        Sleep(for a short duration)
    end while
end function
MonitorCPU
repeat
    - Read the next throughput value from the dataset
    if CPU consumption exceeds LimitB then
        - Deploy two new UPFs and assign them to available nodes
        - Update UPF tracking variables
    else if CPU consumption exceeds LimitA then
        - Deploy one new UPF and assign it to an available node
        - Update UPF tracking variables
    end if
    for each deployed UPF do
        - Reassign UEs to balance network traffic
        - Measure and adjust CPU consumption as needed
    end for
    if the next dataset value falls below both CPU utilization limits then
        if Number of active UPFs > 1 then
            if CPU consumption is between LimitA and B then
                - Reassign UEs to the remaining two UPFs as needed
                - Deactivate the third UPF
            else if CPU consumption is below LimitA then
                - Reassign UEs to the original UPF
                - Deactivate two UPFs, keeping only the original
            end if
        end if
    end if
    Continue network traffic generation and monitoring
until End
```

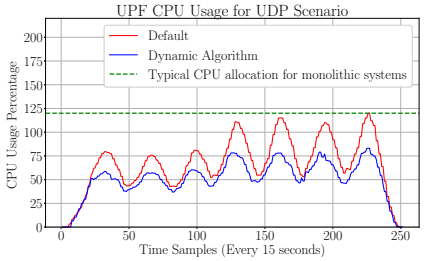
$LimitB$, a new UPF node is deployed, and half of the UEs are reassigned to UPF2. When $LimitB$ is exceeded, the algorithm also redistributes the UEs, leaving 1/3 of the UEs with the original UPF, 1/3 switching to UPF2, and the remaining assigned to UPF3. This redistribution ensures that the network traffic is distributed among the active UPFs, optimizing network performance. On the other hand, the algorithm can also adapt to situations where multiple UPFs were previously deployed and no longer needed. For example, if the next dataset value is low enough that the CPU utilization is under both limits, the UEs fall back to the original UPF and the extra deployed UPFs are terminated. Respectively, in the scenario where the CPU allocation is between the two limits, the UEs are evenly reassigned to the remaining UPFs and the 3rd UPF is terminated. The high-level concept of the algorithm is depicted in Algorithm 1. It's important to recognize that while some data loss might happen during the re-assignment process, it is significantly less than the data loss experienced when the CPU is approaching its maximum capacity.

IV. Experimental Findings

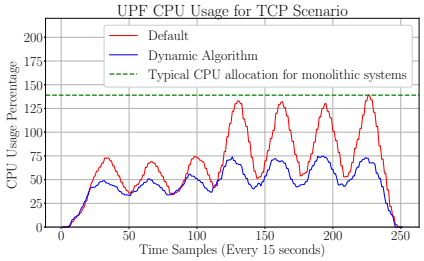
This section delves into the experimental part of our research, where we bridge the gap between theory and practice. It is worth mentioning that each experiment was conducted multiple times to ensure the consistency of the algorithm. The experiments are held under two categories: the monolithic (default) architecture and the architecture enhanced by the dynamic scaling algorithm. Within each category, we focus on two scenarios, one involving the User Datagram Protocol (UDP) and the other using the Transmission Control Protocol (TCP). Employing both UDP and TCP protocols in our experiments ensures a comprehensive evaluation of our architecture. It allows us to explore how our architectural decisions adapt to the unique characteristics of each protocol and perform a comparative analysis. In the first series of experiments, we examine the monolithic architecture. We run iperf commands for all the throughput values from our dataset, for both UDP and TCP scenarios independently, where each iperf command lasts 20 seconds. Subsequently, we perform the exact same sequence of experiments, while also deploying the dynamic algorithm, in order to validate its ability to meet the evolving demands in contrast with the monolithic scenario.

After conducting our experiments, we proceeded to extract and analyze data, generating graphs using Prometheus and Grafana. Figures 4a and 4b illustrate a comparative analysis of CPU allocation in the monolithic and dynamic architectures for both UDP and TCP scenarios. Our observations reveal that, in both protocol scenarios, the CPU usage in the monolithic architecture occasionally exceeds 100%, signifying intensive resource utilization across multiple CPU cores. Specifically, in the TCP scenario, CPU usage spikes to 140%, while in the UDP scenario, it reaches 120%. Notably, TCP exhibits higher peaks in CPU allocation, due to the fact that when the link (i.e.

Authorized licensed use limited to: University of Thessaly. Downloaded on February 24,2025 at 08:35:33 UTC from IEEE Xplore. Restrictions apply.



(a) Monolithic vs. Algorithm for the UDP scenario



(b) Monolithic vs. Algorithm for the TCP scenario

Fig. 4: UPF CPU Allocation Comparison for UDP and TCP

PDU session) and the UPF CPU are stressed, it triggers retransmissions, resulting to additional network load. In contrast, when our scaling algorithm operates, we observe (Figures 4a & 4b) major differences in CPU allocation.

Peak CPU usage reaches around 83% for UDP and approximately 74% for TCP. These numbers represent a marked improvement compared to the monolithic approach, where CPU usage exceeds 100% in high throughput values. In the dynamic setup, CPU usage stays within manageable levels, preventing resource congestion and performance issues. This means that the dynamic scaling achieve better CPU allocation, resulting in a smoother and more reliable network operation.

Figure 5 & Figure 6 present a visual comparison of the UPF packet reception rate (within a 5-minute window) for both the monolithic and dynamic architectures in the context of UDP and TCP scenarios. In order to retrieve this metric, we utilized the Prometheus container-network-receive-packets-total query and specified the interface value so it can track packets that come specifically from the PDU session (i.e. tun0 interface).

The generated graphs reveal substantial differences in packet reception rates. The points where the extra UPFs are deployed can be easily identified by examining the respective UPF lines. In the monolithic architecture, the packet reception rate reaches 45.000 and approximately 34.000 packets for UDP and TCP, respectively. In the UDP scenario, the rate of received packets exceeds that of the TCP scenario. This difference can be explained by UDP's inherent characteristics, which include lower overhead and fewer data reliability measures. This makes UDP transmit packets at a higher rate compared to TCP, which ensures controlled and reliable data delivery. This highlights how the protocol directly affects the rate of packet reception, giving insights into the trade-offs between speed and reliability in network communication. However, when the dynamic mechanism operates, we observe notable differences in peak packet reception rates. In this scenario, maximum peak values reach approximately 22.500 packets for UDP and around 18.500 for TCP. In both UDP and TCP scenarios, during the initial phase, represented by the first three peaks, packets are efficiently forwarded to the two active UPFs.

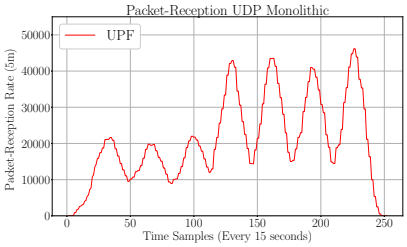
However, as network demands maximize (in the final four peaks), we observe that three active UPFs are deployed to manage the increased traffic load, dividing the network traffic.

Overall, the initial approach of directing all traffic through a single UPF, pushing it to the brink of overload, underscored the limitations of a monolithic system. However, the dynamic scaling algorithm's ability to evenly distribute traffic, while keeping the link uncongested, suggests a significant improvement. By efficiently managing resources and deploying additional UPFs as needed, the dynamic mechanism prevented network congestion and ensured reliable performance. Furthermore, the observed CPU allocation emphasizes the advantages of dynamic scaling further, showcasing its ability to maintain optimal performance, while efficiently managing CPU resources. Projecting the CPU consumption directly to the energy consumption of the platform, it is evident that lower CPU utilization leads to higher energy efficiency. Arguably, when the CPU allocation reaches full capacity, high temperature and bad scheduling can occur, leading to higher overall energy consumption compared to distributing the workload to other CPUs. For the cases of monolithic deployment, resources need to be over-provisioned during their initial deployment to meet the demand (provisioning for the peak of the demand). Therefore, an automatic scaling process can provide clear benefits for the case of the overall energy efficiency, allowing provisioning of lower-utilized UPF deployments only when needed. It's worth noting that the algorithm can be easily deployed and employed across various machines, as it doesn't necessitate the use of advanced tools. In addition, it can support the deployment of more than 3 UPFs and adjust accordingly. However, mindlessly deploying new UPFs can lead to wasted resources, as our goal is to reduce CPU usage to levels that don't affect network performance, not zero it.

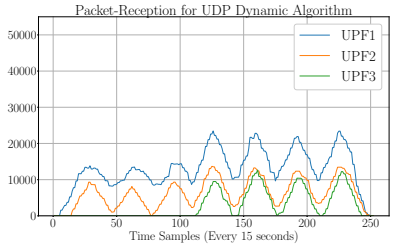
V. Conclusion

In this work, we have deployed a 5G network in a real-world micro-services environment with the use of Kubernetes & NITOS testbed. Subsequently, we have designed and implemented a dynamic algorithm, as opposed to the traditional monolithic approach, to incorporate additional UPF functions based on real-time metrics, that reflect the network conditions. Extensive experiments using a real-world telecommunications dataset have been employed to evaluate our approach. By analyzing the generated data and graphs, we have seen obvious improvements in the CPU allocation percentage, as well as in the network congestion part. All in all, the dynamic mechanism demonstrated its efficiency, by optimizing resource management, reducing costs, and enhancing network reliability. In the future, we consider conducting more complex scenarios, taking more parameters into consideration and see how our algorithm works with multiple UPFs (up to 10). Additionally,

Authorized licensed use limited to: University of Thessaly. Downloaded on February 24,2025 at 08:35:33 UTC from IEEE Xplore. Restrictions apply.

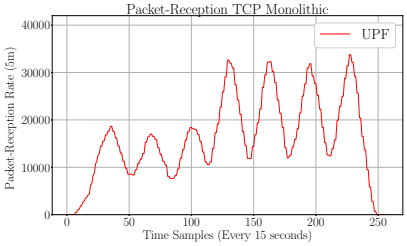


(a) Monolithic Packet Reception for the UDP scenario

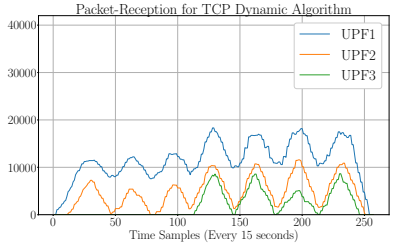


(b) Dynamic Algorithm Packet Reception for the UDP scenario

Fig. 5: UPF Packet Reception for the UDP scenario



(a) Monolithic Packet Reception for the TCP scenario



(b) Dynamic Algorithm Packet Reception for the TCP scenario

Fig. 6: UPF Packet Reception for the TCP scenario

we foresee the further development of our scheme to support proactive management of the number of UPFs based on the predicted network traffic.

REFERENCES

[1] B. Burns, J. Beda, K. Hightower, and L. Evenson, *Kubernetes: up and running*. " O'Reilly Media, Inc.", 2022.

[2] T. Rosado and J. Bernardino, "An overview of Openstack architecture," in *Proceedings of the 18th International Database Engineering & Applications Symposium*, 2014.

[3] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Open, programmable, and virtualized 5G networks: State-of-the-art and the road ahead," *Computer Networks*, vol. 182, 2020.

[4] M.-I. Csoma, B. Koné, R. Botez, I.-A. Ivanciu, A. Kora, and V. Dobrota, "Management and orchestration for network function virtualization: An open source MANO approach," in *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*. IEEE, 2020.

[5] P. Riti, D. Flynn, P. Riti, and D. Flynn, "The HashiCorp Ecosystem," *Beginning HCL Programming: Using Hashicorp Language for Automation and Configuration*, 2021.

[6] P. Amogh, G. Veeramachaneni, A. K. Rangisetti, B. R. Tamma, and A. A. Franklin, "A cloud native solution for dynamic auto scaling of MME in LTE," in *2017 IEEE PIMRC*. IEEE.

[7] O. Pozdniakova, D. Mažeika, and A. Cholomskis, "Adaptive resource provisioning and auto-scaling for cloud native software," in *Information and Software Technologies*, R. Damaševičius and G. Vasiljevienė, Eds. Cham: Springer International Publishing, 2018, pp. 113–129.

[8] H. Zhao, H. Lim, M. Hanif, and C. Lee, "Predictive container auto-scaling for cloud-native applications," in *2019 ICTC*. IEEE.

[9] N. Marie-Magdelaine and T. Ahmed, "Proactive Autoscaling for Cloud-Native Applications using Machine Learning," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020.

[10] D. Buchaca, J. L. Berral, C. Wang, and A. Youssef, "Proactive Container Auto-scaling for Cloud Native Machine Learning Services," in *IEEE 13th International Conference on Cloud Computing (CLOUD)*, 2020.

[11] S. Dutta, T. Taleb, and A. Ksentini, "QoE-aware elasticity support in cloud-native 5G systems," in *2016 IEEE International Conference on Communications (ICC)*, 2016.

[12] V. Passas, N. Makris, Y. Wang, A. Apostolaras, A. Mpatziakas, A. Drosou, T. Korakis, and D. Tzovaras, "Artificial Intelligence for network function autoscaling in a cloud-native 5G network," *Computers and Electrical Engineering*, vol. 103, 2022.

[13] A. Mudvari, N. Makris, and L. Tassiulas, "Ml-driven scaling of 5g cloud-native rans," in *IEEE GLOBECOM*, 2021.

[14] T. Subramanya, D. Harutyunyan, and R. Riggio, "Machine learning-driven service function chain placement and scaling in mec-enabled 5g networks," *Computer Networks*, vol. 166, 2020.

[15] V.-G. Nguyen, K.-J. Grinnemo, J. Taheri, J. Forsman, T. Le Duc, and A. Brunstrom, "On auto-scaling and load balancing for user-plane gateways in a softwarized 5g network," in *2021 17th International Conference on Network and Service Management (CNSM)*, 2021.

[16] I. Leyva-Pupo, C. Cervelló-Pastor, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic UPF placement and chaining reconfiguration in 5G networks," *Computer Networks*, vol. 215, p. 109200, 2022.

[17] H. T. Nguyen, T. Van Do, and C. Rotter, "Scaling UPF Instances in 5G/6G Core With Deep Reinforcement Learning," *IEEE Access*, vol. 9, 2021.

[18] C. Rotter and T. Van Do, "A Queueing Model for Threshold-Based Scaling of UPF Instances in 5G Core," *IEEE Access*, vol. 9, 2021.

[19] T. V. K. Buyakar, A. K. Rangisetti, A. A. Franklin, and B. R. Tamma, "Auto scaling of data plane vnfs in 5g networks," in *2017 13th International Conference on Network and Service Management (CNSM)*, 2017.

[20] N. Makris, C. Zarafetas, S. Kechagias, T. Korakis, I. Seskar, and L. Tassiulas, "Enabling open access to LTE network components; the NITOS testbed paradigm," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, 2015.

[21] S. Fdida, N. Makris, T. Korakis, R. Bruno, A. Passarella, P. Andreou, B. Belter, C. Crettaz, W. Dabbous, Y. Demchenko, and R. Knopp, "SLICES, a scientific instrument for the networking community," *Computer Communications*, vol. 193, pp. 189–203, 2022.

[22] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A flexible platform for 5G research," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, 2014.

[23] A. Güngör, "Ueransim," *GitHub Repository: https://github. com/aligungr/UERANSIM*, vol. 1, 2022.

[24] J. Turnbull, *Monitoring with Prometheus*. Turnbull Press, 2018.

[25] M. Chakraborty and A. P. Kundan, "Grafana," in *Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software*. Springer, 2021.

[26] G. Barlacchi, M. De Nadai, R. Larcher, A. Casella, C. Chitic, G. Torrisi, F. Antonelli, A. Vespignani, A. Pentland, and B. Lepri, "A multi-source dataset of urban life in the city of Milan and the Province of Trentino," *Scientific data*, vol. 2, no. 1, 2015.

Authorized licensed use limited to: University of Thessaly. Downloaded on February 24,2025 at 08:35:33 UTC from IEEE Xplore. Restrictions apply.