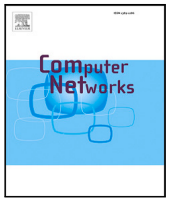
Computer Networks 254 (2024) 110750

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Softwarized and containerized microservices-based network management analysis with MSN

Sisay Tadesse Arzo [a], Domenico Scotece [b,*], Riccardo Bassoli [c], Michael Devetsikiotis [a], Luca Foschini [b], Frank H.P. Fitzek [c,d]

[a] Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, USA
[b] Department of Computer Science, University of Bologna, Bologna, Italy
[c] Deutsche Telekom Chair of Communication Networks, Institute of Communication Technology, Faculty of Electrical and Computer Engineering, Technische Universität Dresden, Dresden, Germany
[d] Centre for Tactile Internet with Human-in-the-Loop (CeTI), Cluster of Excellence, Dresden, Germany

ARTICLE INFO

Keywords:
Microservice
Containerization
Cloudification
5G
Beyond 5G
6G

ABSTRACT

Microservice architecture is a service-oriented paradigm that enables the decomposition of cumbersome monolithic-based software systems. Using microservice design principles, it is possible to develop flexible, scalable, reusable, and loosely coupled software that could be containerized and deployed in a distributed edge/cloud environment. The flexible deployment of microservices in an edge environment increases system performance in terms due to dynamic service function placement and chaining possibly resulting in latency reduction, fault tolerance, scalability, efficient resource utilization, cost reduction, and energy consumption reduction. On the other hand, virtualization and containerization of microservices add processing and communication overheads. Therefore, to evaluate end-to-end microservices-based system performance, we need to have an end-to-end mathematical formulation of the overall microservice-based network system. Incorporating the virtualization overhead, here we provide end-to-end mathematical formulation considering system parameters: latency, throughput, computational resource usage, and energy consumption. We then evaluate the formulation in a testbed environment with the Microservice-based SDN (MSN) framework that decomposes the Software-defined Networking (SDN) controller in microservices with Docker Container. The final result validates the presented mathematical modeling of the system's dynamic behavior which can be used to design a microservice-based system.

1. Introduction

Microservice is a Service Oriented Architecture (SOA) that is being used and deployed in the industry and research community [1,2]. It is a software development paradigm that decomposes a cumbersome monolithic system into smaller manageable loosely-coupled services. Moreover, microservices-based decomposition of functionalities is adopted by companies managing big cloud services and infrastructures [3,4].

As we have presented in our previous research work, there are two competing paradigms for decomposing monolithic-based systems: microservices-based architecture and multi-agents-based architecture [1]. Both architectures break monolithic software systems into small services, whose instances may run independently in virtual environments or containers. As outlined in [1], the main difference between microservices and agents is that the former act reactively while the latter act both reactively and proactively.

Along with the service-oriented technology, Network Function Virtualisation (NFV) [5] and Software-defined Networking (SDN) [6] are the two most promising paradigms for network softwarization, decomposition, and orchestration in edge/cloud-based distributed environment. This paves the way for network services cloudification [7] that allows deploying network services in the edge/cloud environment as virtual entities such as Virtual Machine (VM) and containers. Furthermore, it gives network management system flexibility in terms of dynamic service scaling, dynamic deployment, dynamic instantiation, and decoupling for spatial and temporal flexible deployment. The decoupled microservices-based network functions could be deployed in a cloud or edge/fog using containers such as Docker Container.

In a softwarized networking environment, network devices could host multiple virtual entities like containers. In particular, these containers could be network functionalities developed as microservices

* Corresponding author.
E-mail address: domenico.scotece@unibo.it (D. Scotece).

https://doi.org/10.1016/j.comnet.2024.110750
Received 2 October 2023; Received in revised form 17 June 2024; Accepted 23 August 2024
Available online 26 August 2024

or multi-agents. Even end-user devices could host multiple containers. Since the networking technological trend is in the direction of service decoupling and containerization, we need a comprehensive study of microservices-based and containerized network systems. Moreover, future networks such as beyond 5G and 6G are expected to have more modular and distributed deployments, employing in-network intelligence of autonomous and atomic functions for network management, orchestration, and operations [8]. In particular, network functions are going to be deployed mainly as decomposed, softwarized, and containerized functions. New requirements of future networks like 6G have been preliminarily defined through some projects' deliverables like the ones from the EU flagship Hexa-X [9]. This documents [9,10] has started the characterization of the future 6G Key Performance Indicators (KPIs). It mainly suggested full network softwarization with microservice-based and agent-based implementation and in-network intelligence play a key role [8]. However, the existing works mostly focus on microservices' placement or resource allocation in an edge data center or load-balancing in microservices chain [11–13].

Therefore, to analyze and design the microservices-based communication system, an analytical framework is required. Moreover, the microservices could be deployed in geographically distributed edge data centers, which may significantly affect the end-to-end latency, throughput, energy, and resource usage. To calculate these parameters, a more accurate mathematical model is required that considers the containerization of microservices. In [14], an end-to-end mathematical model of the overall Cloud Radio Access Network (C-RAN), also considering edge data centers, is presented, using six representative parameters for the characterization and performance evaluation. However, the article only considers generalized Virtual Network Functions (VNFs) in the data center for the service function chain. The end-to-end latency is affected when the microservices chain becomes longer as each deployed microservice uses containers that incur an additional delay due to virtualization and containerization. To the best of the authors' knowledge, there is no work that considers the characterization of the overhead delay in a microservice-based containerized network system deployment providing an accurate description with an end-to-end realistic system model. We used the Microservice-based SDN (MSN) framework which is our implementation of a decomposed SDN controller functions designing the subfunctions as independent containerized units that are chained to create a decoupled and flexible SDN system [15].

Therefore, this article deals with microservices in containers, deployed in the edge and/or the cloud. We provide a mathematical formulation considering multiple parameters, such as end-to-end latency, throughput, and energy consumption. The contribution of this article can be summarized as follows:

- comprehensive review of microservices-based services and networking functions;
- comprehensive mathematical model for containerized microservices used as building blocks for network management systems;
- performance evaluation of the proposed mathematical using the MSN framework. We have also evaluated nested dockers incorporating docker inside docker. We have evaluated the latency and validated the formulation for added latency per a layer of docker.

Finally, this work serves as a valuable reference for both mathematical and experimental evaluations of next-generation networks that utilize microservices chains. In particular, the methodologies and findings presented in this paper offer practical tools for assessing contemporary system deployments. This includes environments leveraging Docker Containers, VMs, and Kubernetes orchestration. By providing comprehensive evaluation techniques, this research can guide the development and optimization of modern, scalable, and efficient network architectures based on microservices.

The rest of the article has the following structure: Section 2 organized and presented the literature survey. Section 3 defines and describes the end-to-end system model for analyzing the problem of microservice-based management. The mathematical formulation of the overall system that considers latency, throughput, resource utilization, and energy consumption is presented in Section 4 and Section 5. The performance results of the mathematical formulation based on both simulation and testbed are presented in Section 6. Final concluding remarks are presented in Section 7.

2. Related work

This section reviews the existing work on microservices-based systems, focusing on networking functions used in building network management systems.

2.1. Microservices modeling

The authors in [16] presented a mathematical model for microservices-based systems considering smart manufacturing as a target application. The work focuses on the placement methods for microservices considering latency in edge and cloud collaboration using an accurate data-driven end-to-end latency estimation. The article in [17] discusses about microservices scheduling to optimize end-to-end delay, average price, satisfaction level, Energy Consumption Rate (ECR), failure rate, and network throughput. The authors claim that the proposed scheduling algorithm is able to improve the performance in the aforementioned evaluation matrix. Even if the work is comprehensive in considering a number of relevant matrices, the article did not consider the containerized environment that imposes additional virtualization overhead delay. A performance modeling for a microservices platform is studied and evaluated in [18]. They used Amazon EC2 cloud to build the microservices platform. Using the platform, they developed a performance mode to make analysis and plan the required microservices capacity scaling.

Finally, in [2] a decomposition architecture for microservices is introduced in a formal language called "Sarch", which enables to describe software using modules and sub-modules. The work enables microservices architecture by utilizing component and connector-based architectural decomposition. The evaluation does not consider the decomposition and containerization challenges in terms of latency and energy consumption.

2.2. Modeling of functions in a virtual environment

With virtualization, it is necessary to model the virtual functions using mathematical formulations to evaluate for system parameters such as latency, reliability, resource utilization, energy consumption, and Quality-of-Service (QoS) parameters. Considering latency the authors in [19] presented an interesting analysis of the virtual environment. The authors performed an in-depth experimental evaluation of the impact of virtualization on processing and communication. They have also presented delay breakdown of the virtualization components of the hypervisor. However, the study is limited to the delay of a single function.

The authors in [1] proposed agents as smallest and autonomous building block for software design. The agents are atomic units with complete functionality in delivering a given service without external intervention and they have communication capability to speak with other agents. The authors presented a mathematical model for agent-based network management systems. The agents could be considered VNFs that are deployed into containers in an edge/cloud environment. The paper only focused on multi-agent systems and mainly focused on the theoretical development of the overall system.

2.3. Microservice resource allocation

The authors, in [11] proposed a resource allocation technique for microservices for efficient utilization of resources in an edge and cloud environment. They used an economic approach to allocate and re-allocate resources from a microservice across multi-tenants based on online auction mechanism incentives microservices. A supervised machine learning-based autoscaling of microservices is presented in [20]. The authors provided a hybrid dynamic microservice scaling targeting elastic applications. The model is an auto-scaling system that responds reactively or proactively depending on the workload or resource usage dynamically which could be horizontal scaling; adding or removing applications instances or vertical scaling computing resources like CPU or memory as per the workload demand.

2.4. Microservice scheduling

A heuristic-based scheduling of incoming services for microservices-based edge computing is presented in [21] which considers latency optimization in the scheduling process. The author developed a scheduler named FLAVOUR by formulating services latency reduction problem. The stochastic latency problem minimization considers microservice completion time constraints and network stability. Solving the formulated problem, they derived an optimized latency scheduler.

Another interesting microservices scheduler in heterogeneous edge/cloud environments is presented in [22]. The authors used a mathematical formulation to describe the architecture. Using realistic environments and examples, they evaluated the proposed scheduling algorithm and they claim that frequently used legacy scheduling methods could perform well in a microservices-based environment.

The work in [23] presented a scheduling algorithm to improve the fault tolerance of edge computing platforms. The work introduced a heuristic services binding algorithm using cache-based edge devices. The problem is formulated as constrained optimization services based on network graphs and composition. Moreover, the author provided heuristic solutions for the function using action values and graph-based state. The system performance and robustness is balanced by the fitting combination.

2.5. Microservice and container placement

Some works have been started focusing on container placement and migration considering the scheduling models [24]. The authors showed the algorithms and framework used to build the scheduling models based on a graph model. Also, for solving the multi-objective optimization problem, a heuristic approach is utilized, which enables the evaluation of the final sub-optimal outcome faster. The scheduling of containers could be at the cloud or in a distributed local edge to satisfy the QoS requirements.

Not only network functions but also the newly designed functions, such as IoT protocol translator [25], new QoS monitoring functions, etc, that could also be developed as microservices or multi-agents. They could be dynamically orchestrated and deployed to provide the required functionality. The work in [26] developed a QoS-aware model for VNF placement and provisioning that guarantees the latency requirements of the service chains. However, the authors did not consider containerization.

3. Graphic modeling of a softwarized system

In order to analyze and study the characteristics of the end-to-end softwarized network continuum of future generation networks, a proper theoretical model is necessary. This model requires the description and characterization of the hardware and software components of the network. An analytical model with these properties was first published in [14] to study virtual network functions' placement.

This article now enhances the characteristics of that original model for the study of microservices-based networks. The mathematical model consists of a multi-layer hypergraph as depicted in Fig. 1. As can be seen from the graph, we assume the network infrastructure to be described by several layers, which represent the abstractions of the resources (communication, computing, and storage) available. These layers are the virtual networks sliced from the actual network infrastructure. These slices (layers) represent the end-to-end softwarized network continuum of communication. The softwarized continuum embraces the resources at the end communicating devices, the Radio Access Network (RAN), the edge network and data centers, and the cloud. They can also be mapped to different layers of the hypergraph. As already mentioned in the literature [14], this mathematical structure can provide a more accurate description of how communicating services could connect via multiple data centers or the cloud to build a microservice chain to perform a complex communication operation, task, functionality, etc.

The end-to-end system resource is considered a multi-layer hypergraph. The graph is defined as $H(X_m, L_m, S, L)$, where X_m represents the set of node elements, that are abstracting and slicing the resources of each network node of the infrastructure. L_m is the set of edge elements, which consists of the links that connect the network nodes. Next, S represents the set of network nodes in the actual network infrastructure. Finally, $Ly = L_1, \ldots, L_a$ are the sets of layers according to the number a of aspects. The aspects indicated the number of elementary layers that compose a specific layer.

Fig. 1 depicts the general visual representation of the hypergraph H for mapping the end-to-end network infrastructure and communication. The RAN is mathematically described by a random distribution of points and the coverage is modeled using the specific Voronoi tessellation [27]. These points are then connected to other layers, which can model the edge and core network and the Internet. Within these network areas, data centers can be further layers since they are networks themselves. Next, each computing node of the network can run several virtual environments that can host several other virtual entities and processes respectively. These virtual networks and interconnections within the actual computing network nodes are represented by other layers and elementary layers. All these entities, softwarized or hardware-based, own a set of parameters that describes the resources used. This becomes important for the performance evaluation and the characterization of the KPIs in such a complex scenario.

4. Microservice based system modeling

In this section, we formulate the overall microservices-based workload processing system considering the following aspects. First, we defined a mathematical model for services (i.e., tasks or workload) definition; second, we characterized the workload arrival modeling in the edge and cloud environments; third, we modeled the incoming throughput for services; finally, we defined a mathematical model for microservices-based function chains.

4.1. Service definition and mathematical representation

First, let us define some terminologies. To differentiate between the two types of services, incoming user workload to be processed at the data center by functions(services), we refer to the user services as incoming user workload, whereas the processing function as functions of microservices. In the data center, various types of services could be deployed as indicated in the above section. Here we focus on network services or functions that are designed as microservices including authentication, routing function, QoS monitoring function, security function, mobility management, serving/packet gateway, etc.

Now let us present the definition and mathematical model for services (workload/tasks). A *user's service workload* (W_i) is represented using the ith commodity flow with a quadruple constraint parameters $(Sr_i, \beta_i, \gamma_i, D_i)$, where $Sr_i \in W$ is the source, where W is the set of

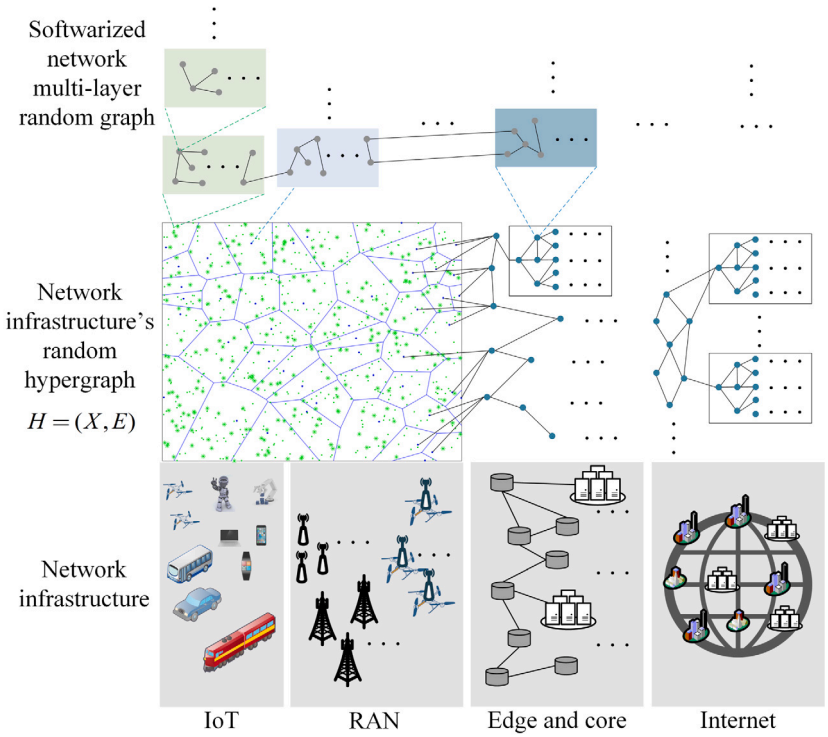


Fig. 1. System Scenario and Model.

Table 1
Table for Notations.

Notation	Description
W_i	User's service workload, represented as the i-th commodity flow
$(S_{ri}, \beta_i, \gamma_i, D_i)$	Quadruple constraint parameters of the service workload
$S_{ri} \in W$	Source of the i-th commodity, where W is the set of sources
$\beta_i \in \beta$	Sink of the i-th commodity, where β is the set of sinks
γ_i	Indicator function $1_K : K \to \{0,1\}$, which indicates a value of 1 if the commodity k_i is in the subset $K \subseteq K$ of commodities with elastic demand, and 0 if inelastic
D_i	Demand set, defining service requirement parameters such as priority, workload demand, latency, reliability, and throughput
K	Set of commodities, where $K = \{k_i\}$ for $i = 1, \ldots, m$

sources, $\beta_i \in \beta$ is the sink where β is the set of sinks, and γ_i is the indicator function $1_{\mathcal{K}} : K \to \{0,1\}$, which indicates a value of 1 if the commodity k_i in the subset $\mathcal{K} \in K$ of commodities having elastic demand set and value 0 indicates inelastic demand. D_i is *demand set*, defining the service requirements parameters such as service priority indicator, user's workload demand, latency, reliability, and throughput. For an elastic commodity $D_i = \{\alpha, L_{ij}, [\tilde{\tau}_{min}, \tilde{\tau}_{max}], [\tilde{\rho}_{min}, \tilde{\rho}_{max}], [\tilde{Thr}_{min}, \tilde{Thr}_{max}]\}$. The three service parameters with ranges indicate services Type 2 and Type 3, otherwise inelastic cases indicate service Type 1. The set becomes $D_i = \{\tilde{c}, \tilde{\tau}, \tilde{\rho}, \alpha\}$, with constant first three members. The priority α has a value in the range of $[0, 1]$ and it can be used in classifying the services according to their priority in the service commodity. The sum of all the priorities must be 1. The coexistence of multiple user's applications is called a multi-commodity flow and $K = \{k_i\}$ (with $i = 1, \ldots, m$) is the set of m commodities (see Table 1).

4.2. Workload arrival modeling

In an edge/cloud data center(s) services arrive for processing. Here we are assuming such processing functions to be designed as a group of microservices to process the user's incoming workload. Multiple services arrive at the data center requiring processing. Since we are considering only arriving workloads that require network function processing.

The workload arrival modeling is assumed to be Poisson arrival with arrival rate λ. The total workload arrival rate is the sum of active users requesting a workload to a given edge/cloud data center at a given time.

$$\lambda_{tot} = \sum_{i=1}^{N_{Usr}} \sum_{k=1}^{N_{App}} \lambda_{ik} \tag{1}$$

where N_{App} number of active applications from a given user requesting workload processing at a given time. N_{Usr} is the number of active users at a given time. λ_{ik} is the workload processing demand in Giga Operations Per Second (GOPS) of a given application from a particular user at a given time. λ_{tot} is the total workload in GOPS.

4.3. Arrival throughput modeling

The total throughput of the arrival services is the sum of throughput requested by all active applications in the active users, requesting a given workload to that particular data center at a given time. We also would like to calculate the total throughput arriving at the data center at a given time, which is given by:

$$\psi_{tot} = \sum_{i=1}^{N_{Usr}} \sum_{k=1}^{N_{App}} \psi_{ik} \tag{2}$$

where ψ_{ik} is the throughput of the applications from a given user and ψ_{tot} is the total throughput arriving at a data center. To simplify the formulation, the total arrival to a given data center is assumed to be a

single aggregation node such as the gateway of the edge/cloud data center just before incoming traffic authentication to access the data center network.

4.4. Modeling microservices-based network functions

As discussed in the introduction section, the trend in software development is to build a softwarized system with loosely coupled units. The network is also being softwarized using SDN and NFV. The architectures help in building a softwarized network, softwarizing network functions such as Network Address Translator (NAT), routing, firewall, serving gateway (s-gateway), packet gateway (p-gateway), Mobility Management Entity (MME). These network functions can be developed as a monolithic system or using SOA such as loosely-coupled microservices and multi-agent systems [1]. Here we focus on formulating microservices-based softwarized network systems.

In an SOA the legacy monolithic system is disaggregated into loosely-coupled small and specific services. The developed microservices-based network functions are usually encapsulated in VMs or containers and deployed in a distributed environment such as the edge/cloud. VM and containers facilitate scalable, flexible, and efficient deployment of network functions in an edge and/or fog and/or cloud environment. However, the desegregation comes at the cost of the added distance between the disaggregated functions. This could result in additional delay. On the other hand, the possibility of deploying only the required function near to user or application such as an edge data center could reduce the delay with a reduced resource and energy cost. Moreover, flexible deployment could enable a reduction in the end-to-end delay and improve reliability if careful design and deployment are followed.

Therefore, formulating the complete decomposed microservices-based system is necessary. This is because, to calculate and perform analysis of microservices-based systems deployed in a distributed environment, we need to have a complete mathematical model for the system. The mathematical model should be comprehensive to consider the overall system behavior when microservices-based service decomposition is employed in the system design. Moreover, containerization and cloudification also alter and may impose further constraints due to added overheads and recomposition possibilities such as decomposition delay, instantiation delay, and overhead delays in the virtual environment. Furthermore, reliability improvement through backup microservices could also reduce delay if possible services can be deployed in the vicinity of the user's spiky workload processing demand.

4.4.1. Microservice modeling

Here we present a mathematical model formulating the overall system for latency as the main parameter while also considering throughput, resource utilization, and energy consumption. First, let us provide a mathematical model for the microservice. A microservice MS can be modeled as

$$MS_f = [P_c, P_l, E_c, In_{wl}, Out_{wl}, In_{thu}, Out_{thu}] \tag{3}$$

where MS_f is a microservice based function with a processing capacity P_c, processing load P_l, energy consumption E_c, input workload In_{wl} and output workload Out_{wl}, input throughput In_{thu} and output throughput Out_{thu}.

Depending on the workload processing requirement, we could have multiple microservices that are chained to provide the required service demanded by a given user. The microservices chain could be arranged in parallel or in series depending on the possibility of parallel or series workload processing. Depending on the arrangement, the formulation for latency varies. The mathematical formulation of latency in microservices chain will be presented in the later section. Here the cumulative services are the combination of individual microservices.

$$MS_{Tot} = MS_{f1}, MS_{f2}, MS_{f3}, MS_{f4}, \ldots, MS_{fn} \tag{4}$$

MS_{Tot} is the number of microservices required to be instantiated to create a microservices chain to execute a given user's workload. The structure of a microservices-based chain is dependent on the requirement of the application/user.

5. Overall system mathematical model formulation

As discussed above the physical and virtual network, the edge data center, and cloud infrastructure are considered in the Multi-Layer Graph. Now we will formulate the resource utilization, end-to-end latency, throughput, energy consumption, and containerization.

5.1. Formulating resource constraints

We considered the presence of multiple edge data centers and cloud data centers. The end-to-end latency of a given workload is the sum of the latency incurred in the physical and virtual network, at the processing server, and in the waiting queues. These latencies are dependent on the scheduling algorithm: which determines where the task should be scheduled, and available resources at scheduled sites such as nearby data centers: which determines waiting time in the queue as well as the processing capacity to be availed for that particular workload, the number and type of users: which determines the amount of workload to be handled and priority class if the scheduler is using a prioritization in the scheduling process. The available resource in a given data center is finite. Which means there is a capacity constraint limiting the utilization of resources. Moreover, active resources such as servers and switches consume energy. Energy consumption is expensive and there is associated carbon emission [28]. Therefore, energy consumption should be minimized. There are various techniques to minimize resource usage and energy consumption [14,29–31]. However, none has considered microservices-based function decomposition and containerization details in the resource and energy consumption modeling.

Here, we consider containerization of microservices that are required to perform a given workload. The granularity of a decomposed system that is containerized may add further resource demand in terms of memory and CPU. That should be considered in the resource constraint formulation. We extend the resource constraint formulation presented in [14]. In general, the total resource required by a group of microservices to process incoming service is given by:

$$R_{cpu-tot-srv} = \sum_{j=1}^{N_{MS}} P_{c-j} \tag{5}$$

where $R_{cpu-tot-srv}$ is the overall resource consumption of a given service without considering containerization of services, and P_c is processing capacity related to the jth microservice.

Microservices can be deployed in multiple edge or cloud data centers. The resource consumption and latency should be modeled considering this distribution. Fig. 2 depicts the path an incoming task or service is required to pass for execution traversing multiple microservices. Some services may only need to use a microservices chain in a single data center and others may have more. Considering multiple edge data centers, the total available CPU that could potentially be used for workload scheduling is given by:

$$R_{cpu-tot} = R_{EDC-1} + R_{EDC-2} + R_{EDC-3} + \cdots + R_{EDC-n} \tag{6}$$

where $R_{cpu-tot}$ is the total processing power of a group of edge data centers working together to perform collaborative service processing. R_{EDC-n} processing power of the nth edge data center.

The resource utilization by a group of microservices is formulated in Eq. (5). Using this equation, the total resource demand is dependent on the incoming workload that requires processing in the microservice. Depending on the demand and the available resource in a given data center, the required microservices are instantiated. Assuming the total

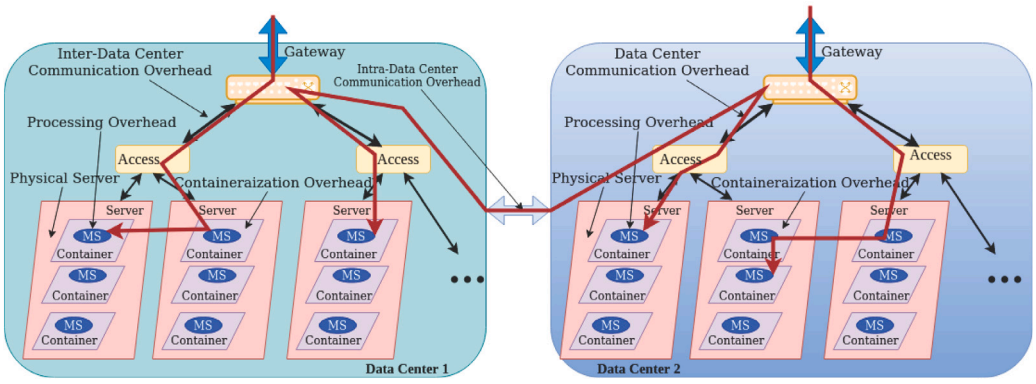


Fig. 2. Microservices-based chains containerization/virtualization delay.

available edge data center is allocated for instantiating containerized microservice, the maximum number of microservices that can be instantiated is constrained as

$$R_{cpu-tot-vir} > \sum_{j=1}^{N_{MS}} P_{c-j} + \sum_{j=1}^{N_{MS}} P_{ov-j} \tag{7}$$

where $R_{cpu-tot-vir}$ is the overall resource consumption of a given service considering containerization of services, P_c is processing capacity related to the jth microservice and P_{ov-j} is processing overhead due to containerization of microservices.

5.2. Modeling latency constraint

Here we present the formulation for the end-to-end latency of the system that will be imposed by the workload, which is dependent on the scheduler that considers geographically distributed edge data centers in the scheduling of the workload. The overall latency is the sum of the delays from the source (user) to sink (edge data center) which is given as

$$L_{e2e} = L_{Trn-ntk} + L_{Qu} + L_{vr} + L_{Trn-edge} + L_{Trn-bwn-edge} + L_{Pro} \tag{8}$$

where $L_{Trn-ntk}$ is the transmission latency of the workload in the network before arriving at a scheduling data center. L_{Qu} is the latency in the scheduling queue. $L_{Trn-edge}$ is the delay incurred in the edge data center, which is the network delay between the scheduling server and microservices hosting servers and/or between microservices hosted on different servers. The workload may require traversing multiple microservices that could be hosted on different servers. $L_{Trn-bwn-edge}$ is the transmission delay between the scheduling data center and the processing data center. Our scheduling assumption is, if there is an available resource and required microservices chain in the scheduling edge data center (where the workload first arrived), the workload is scheduled in the same data center. Otherwise, the workload is forwarded to the next best available edge data center considering added latency between the edge data centers and the available resource in the edge data center. Suppose the workload is scheduled in the same data center as they arrived. In that case, the $L_{Trn-bwn-edge}$ value becomes zero as there is no additional transmission latency to other edge data centers or the cloud. L_{vr} is the virtualization latency due to containerization or virtualization of microservices. The following section discusses how to formulate the virtualization delay.

5.2.1. Virtualization latency

The authors in [26] provided an accurate formulation of the virtualization delay overhead involved in a containerized environment. It is the sum of the delay in the virtual NIC (vNIC), vSwitch, VMKernel, and NIC driver. It is given by [26]:

$$L_{vr} = D_{vNIC} + D_{vSW} + D_{VMKernel} + D_{PNICDriver} \tag{9}$$

Using the above formulation, we have calculated the virtualization delay, in the case of Ryu controller decomposition based on the MSN implementation. The decomposed functions are containerized and the virtualization delay is modeled as depicted in Fig. 3 which shows three stages of the decomposed and containerized Ryu controller functions showing the virtualization latency. The virtualization delay depends on the number of VM hosted in a single server and the total workload of that server. First, the total workload is calculated as:

$$(P_{Tot}) = \sum_{n=1}^{N_V} (P_L)_n \tag{10}$$

So the virtualization overhead delay as a function of co-located VM and workload is given by [26]:

$$F(N_V, P_L) = 0.338 \times N_V \times (P_L)^{12.15} + 0.51 \times (P_L) \tag{11}$$

5.2.2. Physical link latency

The physical latency in the data center is the sum of the delay in each link, each switch and associated queues. This is given by the following equation [14]:

$$D_{phy} = \sum_{o=1}^{O_{pylink}} D_n + \sum_{k=1}^{K_{pysw}} D_k + \sum_{m=1}^{N_{Que}} D_m \tag{12}$$

5.2.3. Processing latency

The processing latency in a serving microservice is:

$$L_{proc} = \sum_{i=1}^{N_{MS}} \frac{L_{ij}}{U_{cont}} \tag{13}$$

The processing delay L_{proc} is variable depending on the given service workload (WL_{ij}), serving rate of each containerized microservice (U_{con}) from a group of microservices (i.e. a so called microservices chain) and number of microservices N_{MS} of a given type.

5.2.4. End to end latency

$$L_{end} = Sourc_{enode} - Sink_{node} \tag{14}$$

The end-to-end latency is calculated as [14].

$$L_{end} = L_{RAN} + L_{Tran} + L_{DC} \tag{15}$$

Moreover, the data center latency can be calculated as

$$L_{DC} = L_{Ntk} + L_{Prc} + L_{Que} \tag{16}$$

However, since the microservice is hosted in a virtualization environment or container, there is an overhead delay that should be considered for each network function that is part of the service function chain. Therefore, using Eq. (2), the processing latency is given by:

$$L_{DC} = L_{Ntk} + F(N_V, P_L) + L_{Que} \tag{17}$$

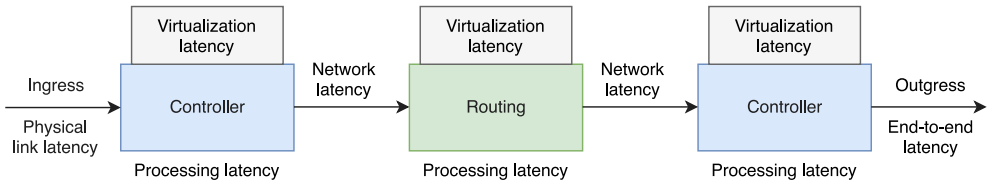


Fig. 3. Virtualization Delay in the MSN microservices Chain.

which can also be written as:

$$L_{DC} = L_{Ntk} + 0.338 \times N_V \times (P_L)^{12.15} + 0.51 \times (P_L) + L_{Que} \tag{18}$$

Assuming the placement of the microservices to the nearest server interns of latency or a server with the least load, we now calculate the cascaded delay in the Service Function Chaining (SFC). A given user application task is constrained to pass through a given sequence of microservices chain. Depending on the application requirements, there are different possibilities for chains. Accordingly, some chains could be in parallel which means some tasks could be performed in parallel. On the other time, they are in series. In a parallel case, the delay is dependent on the path that incurred the highest latency. However, in the series chain case, the delay is calculated as a sum of the cascaded microservices chains. This is formulated as:

$$L_{MSC} = \begin{cases} \sum_{o=1}^{N_{MSC}} L_{MS}, & \text{if series MS chain} \\ L_{MS-C}, & \text{parallel MS chain} \end{cases} \tag{19}$$

5.3. Modeling throughput in virtual environment

The throughput in a virtual environment can be formulated using classical throughput aggregation or desegregation formulation. However, when the service function node is a microservice processing node, the classic formulation could be different. For instance, in a service function chain, given an input throughput, the output throughput amount may be higher or lower or even zero since the nodes in the chain could respond according to the functionality they are designed to perform:

$$\psi_{out} = \begin{cases} \psi_{out} < \psi_{in}, \\ \psi_{out} = 0, \\ \psi_{out} > \psi_{in} \end{cases} \tag{20}$$

where ψ_{out} throughput in the output interface and ψ_{in} throughput in the input interface. Therefore, the throughput can only be determined by the incoming task request and the processing microservice node.

5.4. Modeling energy constraint

Power consumption of data centers hosting network management functions is significant that should be considered in the design [30, 31]. In this regard, microservices-based functions consume power that should be modeled to account for the power consumption of the modularized and containerized functions. Processing and hardware consumption are well studied subjects. However, the modularization of monolithic systems and virtualization/containerization of the modules need a thorough investigation to understand the impact. Here, we have formulated the overall consumption considering each portion of the modularized and containerized function. Processing consumption of a physical server is given by [31]:

$$P_{WL} = P_{idl} - \frac{P_{pk} - P_{idl}}{2} \times (1 + WL - e^{\frac{WL}{\gamma}}) \tag{21}$$

Total power consumption is the sum of the physical server and networking device power consumption. Therefore, the total energy consumption of undecomposed monolithic systems is given by:

$$P_{tot} = P_{pro} + P_{com} \tag{22}$$

However, when we consider a microservices-based decomposition along with containerization, the consumption would be different. In general, the consumption of a given containerized microservice becomes the sum of processing, communication, and virtualization power consumption, which is given by:

$$P_{tot} = P_{pro} + P_{com} + P_{vir} \tag{23}$$

where P_{vir} is the additional power consumption due to the containerization.

In general, placing a given microservice in a server has an impact on the overall energy consumption of the system. Therefore where to place the microservice to minimize energy consumption has to be studied, e.g placing a microservice considering energy cost or without instantiating new server as much as possible. Load concentration on the given physical server without congesting the physical link. Such aspects are crucial for the efficient deployment of microservices-based systems. The study of placement considering power consumption is beyond the scope of this work and is left for future work.

5.5. Modeling nested containers latency

The containerization of functions has added additional delay. To improve the organization and security of decoupled functions, it could be necessary to put two functions in different containers and place these containers in another container. Such kind of organization could be more useful for service function chaining. However, the added latency must be considered. Here we formulate the latency in terms of the number of containerization layers. Let Cl_i be the containerization latency of the ith container in the nested containerization series. The total latency Cl_{tot} of ith level function becomes:

$$Cl_{tot} = L_{pro} + \sum_{i=0}^{N_{con}} Cl_i \tag{24}$$

where L_{pro} is the processing latency of the ith level function, i = 0, 1, 2, 3...

N_{con} i = 0 indicate that the function is not containerized.

6. Experimental evaluation

This section presents our experimental evaluation of the proposed models for microservices-based service chains. First, we introduce some preliminary results on the Docker Container networking in Section 6.1. Second, the latency evaluation of our proposed models for microservices-based service chains based on the Ryu SDN Framework is presented in Section 6.2. Third, in Section 6.3 we present the evaluation of the throughput model based on a video streaming application. Finally, we show the experimental evaluation of the energy consumption in Section 6.4.

For the evaluation purpose, we have run all the experiments in a micro datacenter machine which is equipped with 32-cores AMD Opteron(TM) 2.3-GHz PC with 32-GB RAM. Moreover, the machine provides a specific Docker version such as the 20.10.17 version.

Table 2
Latency comparison between the Docker bridge and host mode.

Payload (bytes)	Bridge		Host	
	median (µs)	std (µs)	median (µs)	std (µs)
1	47.41	4.92	46.03	4.90
2	49.00	4.39	47.59	3.64
4	49.04	3.36	47.57	4.57
8	49.08	4.00	47.61	3.86
16	49.06	3.86	47.65	3.64
32	48.94	5.89	47.63	3.81
64	49.03	3.37	47.64	3.72
128	49.23	3.69	47.98	3.31
256	49.07	3.47	47.97	3.20
512	49.20	3.35	48.06	3.11
1024	49.39	3.73	47.96	3.32
2048	57.64	3.23	48.12	2.98
4096	57.99	5.33	48.45	4.85
8192	59.68	4.30	49.03	3.52
16384	69.07	4.36	50.16	3.11
32768	71.43	5.72	52.31	4.74

6.1. Preliminary results

Before delving into the actual results, we make a few clarifying statements on the impact of Docker Container on the network I/O performance. As already stated and proofed for VNF, packet I/O and processing operations inside the VMs introduce latency [32]. Similarly, containers run in an isolated way on top of the operating system's kernel. Having this additional layer of abstraction may lead to performance degradation. Docker offers two different networking setups: (i) **bridge** that is the default network in Docker where each container has its own network namespace; (ii) **host** that does not create a separated network stack for containers, instead, each container shares the same network stack with the host. Finally, we calculated the delay introduced by a virtualization layer when we have multiple levels of Docker containers (i.e., nested Docker containers).

6.1.1. Docker latency evaluation

To measure latency over a network we used the software *sfnt-pingpong*[1] that measures ping-pong latency over a range of message sizes by using standard network protocols including TCP and UDP. In particular, we measured the ping-pong time between a request made at the host level by the client towards the server that is instantiated in a Docker Container in both network modes (i.e., bridge and host). Note that we reported only results about TCP because is the most used for the majority of the applications. Results are reported in Table 2. These preliminary results show that latency times are a bit lower in the case of Docker host network configuration. This is justified by the fact that as mentioned above, the Docker host configuration shares the network stack with the host, while the Docker bridge creates a virtual network and a virtual bridge named *docker0*.

6.1.2. Nested docker evaluation

To assess the model defined in Eq. (24), we focus on the latency times introduced by nested levels of Docker Container. To ensure this, we leveraged the DinD Docker image[2] that contains the Docker daemon to instantiate a new Docker Container inside it. Note that, for this evaluation, we present only the Docker bridge network mode because the Docker host network mode does not allow the instantiating of the Docker daemon at the same port. The results shown in Fig. 4, represent the latency times for a simple Docker Container httpd server[3] at several different virtualization levels. In particular, we calculate the latency time for nine different levels of virtualization. The requests always start from the host level (L0) while the server is shifted at each level over time (from L1 to L9). Moreover, we used Apache ab[4] for testing the average time per request over 1000 requests. Finally, the request time seems to be linear with the virtualization layers.

6.2. Latency evaluation

In this section, we evaluate the latency model as described in Section 5.2 by leveraging the MSN implementation. Fig. 5 shows the setup of our MSN evaluation environment. In the MSN architecture, we decomposed each SDN functionality as a microservice and successively encapsulated it in a Docker container. The basic MSN configuration is composed of three microservices including the *ofp_handler* (for event handler functionalities), the two internal applications *ofp_emitter* and *ofctl_rest* (we called middleware) and, the external Ryu application (in our case the *simple_switch* application). Note that to work properly MSN needs at least a service chain composed of the three mentioned microservices. The workflow of the MSN microservices chain is as follows. When a host in the network tries to communicate with another host, the *packet_in* event is created. There is no rule installed in the SDN controller for the first time, so the *packet_in* event is transformed into a REST request for the *simple_switch* routing application. Finally, the packet returns via the *send_packet* event.

For each microservice, we evaluated the virtualization latency (if present) and the processing latency, while we included the network latency for the entire service chain evaluation. Moreover, we tested MSN in three different scenarios such as No Docker (no virtualization), Docker host network mode, and Docker bridge network mode. A summary of obtained results are shown in Fig. 6. We did these tests in a Mininet network with a standard three-levels topology composed of 10 hosts and 5 switches. Each evaluation took around 50 runs. Overall, the No Docker scenario has shown to drop down the end-to-end latency time to around 25% compared to the Docker Bridge implementation.

6.3. Throughput evaluation

With the experiments, we analyze the impact of the virtual environment of our microservices-based SDN controller on the average throughput during operation. As in the previous experiments, we leverage our MSN implementation depicted in Fig. 5. In particular, we recorded the control plane traffic incoming and outgoing at the MSN service chain with the Wireshark tool. For each microservice, in this case, the *ryu_middleware* and the *simple_switch*, we evaluated the incoming and outgoing throughput.

To assess the throughput evaluation, we devise a Smart City application aligned with one of the ETSI MEC use cases [33], such as video streaming. Specifically, we focus on a scenario where a user in a locality is connected to the 5G infrastructure to send continuous video streaming. To emulate this, we created a virtualized network with the Mininet simulation tool which creates the network topology composed of 10 hosts, 5 switches, which provide a 1 Mbits bandwidth, and the MSN controller. For the video streaming task, we use the *ffmpeg* software which is a Linux-based standard tool for video/audio streaming.

Fig. 7 shows the observed throughput of the control plane traffic on the link interconnecting the MSN middleware dockerized microservice. Fig. 7(a) shows the average incoming throughput while Fig. 7(b) depicts the average outgoing throughput. On the contrary, Fig. 8 shows the observed throughput of the control plane traffic on the *simple_switch* microservice. As in the previous case, Fig. 8(a) shows the incoming traffic, while Fig. 8(b) shows the outgoing traffic. The performance results show the throughput during the processing of the video streaming

[1] https://github.com/Xilinx-CNS/onload
[2] https://hub.docker.com/_/docker
[3] https://hub.docker.com/_/httpd
[4] https://httpd.apache.org/docs/2.4/programs/ab.html

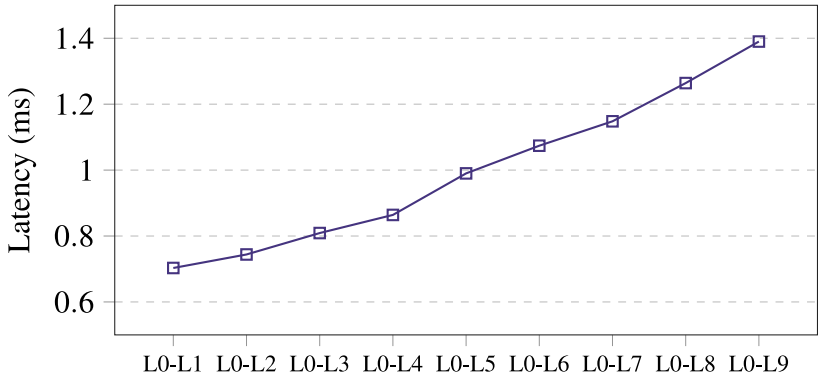


Fig. 4. Nested Docker containers latency.

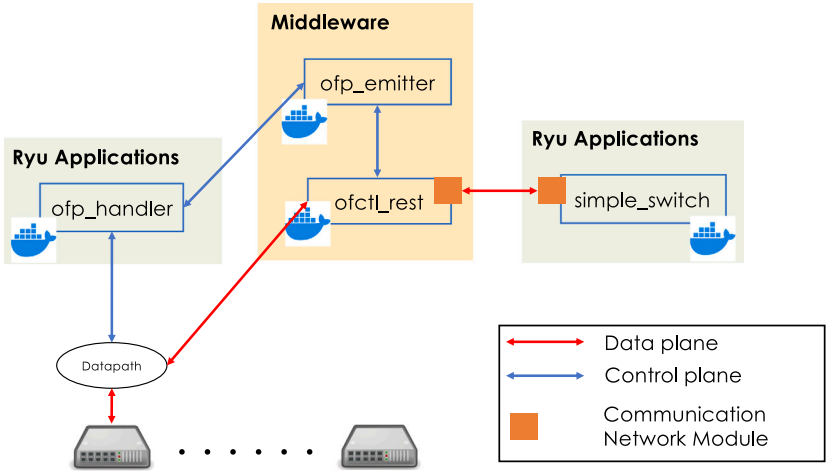


Fig. 5. Microservices-based Ryu SDN Framework implementation.

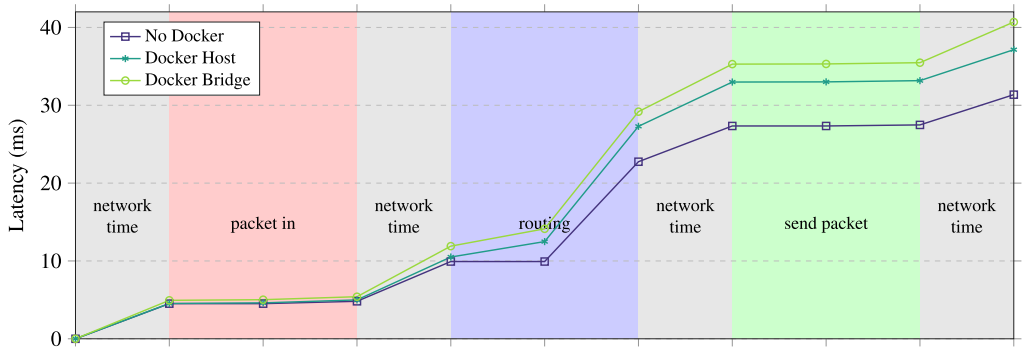


Fig. 6. MSN service chain latency evaluation for each stage.

application. In particular, from Fig. 7 and Fig. 8, is possible to note that the SDN traffic is only for the first packets and for the update packets.

6.4. Energy evaluation

To assess energy consumption, we leverage a tool named Scaphandre[5] which allows calculating the electrical power consumption of software services. Specifically, we focus on the MSN power consumption in both non-dockerized and dockerized fashion.

Fig. 9 shows the comparison of the power consumption between the MSN non-dockerized and dockerized one. We recorded the power consumption during the operation of the network. In particular, we leverage the same Mininet network described for the other experiments and we run the *pingall* command on the entire network for around 30 s. Overall, it is possible to conclude that the non-dockerized version of the MSN consumes around 10% less than the dockerized version. Table 3 shows detailed power consumption per process in correspondence with the maximum peak of power consumption. As observed, running MSN in a containerized environment with Docker involves multiple interconnected processes, each contributing to the overall power consumption.

[5] https://hubblo-org.github.io/scaphandre-documentation/index.html

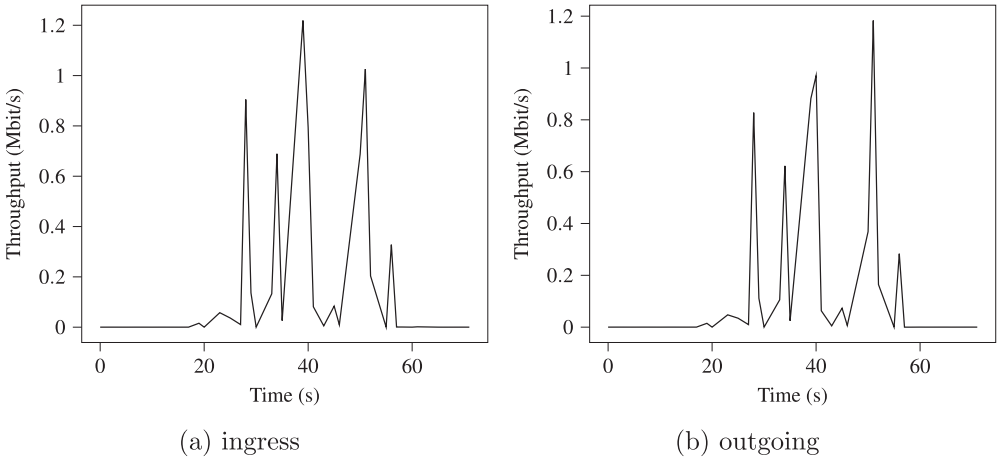


Fig. 7. Throughput observed for control plane traffic in the MSN middleware microservice during video streaming operation.

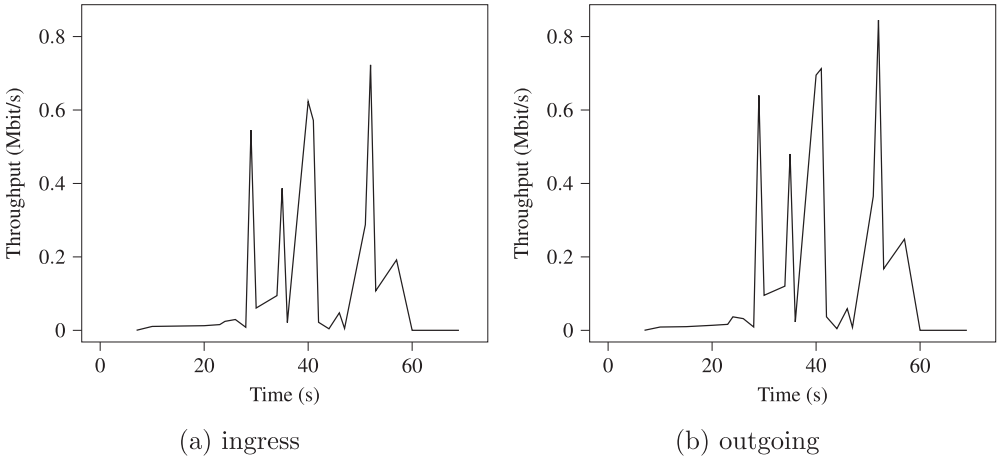


Fig. 8. Throughput observed for control plane traffic in the MSN simple switch microservice during video streaming operation.

Table 3
Comparison of peak energy consumption per process in non-dockerized versus dockerized MSN environments.

Non-dockerized		Dockerized	
Process	Power (W)	Process	Power (W)
ryu-manager	12.65	ryu-manager	11.40
simple-switch	11.46	simple-switch	10.31
ovs-vswitchd	3.74	ovs-vswitchd	3.07
–	–	dockerd	2.35
–	–	containerd-shim	1.98
–	–	docker	1.08
–	–	docker-proxy	0.18
Total cons.	27.85	**Total cons.**	30.37

7. Discussion and conclusion

In this paper, the microservices-based decomposition architecture for future 6G networks is proposed and evaluated. To the best of the authors' knowledge, this is the first performance evaluation in terms of major KPIs (e.g. latency, throughput, and energy) of communication in the end-to-end softwarized network continuum. The theoretical modeling based on hypergraphs makes it more accurate in representing the characteristics and the behavior than the revised state-of-the-art. The theoretical description of the latency of softwarized environments and the tests in the real MSN environment to make the analysis more accurate are used to make the evaluation and the theoretical work more solid and closer to describe a real network behavior. The MSN framework, evaluated in this paper, paves the way to a new generation of microservices-based approaches for the next generation 5G-ready and beyond SDN networks. Microservice-based networks will be the first pillar of future 6G architecture and this work tries to highlight important design guidelines in that perspective. The results presented in this paper are focused on latency, throughput, and energy consumption. Obtained results demonstrate the feasibility of applying microservices-based modeling for 5G and beyond networks including virtualization and containerization aspects.

Boosted by obtained results, we are now working along different ongoing work directions. First, we are working on the inclusion of in-network intelligence both centralized and distributed in the form of multi-agents. The multi-agent system is the key enabler for the next 6G network generation. In particular, we are defining and proposing multi-agent-based autonomic network architecture along with the mathematical modeling of the autonomic network management system. In parallel, we are working on the introduction of intelligent agent support for the MSN framework. In particular, we are creating a new functionality named *topology_learning* as an extension of the SDN topology manager for enabling the *shortest_path* agent. To the best of the author's knowledge, we claim that could pave the way for supporting intelligent agents into microservice-based SDN networks.

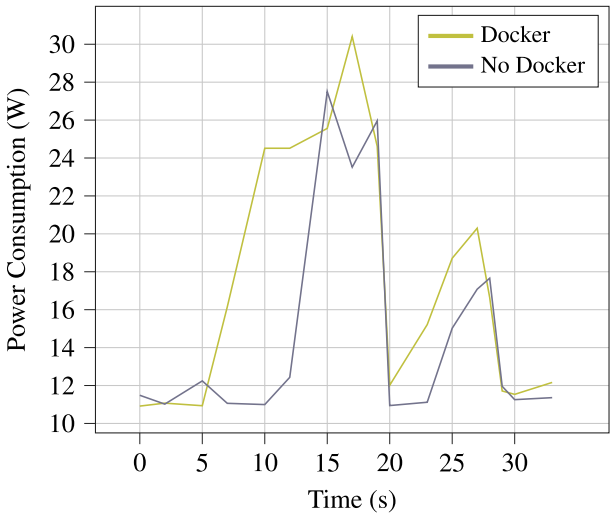


Fig. 9. Comparison between energy consumption for MSN non-dockerized and dockerized.

CRediT authorship contribution statement

Sisay Tadesse Arzo: Conceptualization, Methodology, Writing – original draft. **Domenico Scotece:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. **Riccardo Bassoli:** Conceptualization, Data curation, Supervision, Validation, Writing – original draft. **Michael Devetsikiotis:** Conceptualization, Supervision, Writing – original draft. **Luca Foschini:** Conceptualization, Supervision, Validation, Writing – original draft, Writing – review & editing. **Frank H.P. Fitzek:** Conceptualization, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work has been partially funded by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy – EXC2050/1 – Project ID 390696704 – Cluster of Excellence "Centre for Tactile Internet with Human-in-the-Loop" (CeTI) of Technische Universität Dresden. The authors also acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the programme of "Souverän. Digital. Vernetzt.". Joint project 6G-life, project identification number: 16KISK001K. This work has also been partially funded by the US National Science Foundation under the New Mexico SMART Grid Center - EPSCoR cooperative agreement Grant OIA- 1757207. This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART") CUP: J33C22002880001.

References

[1] S.T. Arzo, R. Bassoli, F. Granelli, F.H. Fitzek, Multi-agent based autonomic network management architecture, IEEE Trans. Netw. Serv. Manag. (2021) http://dx.doi.org/10.1109/TNSM.2021.3059752, 1–1.

[2] E. Berrio-Charry, J. Vergara-Vargas, H. Umaña-Acosta, A component-based evolution model for service-based software architectures, in: 2020 IEEE 11th International Conference on Software Engineering and Service Science, ICSESS, 2020, pp. 111–115, http://dx.doi.org/10.1109/ICSESS49938.2020.9237747.

[3] CodeandPepper, 10 Companies Using Microservices. Who is Using Them?, 2021, URL https://codeandpepper.com/companies-using-microservices/.

[4] NordicAPIS, 4 Examples of Microservices Architectures Done Right, 2021, URL https://nordicapis.com/4-examples-of-microservices-architectures-done-right/.

[5] R. Mijumbi, J. Serrat, J.L. Gorricho, N. Bouten, F. De Turck, R. Boutaba, Network function virtualization: State-of-the-art and research challenges, IEEE Commun. Surv. Tutor. 18 (1) (2016) 236–262, http://dx.doi.org/10.1109/COMST.2015.2477041.

[6] H. Farhady, H. Lee, A. Nakao, Software-defined networking: A survey, Comput. Netw. 81 (2015) 79–95, http://dx.doi.org/10.1016/j.comnet.2015.02.014, URL https://www.sciencedirect.com/science/article/pii/S1389128615000614.

[7] Q. Duan, S. Wang, N. Ansari, Convergence of networking and cloud/edge computing: Status, challenges, and opportunities, IEEE Netw. 34 (6) (2020) 148–155, http://dx.doi.org/10.1109/MNET.011.2000089.

[8] S.T. Arzo, D. Scotece, R. Bassoli, F. Granelli, L. Foschini, F.H. Fitzek, A new agent-based intelligent network architecture, IEEE Commun. Stand. Mag. 6 (4) (2022) 74–79, http://dx.doi.org/10.1109/MCOMSTD.0001.2100053.

[9] Hexa-X, D1.2 – Expanded 6G vision, use cases and societal values — including aspects of sustainability, security and spectrum, 2021, URL https://hexa-x.eu/wp-content/uploads/2021/05/Hexa-X_D1.2.pdf.

[10] Hexa-X, D5.1 – Initial 6G Architectural Components and Enablers, 2021, URL https://hexa-x.eu/wp-content/uploads/2022/01/Hexa-X_D5.1_full_version_v1.0.pdf.

[11] A. Samanta, L. Jiao, M. Mühlhäuser, L. Wang, Incentivizing microservices for online resource sharing in edge clouds, in: 2019 IEEE 39th International Conference on Distributed Computing Systems, ICDCS, 2019, pp. 420–430, http://dx.doi.org/10.1109/ICDCS.2019.00049.

[12] R. Yu, V.T. Kilari, G. Xue, D. Yang, Load balancing for interdependent IoT microservices, in: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 298–306, http://dx.doi.org/10.1109/INFOCOM.2019.8737450.

[13] Y. Niu, F. Liu, Z. Li, Load balancing across microservices, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, pp. 198–206, http://dx.doi.org/10.1109/INFOCOM.2018.8486300.

[14] S.T. Arzo, R. Bassoli, F. Granelli, F.H.P. Fitzek, Study of virtual network function placement in 5G cloud radio access network, IEEE Trans. Netw. Serv. Manag. 17 (4) (2020) 2242–2259, http://dx.doi.org/10.1109/TNSM.2020.3020390.

[15] S.T. Arzo, D. Scotece, R. Bassoli, D. Barattini, F. Granelli, L. Foschini, F.H.P. Fitzek, MSN: A playground framework for design and evaluation of MicroServices-based sdn controller, J. Netw. Syst. Manage. 30 (1) (2021) 1573–7705, http://dx.doi.org/10.1007/s10922-021-09631-7.

[16] Y. Wang, C. Zhao, S. Yang, X. Ren, L. Wang, P. Zhao, X. Yang, MPCSM: Microservice placement for edge-cloud collaborative smart manufacturing, IEEE Trans. Ind. Inform. 17 (9) (2021) 5898–5908, http://dx.doi.org/10.1109/TII.2020.3036406.

[17] A. Samanta, J. Tang, Dyme: Dynamic microservice scheduling in edge computing enabled IoT, IEEE Internet Things J. 7 (7) (2020) 6164–6174, http://dx.doi.org/10.1109/JIOT.2020.2981958.

[18] H. Khazaei, N. Mahmoudi, C. Barna, M. Litoiu, Performance modeling of microservice platforms, IEEE Trans. Cloud Comput. (2020) http://dx.doi.org/10.1109/TCC.2020.3029092, 1–1.

[19] D.B. Oljira, A. Brunstrom, J. Taheri, K.J. Grinnemo, Analysis of network latency in virtualized environments, in: 2016 IEEE Global Communications Conference, GLOBECOM, 2016, pp. 1–6, http://dx.doi.org/10.1109/GLOCOM.2016.7841603.

[20] N. Cruz Coulson, S. Sotiriadis, N. Bessis, Adaptive microservice scaling for elastic applications, IEEE Internet Things J. 7 (5) (2020) 4195–4202, http://dx.doi.org/10.1109/JIOT.2020.2964405.

[21] A. Samanta, Y. Li, F. Esposito, Battle of microservices: Towards latency-optimal heuristic scheduling for edge computing, in: 2019 IEEE Conference on Network Softwarization, NetSoft, 2019, pp. 223–227, http://dx.doi.org/10.1109/NETSOFT.2019.8806674.

[22] I.D. Filip, F. Pop, C. Serbanescu, C. Choi, Microservices scheduling model over heterogeneous cloud-edge environments as support for IoT applications, IEEE Internet Things J. 5 (4) (2018) 2672–2681, http://dx.doi.org/10.1109/JIOT.2018.2792940.

[23] C. Lei, H. Dai, A heuristic services binding algorithm to improve fault-tolerance in microservice based edge computing architecture, in: 2020 IEEE World Congress on Services, SERVICES, 2020, pp. 83–88, http://dx.doi.org/10.1109/SERVICES48979.2020.00031.

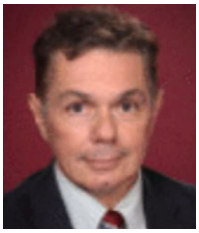
[24] O. Oleghe, Container placement and migration in edge computing: Concept and scheduling models, IEEE Access 9 (2021) 68028–68043, http://dx.doi.org/10.1109/ACCESS.2021.3077550.

[25] S.T. Arzo, F. Zambotto, F. Granelli, R. Bassoli, M. Devetsikiotis, F.H. Fitzek, A translator as virtual network function for network level interoperability of different IoT technologies, in: 2021 IEEE 7th International Conference on Network Softwarization, NetSoft, 2021, pp. 416–422, http://dx.doi.org/10.1109/NetSoft51509.2021.9492677.
[26] D.B. Oljira, K.J. Grinnemo, J. Taheri, A. Brunstrom, A model for qos-aware VNF placement and provisioning, in: 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN, 2017, pp. 1–7, http://dx.doi.org/10.1109/NFV-SDN.2017.8169829.
[27] L. Liu, M.T. Özsu (Eds.), Voronoi tessellation, in: Encyclopedia of Database Systems, Springer US, Boston, MA, 2009, http://dx.doi.org/10.1007/978-0-387-39940-9_3981, 3440–3440.
[28] M. Mursleen, Y. Kothyari, An energy-efficient allocation technique for distributing resources in a heterogeneous data center, in: 2019 International Conference on Advances in Computing and Communication Engineering, ICACCE, 2019, pp. 1–3, http://dx.doi.org/10.1109/ICACCE46606.2019.9079973.
[29] P. Nehra, A. Nagaraju, Sustainable energy consumption modeling for cloud data centers, in: 2019 IEEE 5th International Conference for Convergence in Technology, I2CT, 2019, pp. 1–4, http://dx.doi.org/10.1109/I2CT45611.2019.9033927.
[30] D. Kliazovich, S.T. Arzo, F. Granelli, P. Bouvry, S.U. Khan, Accounting for load variation in energy-efficient data centers, in: 2013 IEEE International Conference on Communications, ICC, 2013, pp. 2561–2566, http://dx.doi.org/10.1109/ICC.2013.6654920.
[31] D. Kliazovich, S.T. Arzo, F. Granelli, P. Bouvry, S.U. Khan, e-STAB: Energy-efficient scheduling for cloud computing applications with traffic load balancing, in: 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, 2013, pp. 7–13, http://dx.doi.org/10.1109/GreenCom-iThings-CPSCom.2013.28.
[32] Z. Xiang, F. Gabriel, E. Urbano, G.T. Nguyen, M. Reisslein, F.H.P. Fitzek, Reducing latency in virtual machines: Enabling tactile internet for human-machine co-working, IEEE J. Sel. Areas Commun. 37 (5) (2019) 1098–1116, http://dx.doi.org/10.1109/JSAC.2019.2906788.
[33] ETSI, Mobile-Edge Computing – Introductory Technical White Paper, 2014, URL https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf.

Sisay Tadesse Arzo is a PostDoc Fellow at the University of New Mexico, USA. He got his Ph.D. in Information and Communication Technology and MSc. in Telecommunication Engineering from the University of Trento, Italy. He received his BSc. from Hawassa University, Ethiopia. He has more than five years of industrial experience in the Telecom industry. He has more than ten journals and papers in cloud computing, network softwarization, and network automation. His research interest included Network Softwarization. Internet of Things(IoT), Network Automation, SmartGrid, Quantum Computing, and Communication and Computing in Space.

Domenico Scotece is an Assistant Professor at the University of Bologna. He received his Ph.D. and M.Sc. degrees in Computer Science Engineering from the University of Bologna in 2020 and 2014, respectively. His research interests include pervasive computing, middleware for fog and edge computing, IoT, management of cloud computing systems, Software-defined Networking, 5G and 6G.

Riccardo Bassoli is a Junior professor (US Assistant Professor, UK Lecturer) at the Deutsche Telekom Chair of Communication Networks and Head of the Quantum Communication Networks Research Group, at the Faculty of Electrical and Computer Engineering, at Technische Universitat Dresden. He is also "member of the Centre for Tactile Internet with Human-in-the-loop (CeTI), Cluster of Excellence, Dresden. He got his Ph.D. from 5G Innovation Centre at University of Surrey (UK), in 2016. Between 2016 and 2019, he was postdoctoral researcher at Università di Trento (Italy). He is IEEE and ComSoc member. He is also member of Glue Technologies for Space Systems Technical Panel of IEEE AESS. His research interests include: quantum communication networks and their integration with 6G and the Tactile Internet, 6G three-dimensional networking, network virtualization, and low-latency resilient communications.



Michael Devetsikiotis (FIEEE) received the Diploma degree in electrical engineering from the Aristotle University of Thessaloniki, Greece, in 1988, and the M.S. and Ph.D. degrees in electrical engineering from North Carolina State University, Raleigh, NC, USA, in 1990 and 1993, respectively. He joined the University of New Mexico, Albuquerque, NM, USA, in July 2016, as a Professor and the Chair of the ECE Department, School of Engineering. His work has received well over 7,000 citations. In 2017, he was inducted to the NC State ECE Alumni Hall of Fame. His research work has resulted in 50 refereed journal articles, 130 refereed conference papers, and 61 invited presentations, in the area of design and performance evaluation of telecommunication networks, complex sociotechnical and cyber–physical systems, efficient simulation, and smart grid communications.

Luca Foschini graduated from the University of Bologna, where he received a Ph.D. degree in computer science engineering in 2007. He is now an associate professor of computer engineering at the University of Bologna. His interests span from integrated management of distributed systems and services to wireless pervasive computing and scalable context data distribution infrastructures and context-aware services. Currently, he is working on mobile crowdsensing and crowdsourcing and management of cloud systems for smart city environments.

Frank H. P. Fitzek (Senior Member, IEEE) is currently a Professor and the Head of the Deutsche Telekom Chair of Communication Networks, TU Dresden. He is the spokesman of the DFG Cluster of Excellence CeTI and the BMBF 6G Hub "6G-life."