Microservices-Adaptive Software-Defined Load Balancing for 5G and Beyond Ecosystems

Sarantis Kalafatidis, *Member, IEEE,* and Lefteris Mamatas, *Member, IEEE*

***Abstract*—Although 5G networks achieve impressive improvements in terms of latency, data rates and network capacity, their evolution requires the synergy of networks, clouds and applications. For example, targeted challenging applications require efficient end-to-end performance, impacted by emerging technologies beyond radio, including: (i) Software-Defined Networks (SDNs) enabling programmability in the network environment from a centralized viewpoint; (ii) cloud orchestrators dynamically adapting server resource allocation to dynamic workloads; and (iii) microservice-based applications consisting of minimal, single-purpose service functions communicating with each other, individually implemented and scaled towards efficient resource allocation and fault-tolerance. In this context, we propose an SDN load balancing solution for a specific class of services consisting of microservices with heterogeneous but simpler network and server resource requirements, compared to the resource-demands of the whole service. Our proposal adapts to the estimated resource demands of individual microservices, balances resource utilization among cloud servers and improves service operation in terms of response times, throughput and fairness. We validate experimentally the proposed platform and its mechanisms with a particular 5G and beyond use-case with the above characteristics.**

***Index Terms*—Software-Defined Networks, Load Balancing, Microservices, Microservices Profiling, 5G and Beyond Networks**

I. Introduction

5G and beyond networks (5GB) are enabling new applications and network services with challenging, stringent requirements, including for ultra-low delays and high throughput, which are radically transforming vertical sectors, including manufacture, media & entertainment, automotive industry and energy. The main goals of 5GB include (i) exploiting higher frequency bands for improved throughput, e.g., millimeter-wave (mmWave) spectrum; (ii) improving spectrum usage efficiency through its intelligent management; and (iii) radically transforming the telecommunication system with the virtualization of physical network functions, reducing the capital expenditure (CAPEX) and improving the deployment and configuration flexibility.

Since challenging application performance concerns end-to-end (E2E) communication efficiency, 5G and beyond networks participate in flexible ecosystems integrating emerging technologies that maintain a systemic view of and adapt the network or cloud environment to dynamic changes in resource conditions or application requirements. Indicatively, for a 50ms E2E delay documented in 5G paper [1], radio access network (RAN) contributed to the 13%, while network and cloud application aspects to the 87% of delay. As a bottom line, communication and capacity improvements should be matched by particular transformations at higher layers of network stack and computation aspects, i.e., jointly implementing *awareness* and *adaptability* of radio, network and cloud-based applications.

Regarding radio, the cloudification of RAN processes, e.g., Open RAN [2], allows multiple independent instances of RAN functions to run on common hardware platforms. However, this technological approach requires efficient workload assignment and resource management improvements, since the virtualization of network devices may introduce performance issues, such as increased energy consumption. For example, Open RAN deployments can balance their user load among different radio cells, which is typically asymmetric. Furthermore, the resource requirements of highly-demanding 5GB applications (e.g., virtual reality) vary over time and call for dynamic adaptations of RAN resources towards achieving acceptable service performance levels.

On the networking front, telecommunication networks and relevant standards are gradually adopting novel networking paradigms, including Software-Defined Networks (SDNs). SDNs introduce a global view and programmability of network environments to implement network services that are bespoke to particular application requirements. For example, such requirements can be realized by SDN-based load balancing, bringing an efficient operation of Open RAN, cloud and network, as well as reduced energy expenditure. However, there is a need for a better awareness and adaptability of SDNs towards challenging services.

The cloud-based applications benefit from the significant advantages cloud computing brings, such as economies of scale to support resource-demanding applications, e.g., large numbers of users, as well as flexibility to adapt to dynamic changes. For example, virtual or physical resources may scale (up or down) according to monitored user demands, implementing horizontal and vertical elasticity policies, respectively.

Furthermore, applications should also consider the status of network environment, cloud resources and the challenging requirements. Along these lines, many of them adopt the microservices architectural paradigm. The latter breaks down complex monolithic applications or network services into a collection of simple, single-purpose communicating microservices, independently managed and scaled from microservices orchestrators, such as Kubernetes.

However, in the most complex landscape of 5G networks and beyond, applications or network services may consist of functions with heterogeneous resource requirements (e.g., some being CPU intensive, others network-sensitive, etc), increasing the complexity of resource management. We argue that these functions can be decomposed, based on specific

resource requirements, in stand-alone units of software packages (i.e., containers). In this context, we assume a specific class of applications or network services that organizes its functions in a way that each one of them is characterized by simpler resource-allocation demands, compared to those of the whole service. In practice, these functions are being split into containerized microservices, which we call *Resource-Organized Microservices (ROMs)*. Here, we investigate improved load balancing among ROMs (i.e, in terms of cloud resource efficiency and service performance), which estimates and adapts to the requirements of the latter.

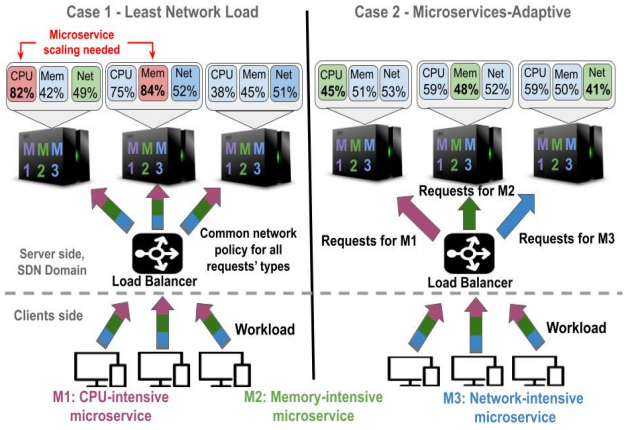

Fig. 1: Simple vs microservices-adaptive load balancing

In Fig. 1, we illustrate an example snapshot on the operation of alternative load balancing solutions with a relevant service. On its left side, we highlight the impact of a simple load balancing solution prioritizing servers with the least network load. This leads to an efficient balancing of network resources, but other resource types may be over-utilized, such as CPU and memory consumption in first and second servers, respectively. This strategy also leads to the deployment of additional microservices to handle the increased load, i.e., to a vertical elasticity event.

On the right side of Fig. 1, we depict a load balancing mechanism that is aware of the particular requirements of all microservices, the latest resource availability of cloud resources, as well as of network properties, e.g., network utilization and flow characteristics. This approach can achieve a better balance of all resource types, improved application performance, while avoiding unnecessary elasticity actions. Consequently, we argue that load balancing should be microservices-aware, in terms of being aware and adaptable to the dynamic resource requirements of all microservices implementing the service

Typical load balancing approaches do not fully address the above requirements, as we show in Table I, where we enlist indicative related works. For example, Kubernetes usually employs simple load balancing policies, including: (i) *round-robin*, balancing one-to-one requests among available servers, in a circular fashion; (ii) *least network load*, assigning requests to the server with the lower network utilization; and (iii) *shortest expected delay*, assigning an incoming job to the server with the lower estimated expected delay.

As we see in Table I, most relevant approaches balance load based on network and flow characteristics, while some of them take into account cloud resource availability [6], [7]. A few proposals consider individual microservices in the load balancing decisions, e.g., [8] proposes a chain-oriented load balancing algorithm to minimize microservice chains response time and [9] introduces a QoS-aware load balancing model considering links' capacity and delay between microservices. Due to the complexity of 5GB microservices-based applications, a sophisticated load balancing mechanism should consider most above aspects, i.e., server resources as well as network properties, including bandwidth availability and flow characteristics, in order to meet the unique requirements of each individual microservice.

TABLE I: Well-known load balancing approaches

Load balancing (LB) mechanism	**Micro-services**	**Cloud**	**Net.**	**Flow char.**
Kubernetes round-robin	-	-	-	-
Kubernetes least network load	-	-	✓	-
Kubernetes shortest expected delay	-	-	✓	-
Mahout [3]	-	-	✓	✓
FDALB [4]	-	-	✓	✓
Hedera [5]	-	-	✓	✓
Server Cluster LB [6]	-	✓	✓	-
e-STAB [7]	-	✓	✓	✓
LB accross microservices [8]	✓	-	-	-
LB for Interdependent IoT Microservices [9]	✓	-	✓	✓
MALB	✓	✓	✓	✓

Along these lines, we propose the Microservices-Adaptive Load Balancing (MALB) platform and corresponding mechanisms, characterized by the following novelties:

- *microservice-awareness* through online profiling that quantifies the level of importance of each resource type, based on simple prediction techniques.
- *microservice-level adaptability* of bespoke SDN-based load balancing policies, considering both cloud (i.e., CPU and memory) and network aspects (i.e., bandwidth allocation and flow sizes, in terms of duration time).
- *real experimentation* of our approach with an assumed use-case that consists of ROMs, demonstrating efficient and balanced server resource allocation, as well as improved application performance in terms of response times, throughput and fairness.

Although MALB can be used for network services as well, we are currently considering applications that can be potentially mapped to ROMs. Investigating network services in the context of Open RAN is in our future work plans. In this context, a presentation of a relevant motivating use-case follows next.

II. Motivating use-case scenario

We assume here a particular *virtual reality gaming use-case*, inspired by [11]. It consists of service functions with diverse characteristics, in terms of resource requirements, function execution times, and flow sizes, including: (i) video streaming

microservices being bandwidth-intensive and mainly producing long flows; (ii) user interactions' microservices, producing short flows and being characterized by low-latency and moderate CPU and bandwidth demands; and (iii) back-end microservices, including on user behavioral analysis, which are CPU-intensive, but with a fixed execution time to maintain a given latency, i.e., return the best outcome within a deadline.

Typical dynamic load balancing techniques may monitor a specific resource type, e.g., network utilization, and assign each new flow to the least overloaded entity (e.g., server, server cluster, edge cloud, etc.), in a reactive manner. This may work well for the video streaming microservices of the use-case, characterized by intense network communication, however, it may cause performance issues in other microservices that are sensitive to different types of resources (e.g., the back-end microservices) or accommodate flows with shorter sizes than the monitoring period (e.g., the user interactions' microservices).

Consequently, load balancing in the context of the considered use-case should be able to: (i) detect the diverse resource-requirements of microservices using profiling techniques; (ii) accommodate alternative load balancing policies, matching the particular requirements of each microservice type; and (iii) consider both cloud and network-related viewpoints.

In the sections that follow, we present our relevant proposal and highlight experimentally its capabilities based on the considered use-case.

III. Proposed System

Here, we present our Microservices-Adaptive Load Balancing (MALB) platform and its corresponding mechanisms, targeting: (i) the minimum and balanced resource utilization of both networking and cloud aspects; and (ii) the efficient operation of applications constituting of multiple types of microservices with diverse resource-demands. As we show in Fig. 2, MALB is built on top of an integrated environment consisting of an SDN-based network and a cloud hosting containerized microservices, in the form of ROMs. The three key components of our infrastructure with their operations are:

- the *Monitoring Subsystem* monitors the SDN-based network, cloud servers and microservices, providing a holistic awareness of the application and its environment.
- the *Microservices Profiler* dynamically predicts the level of microservices' impact on each resource type and determines the typical flow size characterizing the latter
- the *MALB Algorithm* balances the load to the particular requirements of considered microservice-types, exploiting the produced insights regarding the network and cloud environment hosting the microservice, as well as the unique resource-demands of the latter.

A detailed description of the above components follows.

A. Monitoring Subsystem

The *Monitoring Subsystem* enables microservice-awareness through online monitoring of microservices and their network and cloud server environment. It consists of two main components: (i) the *Monitoring Broker* collecting centrally all monitoring information; and (ii) the *Agents* handling the extraction of monitoring data from particular cloud servers. For simplicity, we currently employ *Agents* in the servers only, since we emulate SDN devices through Open vSwitch, i.e., the latter reports directly to the *Monitoring Broker*.

Monitoring Broker collects link utilization and flow sizes from SDN network, as well as CPU and memory utilization for both cloud servers and each particular microservice. The network statistics are being provided to the *Monitoring Broker* through the Open vSwitch API. This approach offloads the *SDN Controller* from all monitoring information, because the data are being processed from the *Microservice Profiler*, i.e., the former receives summarized information, only. Alternatively, the *SDN Controller* could lookup link utilization and flow size data from the switches directly, based on the OpenFlow protocol. The cloud statistics are being provided from the *Agents*, which periodically parse monitoring data from the docker stats application. *Monitoring Broker* communicates with the *Agents* through REST calls.

The monitoring aspect is critical and challenging on its own. Our next steps include the implementation of an *Agent* for real SDN switches interconnecting physical servers, utilizing a sophisticated monitoring infrastructure for large-scale service deployments, like [10], and carry out a study on the impact of monitoring period, tuning the trade-off between monitoring accuracy and involved control overhead, e.g., *SDN Controllers* are typically overloaded.

The monitoring data represent the input of *Microservices Profiler*, which description follows.

B. Microservices Profiler

The microservices-based adaptability of MALB is grounded on a profiling activity building the complete view of each microservice, in terms of particular resource demands and resource status of the surrounding network and cloud environment. This process is being handled by the *Microservices Profiler*, producing the following two-fold output for each microservice: its impact degree prediction on diverse resource types and a classification of its network flows into two categories, i.e., short and long. This output is being communicated to the *SDN Controller* via REST calls, while the historical values of monitoring data are being stored in a dataset.

The *Microservices Profiler* collects and processes the latest monitoring information communicated from the *Monitoring Subsystem* for each microservice and calculates a coefficient for each resource type (i.e., CPU, memory or network), reflecting the impact degree of the microservice on the particular resource to the total normalized resource consumption. Such values are placed on a window-based prediction mechanism that estimates the upcoming weights for each resource, after smoothing their evolution to remove outliers (e.g., CPU peaks). Although CPU peaks are important for load balancing [8], we decided to focus here on the average behavior of the system rather than on variance aspects, for simplicity.

Here, we assume that simple mechanisms exhibit a decent accuracy in predicting the resource demands of microservices, due to the single-purpose functionality of the latter. At this point of investigation, we employ window-based mechanisms

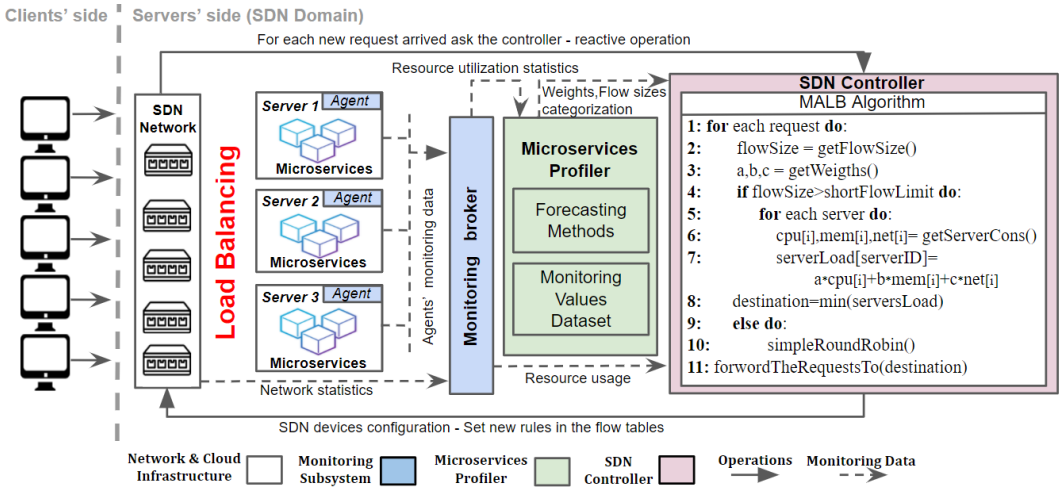

Fig. 2: The MALB Architecture

based on the Rolling Mean (RM) and the AutoRegressive Moving Average (ARMA). RM is biased towards the recent measurements and it smooths the short-term fluctuations, while ARMA assumes linear dependence for the data. Although we cannot claim achieving maximum resource prediction accuracy, such shortcomings were not critical in terms of validating the key message in the paper, i.e., the importance of microservice-awareness and microservice-level adaptability. MALB could ideally employ a bespoke prediction approach to each resource type, matching the structure of the corresponding time-series. The study of more sophisticated prediction mechanisms (e.g., also considering variance), starting from those being recently proposed for Open RAN, is complex enough to deserve an independent study.

The flow sizes' classification determines which microservices produce short and which long flows, in terms of flow duration times, assuming that ROMs produce mostly either short or long flows. In practice, the *Profiler* groups the flow durations per microservice type and tags a microservice as a short flow one, when its flows last less than a specific threshold, and as a long, when they last more. This strategy allows us to handle the case that a flow may be completed, before a load balancing schema is able to determine the status of microservices or servers, in terms of resource demands or availability, respectively.

Here, we argue that simpler policies with low complexity, including the Round Robin load balancing schema, suffice for microservices producing short flows, since there is an insignificant risk of being accumulated in a server, due to their short duration. We note that short flows benefit greatly from an efficient resource allocation of long flows sharing the same resources. In our case, MALB produces an accurate status of the system every 3 seconds, due to performance constraints of the open-source facilities we employ, i.e., Floodlight controller and docker stats tool. For example, a control plane latency that fluctuates by tens of milliseconds may create monitoring accuracy issues, in the case of an 1-sec interval.

A brief description of MALB algorithm follows next.

C. MALB Algorithm

The microservice-level adaptability of our load balancing platform is realized through *MALB Algorithm*, an *SDN Controller* module that defines the appropriate destination (i.e., server, in our case) to forward each incoming request. In practice, MALB aims to balance the load among the servers and bring uniformity in the utilization of network and server resources, i.e., avoiding both network congestion and server overloading.

As shown in Fig. 2, MALB receives the outcome of *Microservices Profiler* expressing the predicted weighted resource demands of each microservice, as well as a categorization of the typical flows' size of the latter. Furthermore, it receives frequent snapshots of the resource utilization of all servers. At this point of investigation, MALB determines the type of microservice based on its layer-4 ports.

MALB applies the Round Robin load balancing schema for the short flows (e.g, the user interactions' microservices), which is a simple and efficient strategy, in our experience. In the case of long flows (e.g., the streaming microservices), requests are being forwarded to the server with the lower estimated load, i.e., combining the weighted resource demands of the particular microservice with the online resource availability of the servers. The load of each server is expressed as the weighted sum of its estimated CPU, memory and network utilization percentage. The respective weights are α, β, and γ, reflecting the importance of each resource type in the particular microservice.

SDN Controller calculates the load of each server and the particular microservice and forwards upcoming requests to the servers with the lowest load, i.e., through applying the corresponding flow table rules to the SDN devices.

IV. Performance Evaluation

In this section, we provide our experimentation analysis on the i) evaluation of microservice profiling process of MALB, while backing relevant design choices; and ii) validation of

MALB proposal, in terms of efficient service performance and cloud resource utilization.

Since our solution brings together SDN load balancing with containerized microservices, we conducted our experiments on Containernet [13], which supports both SDNs and Docker containers. The studied load balancing mechanisms correspond to relevant Kubernetes policies, but are implemented in the Floodlight controller [14]. We simulated the workload, i.e, characterized by the number of requests from clients, through the Apache JMeter. We utilize a test-bed environment consisting of two physical servers and one switch. The first server hosts the client emulation facilities and the second the services.

The considered ROMs are aligned to the proposed use-case, which are: i) a video streaming service delivering videos of different sizes, built using *VLC* server; ii) a web service representing the users' interactions functionality based on *Flask* web framework; and iii) a back-end service (developed through PHP using Apache server) which executes demanding calculations with configurable durations. Each type of microservices is characterized by particular resource requirements, i.e., the video streaming service is bandwidth-intensive, the back-end service is CPU-intensive and the user interaction service utilizes both processing and network resources, at a moderate level. The services are configurable to generate flows with different sizes. We do not vary the number of deployed microservices, since we focus on optimizing the system between elasticity events. We have assigned isolated physical resources to four different sets of microservices, i.e., representing four physical servers.

In our experiments, we consider an on-line game event with a 30-min duration, where the workload is characterized by three different 10-min phases, which descriptions follow: (i) *gradually increasing*, linearly increasing the load over a fixed time period, resembling gamers entering the event; (ii) *stable*, having a fixed workload, assuming a particular number of active players; and (iii) *gradually decreasing*, linearly reducing the workload over the same fixed period, i.e., users are leaving the system. All requests have been conducted asynchronously, i.e, using separate threads.

To realize the objectives of our experimentation analysis and highlight the novel aspects of MALB proposal, we devised the following scenarios: (i) the *microservices profiling*, evaluating the corresponding mechanisms, while motivating and supporting technically our approach to predict the requirements of microservices, in terms of CPU, memory, and network resource-demands; and (ii) the *MALB platform evaluation* scenario that validates the novel features of our proposal.

A. Scenario 1: Microservices Profiling

Here, we validate the dynamic microservices profiling process of MALB and investigate relevant design choices and configurations. Due to the dynamic nature of 5G and beyond services, MALB estimates, for each forthcoming time-instance of the monitoring period, which is 3 sec in our case, the CPU, memory and network utilization of each microservice. This process defines the weight values of MALB algorithm, enabling microservice-aware load balancing. In practice, we evaluate the accuracy of RM and ARMA with different window sizes, in terms of producing efficient weight values.

Here, we describe a *Microservices Profiling* example in which the average utilization values of CPU, memory and network in the full duration of an experiment with 1 video streaming request every 2 seconds are 0.51, 19.26, 22.87, respectively. According to these metrics, the average calculated weights α, β, and γ of the profiling process are 0.012, 0.452 and 0.536. Indicatively, γ value 0.536 reflects the importance of network (22.87) in the total normalized resource allocation 0.51+19.26+22.87. This means that this microservice mainly requires network resources. The idea of our dynamic profiling feature is to carry out a similar process at each time-instance, while considering predicted resource allocations, instead of measured.

Next, we conducted experiments with separate deployments of (i) the three considered microservice types; (ii) microservices profiling with both RM and ARMA having window sizes of 2, 5, 10 and 20, 50, 100, respectively (since the dataset is non-stationary, we avoided using very large window sizes); and (ii) the gradual increasing, stable and gradual decreasing user patterns, i.e., linearly increasing from 1 to 3 requests/sec, having 3 requests/sec, and linearly decreasing from 3 to 1 request/sec, respectively.

According to our results based on Mean Squared Error (MSE) measurements, both RM and ARMA exhibit a descent accuracy (i.e., with MSE ranging most of the times between 0 and 8), while being especially accurate with under-utilized resources. This can be justified by the single-purpose functionality of microservices, i.e., they have a more expectable resource utilization, in contrast to monolithic applications. Smaller window sizes seem to favor RM, while larger ones ARMA. However, it is more challenging to predict CPU and network utilization, characterized by a number of short-term peaks, especially for ARMA with its linear properties. For example, CPU peaks are more frequent with the stable number of clients, since such run utilizes more clients on average, i.e., a higher mean value produces a lower variance, causing a high MSE in the case of back-end service and CPU utilization metric. RM with window 2 either achieves the best accuracy or it is marginally outperformed. This is expected, since it follows the short-term dynamics of the resources.

For simplicity, we selected to use Rolling Mean with window size 10 in our experiments, balancing its accuracy with a smoothness level that follows the average behavior of the system. Such strategy appeared effective in our results, which could be further improved with a more accurate relevant prediction mechanism, ideally considering both mean and variance aspects.

B. Scenario 2: MALB Platform Evaluation

In this scenario, we assess resource-allocation efficiency and impact on service performance of *MALB* in contrast to the *Simple Round Robin (SRR)* and *Least Network Load (LN)* strategies, both being widely used in SDN environments and microservice management platforms, such as Kubernetes.

Our goal here is to evaluate the impact of *MALB* from both provider's (i.e., in terms of server and network resource

TABLE II: Load balancing level among the servers

	SRR				LN				MALB			
Metric	**CPU**	**Memory**	**Network**	**Energy**	**CPU**	**Memory**	**Network**	**Energy**	**CPU**	**Memory**	**Network**	**Energy**
STDEV	17.3	2.1	9.3	27.1	15.0	2.0	9.45	24.7	11.2	1.9	10.1	18.3
Range	44.2	5.2	23.6	70.1	38.4	5.1	24.3	62.8	29.2	5.1	25.1	47.7

utilization as well as energy efficiency) and clients' side (i.e., in terms of request response time, throughput and fairness). Our SDN environment stress tests above mechanisms with the communication of clients with four servers, each one of them hosting all considered microservices. All provided figures and tables illustrate the average values between 10 runs. We did not have significant deviations among the runs, e.g., standard deviation ranged between 0.03 and 0.47, for the average measurements of CPU, memory and network resources.

Here, we emulated: (i) light (i.e., 1MB) and medium-sized (i.e., 10MB) file requests, for the *user-interaction* microservices; (ii) medium (i.e., 10 sec flow size, 15% of CPU) and heavy-sized (i.e., 30 sec flow size, 30-35% of CPU) communication, for the *back-end* microservices; and (iii) a live video broadcasting, i.e., without a specific flow duration, for the *video streaming*. The clients to all above microservices are being deployed, according to the assumed on-line game event.

We estimated the total energy consumption of each server using the non-linear model for CPU introduced in paper [15], as well as a simple linear equation for memory and network (i.e., ranging for both between a baseline and a maximum power consumed). Regarding the non-linear model, we used the coefficient (calibration parameter) $r = 1.4$. We have also considered $160W$, $36W$, $12W$, $25W$, $25W$, and $10W$ as the maximum power drawn (P_{busy}) of CPU (4 processors), Memory (4 DIMMs), disk, pci slots, motherboard and fan, respectively. The baseline power (i.e., $P_{baseline}$) was defined as the 40% of P_{busy}, for all types of resources.

Table II enlists the average of all standard deviations (STDEV) and ranges (i.e., max - min) among the four servers, for each timestamp, i.e., quantifying their load balancing level. We observe that *MALB* demonstrates a significant higher load balancing level, compared to the other two approaches, of the CPU resources (e.g., 34%, and 24% lower Range than *LN* and *SRR*, respectively) and almost equally to the lower Range of *LN* for memory and network. Since CPU is the most energy-consuming resource, *MALB* also appears energy efficient (e.g., 32%, and 24% lower range than *LN* and *SRR*, respectively). This result highlights the performance advantages of *MALB* in terms of load balancing, due to its centralized view of resources and resource-demands of microservices.

In Fig. 3, we illustrate the mechanisms' performance in terms of average and worst-case resource consumption of CPU and network resources, over all servers and at the full duration of experiments. We quantify the worst-case resource consumption as the average of the 30 higher values of the particular metric, i.e., corresponding to the 3% of total measurements. The green bar represents the average of the worst cases among all servers and the purple one the equivalent metric of the most overloaded server. We also denote an elasticity threshold, when one of the metrics exceeds the value 70%.

Although all mechanisms exhibit a similar performance

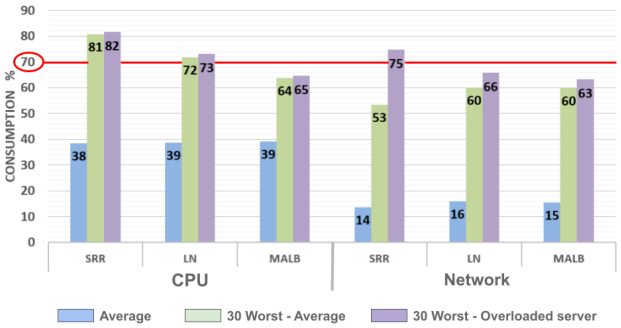

Fig. 3: MALB Provider-Side Evaluation

in terms of average consumption of CPU and network, i.e., attributed to the under-stressed cloud resources, there are cases of high CPU utilization for both *SRR* and *LN*, triggering elasticity events that could be avoided with a better balancing of resources. For example, the worst CPU allocation of *MALB* is 16.9% lower than those of *SRR*. Also, in the case of *MALB*, the overloaded server metrics are close to the average worst cases, highlighting the fairness achieved in the load balancing among the servers.

Table III illustrates our evaluation results from the clients' point of view. It enlists the type of microservices (*Microservice Type*), the total number of requests (*Requests*) for each particular run, as well as the considered metric (*Metric*) and the corresponding values. We document *Requests' Completion Time (RT, ms)* for all microservice types, besides video that does not have a fixed execution time. The performance of the latter is measured in terms of application *Throughput (TP, kB/sec)*. We enlist the average and standard deviation of each corresponding metric over the indicated number of requests. Consequently, the latter metric is an indication of the fairness level among the microservices of the same type.

In general, Table III results verify the observations of Fig. 3 and Table II, showing that efficient load balancing affects the performance of services. On the one hand, we observe from the average measurements that, *MALB* achieves the best *RT* performance of both short-flow and long-flow requests e.g., in the case of *user-interaction medium requests* workload, *MALB* completes requests 107 ms and 109 ms sooner than *SRR* and *LN*, respectively. This also underlines that microservice-level adaptability of *MALB* is an effective strategy. Regarding *video streaming* microservices, *LN* slightly outperforms *MALB* in terms of throughput (e.g., 51 kB/sec), since it is a bandwidth-intensive application. This outcome could be improved with a more accurate prediction mechanism for network utilization.

On the other hand, the STDEV measurements reveal significant advantages of *MALB* in terms of microservice perfor-

TABLE III: MALB Client-Side Evaluation

Microservice Type	Requests	Metric	SRR AVG	LN AVG	MALB AVG	SRR STDEV	LN STDEV	MALB STDEV
User Int. Light	1600	RT	448	460	429	313	256	201
User Int. Medium	1500	RT	1175	1177	1068	552	493	336
Back-end Medium	900	RT	10226	10224	10224	198	125	89
Back-end Heavy	550	RT	30232	30230	30223	142	118	86
Video Stream	650	TP	1560	1818	1767	36	40	43

mance fluctuation and fair operation among the microservices of the same type, aspects being crucial for 5G and beyond services. *MALB* achieves up to 55% and 31% STDEV reduction contrasted to *SRR* and *LN*, respectively.

Our results can be summarized as follows. *MALB* improves resource utilization and application performance for the considered resource-organized microservices. We also confirm main *MALB* design directions: (i) to incorporate a simple load balancing policy, such as *SRR*, for services generating short flows; and (ii) to handle long flows with dynamic load balancing equipped with online resource monitoring of both network and compute resources, as well as adaptability to the particular resource-demands of each microservice type, driven by dynamic microservice profiling.

V. Conclusions

This paper presented MALB, a novel SDN-based load balancing facility that focuses on a special case of 5G and beyond services, i.e., services consisting of microservices with heterogeneous, but simpler resource-demands compared to the service they constitute. MALB adapts its load balancing process to the particular requirements of microservices, based on dynamic microservice profiling. Our experimental results revealed the significant performance advantages of MALB, in terms of resource utilization of cloud environment as well as the response times, application-layer throughput and fairness of the microservices, further supporting the significant radio performance and capacity advantages of 5G and beyond ecosystems. In the evolution of this work, we consider the study of challenging network services and sophisticated profiling mechanisms in the context of Open RAN.

Acknowledgment

The paper is supported by the "GSRI FUNDING FOR THE YEAR 2019 (Award for the participation in competitive E.U. projects)", Novel Enablers for Cloud Slicing - NECOS,GA No 777067, HORIZON 2020 - JOINT ACTION EU - BRAZIL, H2020-EUB-2017, Ministry of Development and Investments – General Secretariat for Research and Innovation. It is also co-funded by Greece and the European Union (European Social Fund-ESF) through the Operational Programme "Human Resources Development, Education and Lifelong Learning" in the context of action "Enhancing Human Resources Research Potential by undertaking a Doctoral Research", sub-action 2: "IKY Scholarship Programme for PhD candidates in Greek Universities".

References

[1] J. F. Santos *et al.*, "Breaking Down Network Slicing: Hierarchical Orchestration of End-to-End Networks.", *IEEE COMMAG*, vol. 58, no. 10, pp. 16-22, 2020.

[2] A. Garcia-Saavedra, and X. Costa-Perez, "O-RAN: Disrupting the virtualized RAN ecosystem." *IEEE Communications Standards Magazine*, 2021.

[3] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection.", *IEEE INFOCOM*, IEEE, 2011.

[4] S. Wang *et al.*, "FDALB: Flow distribution aware load balancing for datacenter networks.", *IEEE/ACM IWQoS*, IEEE, 2016.

[5] M. Al-Fares *et al.*, "Hedera: dynamic flow scheduling for data center networks.", *NSDI*, vol. 10, no. 8, pp. 89-92, 2010.

[6] Q. Du, and H. Zhuang, "OpenFlow-based dynamic server cluster load balancing with measurement support.", *Journal of Communications*, vol. 10, no. 8, pp. 16-21, 2015.

[7] D. Kliazovich *et al.*, "e-STAB: Energy-efficient scheduling for cloud computing applications with traffic load balancing.", *IEEE/ACM GreenCom*, IEEE, 2013.

[8] Y. Niu, F. Liu, and Z. Li, "Load balancing across microservices.", *IEEE INFOCOM*, IEEE, 2018.

[9] R. Yu *et al.*, "Load balancing for interdependent IoT microservices.", *IEEE INFOCOM*, IEEE, 2019.

[10] S. Clayman, A. Galis, and L. Mamatas, "Monitoring virtual networks with lattice", *IEEE/IFIP NOMS*, IEEE, 2010.

[11] M. Elbamby *et al.* "Toward low-latency and ultra-reliable virtual reality.", *IEEE Network*, vol. 32, no. 2, pp. 78-84, 2018.

[12] M. Giordani *et al.*, "Toward 6G networks: Use cases and technologies.", *IEEE COMMAG*, vol. 58, no. 3, pp. 55-61, 2020.

[13] "Containernet." [Online]. Available: https://containernet.github.io/.

[14] "Floodlight Controller - Project Floodlight." [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview.

[15] X. Fan, W.D. Weber, and L.A. Barroso. "Power provisioning for a warehouse-sized computer." *ACM SIGARCH computer architecture news*, 35(2), 13-23, 2007.

Sarantis Kalafatidis pursues his Ph.D. at the University of Macedonia, Greece, in the areas of Software-Defined Networking and Cloud Computing. He holds a MSc Degree in Applied Informatics from the same University. He investigates the research problems of efficient resource allocation and load balancing for 5G and beyond networks in both data center and Smart-City network environments. He participates in various international research projects, such as NECOS (H2020) and FED4FIRE+ OC9 (H2020).

Lefteris Mamatas is an Associate Professor at the Department of Applied Informatics, University of Macedonia, Greece. He leads the Softwarized & Wireless Networks Research Group (http://swn.uom.gr) in the same University. His research interests lie in the areas of Software-Defined Networks, Internet of Things, 5G Networks, and Multi-Access Edge Computing. He participated in many international research projects, such as NECOS (H2020), FED4FIRE+ OC4 (H2020), WiSHFUL OC2 (H2020), MONROE OC2 (H2020), Dolfin (FP7), UniverSELF (FP7), and Extending Internet into Space (ESA). He has published more than 80 papers in international journals and conferences.