UNIVERSITY OF MACEDONIA

SCHOOL OF INFORMATION SCIENCES

DEPARTMENT OF APPLIED INFORMATICS

LIGHTWEIGHT VIRTUALIZATION AND THE NETWORK EDGE

Ph.D. Dissertation

of

Polychronis Valsamas

Supervisor: *Assoc. Prof.* Lefteris Mamatas

Thessaloniki, June 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ ΠΛΗΡΟΦΟΡΙΑΣ

ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΛΑΦΡΙΑ ΕΙΚΟΝΙΚΟΠΟΙΗΣΗ ΚΑΙ ΠΑΡΥΦΕΣ ΔΙΚΤΥΟΥ

Διδακτορική Διατριβή

του

Πολυχρόνη Βαλσαμά

Επιβλέπων: Ελευθέριος Μαμάτας, Αναπληρωτής Καθηγητής

Θεσσαλονίκη, Ιουνίου 2022

Abstract

The evolution of the fifth generation of networks (5G Networks) brought upon a set of new technology solutions that aim to address the increased demands of users (i.e., high data rates, ultra-low latency and reliability) as well as support diverse requirements of next-generation applications. Communication systems are going through radical transformations, incorporating emerging technologies, such as Network Function Virtualization (NFV), Software-Defined Networks (SDNs), and Cloud orchestration systems.

At the same time, micro-data centers are being deployed to the corresponding unified environments, bringing virtualized services closer to the users (i.e., improving service performance and reliability, as well as reducing the communication latency). This new research area is defined as edge computing. However, legacy cloud deployments usually employ traditional virtual machines (VMs), which are resource-costly, face slow time for deployment or scaling up of virtual resources. Given the limited-resource availability as well as the dynamic characteristics of user demands or application requirements in edge clouds, the traditional virtual machines are not suitable in these environments. In this respect, the adoption of alternative lightweight virtualization approaches for edge cloud seek to fulfill such demands, given their particular features e.g. rapid manipulation of virtual resources. Thus, it is essential to seek new flexible orchestration and management solutions that utilize more efficiently the available resources i.e., compared to traditional cloud deployments. In other words, there is a need for systemic adaptations of 5G and Beyond ecosystems towards these goals.

The main challenge of the dissertation is to propose novel flexible, adaptable and efficient edge-cloud orchestration solutions utilizing lightweight virtualization technologies, such as containers and Lightweight Virtual Machines (LVM). The latter technologies bring significant adaptability advantages to dynamic changes in network conditions and service requirements, along with fast responsiveness, being crucial for edge cloud deployments in 5G networks. This dissertation addresses crucial issues associated with 5G networks (i.e., elasticity, scalability, heterogeneity and resource efficiency), identifies the associated trade-offs of utilizing lightweight virtualization technologies, implements and investigates lightweight cloud orchestration platforms with orchestration mechanisms (e.g., efficient resource allocation and service elasticity), as well as proposes a new cloud orchestration strategy (i.e., Virtualization Technology Shifting) that takes into account the performance and resource demands of alternative virtualization technologies to address challenging application and resource requirements.

Keywords:5G Networks and Beyond, Containers, Unikernels, Comparative Evaluation, Edge Cloud Orchestration

Περίληψη

Η εξέλιξη των Δικτύων πέμπτης γενιάς (5G etworks) επέφερε ένα σύνολο νέων τεχνολογικών λύσεων, οι οποίες έρχονται α) να καλύψουν αυξημένες απαιτήσεις/ανάγκες των χρηστών (υψηλούς ρυθμούς μετάδοσης δεδομένων, μείωση της καθυστέρησης επικοινωνίας και αξιοπιστία), β) να υποστηρίξουν τις ποικιλόμορφες απαιτήσεις των αναδυόμενων επόμενης γενιάς εφαρμογών. Τα συστήματα επικοινωνίας μετασχηματίζονται ριζικά ενσωματώνοντας αναδυόμενες ετερογενείς τεχνολογίες, όπως η Εικονικοποίηση Δικτυακών Λειτουργιών (NFV), τα Προγραμματιζόμενα Δίκτυα (SDN) και τα συστήματα ενορχήστρωσης του Υπολογιστικού Νέφους (ΥΝ).

Ταυτόχρονα, στα αντίστοιχα ενοποιημένα περιβάλλοντα αναπτύσσονται τα μικρά νέφη, τα οποία βρίσκονται κοντά στους χρήστες και προσφέρουν εικονικοποιημένες υπηρεσίες υπολογιστικού νέφους, (π.χ. βελτιώνουν την απόδοση, την αξιοπιστία της υπηρεσίας, μειώνουν την καθυστέρηση της επικοινωνίας). Αυτή η νέα ερευνητική περιοχή ορίζεται ως νεφοϋπολογιστική στις παρυφές του δικτύου (Edge Computing). Ωστόσο, τα παραδοσιακά περιβάλλοντα νεφοϋπολογιστικής χρησιμοποιούν συνήθως παραδοσιακά λειτουργικά συστήματα πλήρους κλίμακας (VMs), τα οποία είναι δαπανηρά σε πόρους, αντιμετωπίζουν πολύ αργό χρόνο εκκίνηση ή κλιμάκωσης των εικονικών πόρων. Δεδομένης της περιορισμένης διαθεσιμότητας των πόρων, καθώς και των ποικίλων δυναμικών χαρακτηριστικών των απαιτήσεων των χρηστών ή των απαιτήσεων των εφαρμογών στις παρυφές της νεφοϋπολογιστικής, οι παραδοσιακές εικονικές μηχανές δεν είναι κατάλληλες σε αυτά τα περιβάλλοντα. Από την άποψη αυτή, η υιοθέτηση εναλλακτικών προσεγγίσεων ελαφριάς εικονικοποίησης στις παρυφές του δικτύου επιδιώκει να ικανοποιήσει αυτές τις απαιτήσεις αξιοποιώντας τα ιδιαίτερα χαρακτηριστικά τους, όπως η ταχεία διαχείριση των εικονικών πόρων. Συνεπώς, είναι απαραίτητο να αναζητηθούν νέες ευέλικτες λύσεις ενορχήστρωσης και διαχείρισης που να αξιοποιούν αποτελεσματικότερα τους διαθέσιμους πόρους, σε σύγκριση με τα παραδοσιακά περιβάλλοντα νεφοϋπολογιστικής. Με άλλα λόγια, υπάρχει ανάγκη για συστημικές προσαρμογές των οικοσυστημάτων πέμπτης και των επόμενων γενιών δικτύων (5G and Beyond) προς την κατεύθυνση αυτών των στόχων.

Η κύρια πρόκληση της διατριβής είναι να προταθούν νέες ευέλικτες και προσαρμόσιμες αποδοτικές λύσεις ενορχήστρωσης του νέφους στις παρυφές του δικτύου χρησιμοποιώντας τεχνολογίες ελαφριάς εικονικοποίησης, όπως οι τεχνολογίες υποδοχέων (Containers) και ελαφριών εικονικών μηχανών (LVM). Οι τελευταίες τεχνολογίες επιφέρουν σημαντικά πλεονεκτήματα προσαρμοστικότητας στις δυναμικές αλλαγές των συνθηκών του δικτύου και των απαιτήσεων των υπηρεσιών, προσφέροντας γρήγορη απόκριση, που είναι ζωτικής σημασίας για την λειτουργία των νεφών στις παρυφές της νεφοϋπολογιστικής στα δίκτυα πέμπτης γενιάς. Η παρούσα διατριβή εστιάζει σε κρίσιμα ζητήματα που σχετίζονται με τα δίκτυα πέμπτης γενιάς (δηλ., ελαστικότητα, επεκτασιμότητα, ετερογένεια και αποδοτικότερη χρήση των πόρων) και προσδιορίζει σχετικά εναλλακτικά οφέλη της χρήσης διαφορετικών ελαφριών μηχανών εικονικοποίησης. Γιάυτό το σκοπό υλοποιεί και διερευνά πλατφόρμες ενορχήστρωσης των νεφών στις παρυφές του δικτύου με μηχανισμούς ενορχήστρωσης (π.χ. αποτελεσματική κατανομή των πόρων και ελαστικότητα των υπηρεσιών), και προτείνει μια νέα στρατηγική ενορχήστρωσης νέφους (εναλλαγή των τεχνολογιών εικονοποίησης) λαμβάνοντας υπόψη το αντίκτυπο της απόδοσης και κατανάλωσης των πόρων των εναλλακτικών τεχνολογιών εικονικοποίησης για την αντιμετώπιση των απαιτήσεων των εφαρμογών και πόρων.

Λέξεις κλειδία: Δίκτυα πέμπτης γενιάς, Υποδοχέων, Εικονικές μηχανές ενός σκοπού, Ενορχήστρωσης

Acknowledgements

First and foremost I am extremely grateful to my supervisor, Assoc. Prof. Lefteris Mamatas for his invaluable advice, continuous support, and patience during my Ph.D. study. His immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life. Finally, I would like to express my gratitude to my parents that without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study.

Contents

1 Introduction 1
- 1.1 Context . . . 1
- 1.2 Objectives & Research Contributions . . . 4
  - 1.2.1 Objectives . . . 4
  - 1.2.2 Research Contributions . . . 5
- 1.3 Structure of Dissertation . . . 8
- 1.4 Published Work . . . 9
  - 1.4.1 Scientific Journals . . . 9
  - 1.4.2 International Scientific Conferences . . . 9
  - 1.4.3 Awards . . . 10

2 Platform solutions for 5G Networks 11
- 2.1 Introduction . . . 11
- 2.2 Contributions and Chapter Organization . . . 12
  - 2.2.1 Contributions . . . 12
    - 2.2.1.1 Unikernel-based lightweight cloud technologies and Change-Point Analysis in CDN environments . . . 12
    - 2.2.1.2 Multi-homing with Unikernel-based lightweight cloud technologies in mobile environments . . . 13
    - 2.2.1.3 A Multi-domain Experimentation Environment for 5G Media Verticals . . . 14
  - 2.2.2 Chapter Organization . . . 15
- 2.3 Related Works . . . 16
  - 2.3.1 Relevant CDN Architectures for Resource-Efficient Allocation driven by CPD mechanisms with unikernels technologies. . . . 16
  - 2.3.2 Relevant federated infrastructures for E2E Network Slicing . . . 18
- 2.4 Content Distribution Networks (CDNs) for 5G Networks . . . 18
- 2.5 Implementing elasticity CDN based on Unikernels . . . 19
  - 2.5.1 The UNIC Platform Architecture . . . 19
    - 2.5.1.1 Content Popularity Changes Detection . . . 21
    - 2.5.1.2 Modular VM Placement . . . 24
- 2.6 Experimental Methodology . . . 25
- 2.7 Experimental Results . . . 26
  - 2.7.1 Scenario 1: Impact of change-point detection mechanisms . . . 28
  - 2.7.2 Scenario 2: Impact of VM placement algorithms . . . 30
- 2.8 Extended UNIC platform capabilities for mobile IoT environments . . . 32
  - 2.8.1 Experimental results of multi-homed network utilizing lightweight cloud resources . . . 34
  - 2.8.2 Synergy of multi-homing capabilities along with adaptable network protocols in IoT environments . . . 36
- 2.9 The 5G-CDN Platform solution for heterogeneous distributed infrastructures . . . 37

2.10 Design and Implementation of 5G-CDN Platform 37
2.10.1 Multi-Domain Experiment Engine 39
2.10.2 Management and Orchestration 40
2.11 Evaluation Results 42
2.11.1 Performance and scalability of 5G-CDN platform 42
2.11.1.1 Experimentation Setup 43
2.11.1.2 Experimental Results 44
2.11.2 Dynamic resource discovery for scalable CDNs 47
2.11.2.1 Experimental Results 48
2.11.3 Heterogeneity on E2E slicing 53
2.11.3.1 Experimentation Setup 53
2.11.3.2 Experimental Results 54
2.11.4 Modular service orchestration 58
2.12 Conclusions 59

3 A Comparative Evaluation of Edge Cloud Virtualization Technologies 60
3.1 Introduction 60
3.2 Contributions and Chapter Organization 62
3.2.1 Contributions 62
3.2.2 Chapter Organization 62
3.3 Related Works 63
3.4 Edge Cloud Experimentation Environment 66
3.5 Methodology and Assumptions 71
3.6 Experimentation Results 74
3.6.1 Scenario 1: Deployment and Demobilization 75
3.6.2 Scenario 2: Service Operation 79
3.6.3 Scenario 3: Elasticity 83
3.7 Design Guidelines for Edge Cloud Systems 88
3.8 Conclusions 90

4 Virtualization Technology Shifting for Resource-Efficient Edge Clouds 91
4.1 Introduction 91
4.2 Contributions and Chapter Organization 93
4.2.1 Contributions 93
4.2.2 Chapter Organization 94
4.3 Related Works 94
4.4 System Model 98
4.5 Optimization model for resource allocation 102
4.6 Methodology & Empirical Complexity Analysis 105
4.6.1 Empirical Complexity Analysis 109
4.7 Evaluation Simulation Results 110
4.7.1 Scenario 1: Impact of Adopting VTS 111
4.7.2 Scenario 2: Assessing VTS 114
4.8 Example Integration of VTS 120
4.9 Conclusions 121

5 Conclusions and Further Work Discussion 122
5.1 Summary and Conclusions 122
5.2 Further Work Discussion 124

References 126

Appendices 140

Appendix A Funding 140

List of Figures

1 The Architecture of UNIC Platform 20
2 Platform's orchestration & management features implemented as Node-RED nodes 21
3 Content-views per minute of a particular youtube video and detected change-points for different α and γ values. Red lines symbolize the upward and downward change. 27
4 The servers' CPU utilization with the change-point detection mechanisms disabled, using the Objective Weight Function placement algorithm 28
5 The servers' memory allocation with the change-point detection mechanisms disabled, using the Objective Weight Function placement algorithm 28
6 The servers' CPU utilization with the change-point detection mechanisms enabled, using the Objective Weight Function placement algorithm 29
7 The servers' memory allocation with the change-point detection mechanisms enabled, using the Objective Weight Function placement algorithm 29
8 The servers' CPU utilization with the change-point detection mechanisms enabled, using the Random placement algorithm 31
9 The servers' memory allocation with the change-point detection mechanisms enabled, using the Random placement algorithm 31
10 The extended Architecture of UNIC Platform 33
11 Unikernel-based web server caching real IoT measurements 35
12 Multi-homing for mobile users 35
13 The architecture of proposed 5G-CDN platform 38
14 Content distribution service 43
15 Deployment time of CDN service. 45
16 Network stitching time. 46
17 Resource discovery results in different testbeds 50
18 Number of alternative solutions with different resource requirement per VEs. 51

19 Abstract view of the E2E slicing in our experiment 54
20 Communication performance with different Unikernel technologies in E2E slices 56
21 CPU utilization with different Unikernel technologies and node types. . 57
22 The different unikernel technologies are evaluated in line with the content size (MB) 57
23 The placement algorithms' performance 58
24 Experimentation environment 67
25 Elasticity workflow 70
26 Service fulfillment of alternative virtualized services - vertical involvement of cloud resources 76
27 Service fulfillment of alternative virtualized services - horizontal involvement of cloud resources 77
28 Communication performance - Total Delivery Time as Response (light colored) plus Download Time (dark colored) 80
29 Communication performance - Throughput 81
30 Service assurance - CPU Peak 82
31 Horizontal elasticity with 100Mbps bandwidth for service orchestration 84
32 Horizontal elasticity with 10Mbps bandwidth for service orchestration . 86
33 Conceptual edge cloud orchestration platform 89
34 System Model 99
35 Execution Time of increasing request demands 110
36 Impact of Virtualization Technology Shifting for a best-effort service . . 112
37 Impact of Virtualization Technology Shifting for a service under a performance goal (i.e., $\lambda = 3.5$) 113
38 Assignment of client requests (high bandwidth capacity) 117
39 Overall CPU utilization per PoP (high bandwidth capacity) 117
40 Overall network throughput per PoP (high bandwidth capacity) 118
41 Assignment of client requests (moderate bandwidth capacity) 118
42 Overall CPU utilization per PoP (moderate bandwidth capacity) 119

43 Overall network throughput per PoP (moderate bandwidth capacity) . . 119
44 Example workflow of a Kubernetes-based system integrating VTS (VTS extensions in red) 121

List of Tables

1 Standard deviation of deployment time for core cloud nodes 47

2 Standard deviation of deployment time for edge cloud nodes 47

3 Resource requirements for a test-bed to host a core server 52

4 Resource requirements for a test-bed to host an edge server 52

5 Alternative solutions 53

6 Related works comparing alternative virtualization technologies that are suitable for edge cloud environments 64

7 Image sizes of different virtualized services 72

8 Association of considered metrics with related works' categorization criteria 73

9 Model Notations 102

10 CPU resource requirements of a content request ($RC_{r,s,v}$) 108

11 Memory resource requirements of a content request ($RM_{r,s,v}$) 108

12 Network throughput demands of a content request assigned to a particular virtualized service ($td_{r,v}$) 108

13 Total Delivery Time (sec) with alternative virtualization options ($T_{r,v}$) . . 109

14 The setup of empirical complexity analysis 109

15 Bandwidth capacity ($bc_{p,c}$) of link (p, c) in Mbps, i.e., between Edge Cloud (EC) or Core Cloud (CC) PoPs and user cluster c 115

1 Introduction

1.1 Context

The fifth-generation (5G) of mobile networks is envisioned to support a huge variety of vertical applications, such as augmented/ virtual reality (AR/VR), autonomous driving, tactile Internet, smart city, smart factories, video-streaming, eHealthcare, media and entertainment services. All these applications have diverse stringent requirements in terms of low latency, high throughput, and high reliability in order to achieve adequate Quality of Experience (QoE).

The requirements of the new 5G applications can be grouped into three main use case categories as identified by the international telecommunication union (ITU) [1] and the 5G infrastructure Public Private Partnership (5G-PPP) [2], in particular: (i) Enhancing Mobile Broadband (eMBB), demanding increased network throughput and bandwidth capacity; (ii) Ultra-reliable low-latency communications (URLLC) targeting, ultra-low latency and high availability; (iii) Massive machine-type communications (mMTC) supporting extremely high numbers of devices, typically being energy-constrained and communicating low-volume data. Therefore, it becomes clear that these use cases have a set of heterogeneous requirements that cannot be satisfied by the fourth-generation (4G) networks and are partially satisfied by early 5G deployments. In other words, the one-size-fits-all approach to network infrastructure is no longer sufficient for the evolution of 5G networks, which is expected to meet the requirements of diverse use-cases.

Furthermore, currently, the end-user of next-generation applications not only consumes data (i.e., video) but also produces a large amount of data transmissions (i.e, uploading posts, photos and videos to the social network sites), thus stresses the network resources, which may create a bottleneck towards the cloud hosting the application. In this context, Edge computing [3] is an important enabling technology in 5G ecosystems, which brings computing, storage and network resources deployed at the edge of the Internet, referred to as micro data-centers, i.e., reducing the physical and logical distance between the hosts and the edge devices. For example, Over The Top (OTT) providers (i.e., YouTube, Netflix and Amazon) are pushing their services from the cloud to the edge, since it is more resource-efficient to handle the data produced for the services in the edge, which improves service performance, reliability and responsiveness.

The 5G network landscape is gradually shifting in the edge, in terms of new use case requirements, end-user augmented demands and a general deployment of computing facilities. However, this transition requires holistic and fundamental changes in the network architecture. Furthermore, there is a major concern for network operators and service providers on how to properly exploit the available computing substrate regarding the overall network and service management. Hence, 5G and beyond (5GB) ecosystems need to adopt new architectural approaches that are more adaptable, flexible, programmable and scalable towards addressing the stringent requirements of diverse services associated with the foreseen 5G use cases.

In this context, several research projects targeting 5G in the academic and industrial area have proposed new architecture and technology enablers. To this extent, the main key pillars supporting the key 5G requirements are Software-Defined Networking (SDN) [4] / Network Function Virtualization (NFV) [5], [6], end-to-end network slicing [7], Edge Cloud Orchestration and other enabling technologies [8].

SDN and NFV are complementary technologies aiming to handle the dynamic applications or users' demands and the limitations of cloud resources, in a flexible and adaptable manner. SDN enables a flexible programmable network environment while NFV virtualizes the network functions running on commercial-off-the-shelf (COTS) infrastructure through virtualization technologies that bring on-demand flexible provisioning (i.e., scale up/down) and fast deployments [9]. The use of SDN/NFV can enable Network slicing [7], which is defined as the support of logical on-demand networks, relying on a common underlying infrastructure, comprising of physical and/or virtual resources, with independent control, management and orchestration. These self-contained networks must be mutually isolated from each other, and flexible enough to accommodate different business-related use cases from different tenants, on a shared infrastructure [10].

However, to fully exploit the potential of the above key enablers, while being integrated with edge computing, a novel software architecture is required, that is aligned to the evolution of 5GB ecosystems. In this context, microservices architecture targets overcoming the limitation of centralized, monolithic architecture, as well as offering modularity, continuous and fast delivery, improved scalability and autonomy features. In practice, the microservices paradigm [11] breaks down traditional software architectures to minimal, single-purpose, communicating service functions, scaled towards bespoke resource allocation and fault-tolerance.

Such microservices could exploit light virtualization (LV) approaches. There is an on-going effort towards adopting LV for edge clouds, i.e., from European 5G-PPP [12], including containers [13] and unikernels [14]. Specifically, containers are standardized units implementing application packaging with all of its dependencies and unikernels are single-purpose appliances specialized at compile-time into standalone kernels, acting as individual software components and sealed against modification when deployed in the cloud. Both virtualization options can be adopted, even in the same edge cloud deployment, since they offer diverse performance capabilities. For example, containers can provide a robust performance, while unikernels have rapid manipulation capabilities, e.g., they can boot up just in ms, even with a TCP SYN or DNS lookup request packet [15].

Besides network services, Internet application services benefit from microservices, as well. For example, the microservices paradigm provides new opportunities for Content Delivery Networks (CDNs) solutions. CDNs are usually based on proprietary software tightly coupled to particular data centers and have difficulty exploiting edge clouds, unlike next-generation 5G applications. Hence, CDNs need to embrace the new emergings network architecture paradigm to tackle challenging aspects, such as elasticity, scalability, heterogeneity, resource-efficiency and adaptability to dynamic content consumer requirements, in resource-constraint conditions.

Without a doubt, the microservices architecture, combined with the emerging technologies (i.e., NFV/SDN, NS, edge computing) brings many advantages regarding scalability and flexibility. However, orchestrating edge cloud resources is a complex and non-trivial task introducing new challenging issues on their management, control and operation. For instance, large-scale service deployments usually serve a vast amount of users spread throughout the globe, which require the involvement of edge cloud resources in many different places. However, it is challenging to deploy network and application services near every user, considering that edge cloud resources may be associated with unfeasible or limited resource availability and incompatible virtualization technologies. Furthermore, user demands, application requirements or network conditions may be dynamic, therefore edge clouds should also support a quick deployment or removal of virtual resources, implementing horizontal and vertical elasticity processes, i.e., adapting the service deployment and cloud resources to these requirements [16], respectively.

Consequently, an edge cloud orchestrator should take into consideration a combination of criteria regarding the decision on where to deploy the virtualized network or application service. It should consider the optimization of cloud resources to accommodate particular application performance requirements, mitigate a potentially limited resource availability, consider the heterogeneity of infrastructures, support responsive elasticity capabilities and utilize light virtualization approaches. However, we suggest to consider an additional aspect: the selection of the most appropriate virtualization solution that can be put in place, as an alternative resource control optimization strategy. In the next section, we present the goals and contributions of this thesis.

1.2 Objectives & Research Contributions

1.2.1 Objectives

The thesis investigates the following key research question: "How can the resource-efficiency and rapid deployment capabilities of various lightweight virtualization technologies (i.e., containers and multiple unikernel flavors) contribute in the resource allocation efficiency and elasticity of edge cloud deployments?". In this context, we introduce edge cloud orchestration systems, mechanisms and related experimentation environments. We set the following main objectives:

- Investigate novel edge cloud architectures with corresponding components and abstractions, as well as implement mechanisms exploiting the benefits of diverse lightweight virtualization flavors, in terms of elasticity, scalability, resource efficiency and service performance.

- Propose experimentation environments and methodologies for assessing edge cloud solutions employing lightweight virtualization.

- Analyze performance trade-offs and resource-utilization characteristics of light virtualization (LV) technologies. Build up on these results towards introducing combined virtualization strategies for edge clouds, selecting the most appropriate virtualization technologies to given application requirements or network or cloud resource conditions.

- In parallel to above investigations, we suggest further improvements and key design guidelines that future cloud-native architectures can exploit, in the context of 5GB networks.

1.2.2 Research Contributions

Here, we enlist the main contributions of this dissertation:

- Developed and investigated novel edge cloud orchestration platforms and corresponding mechanisms, enabling new lightweight edge cloud paradigms targeting crucial aspects in the emerging 5G Networks, i.e., elasticity, scalability, heterogeneity, E2E network slicing and adaptability. These facilities include:

    1. UNIC, an elastic CDN platform that efficiently detects viral content and rapidly deploys tiny Unikernel-based VMs, in response to significant changes in the users' demand. UNIC combines: (i) modular orchestration features (i.e. efficient LVs placement considering recent server resource utilization and a dynamic load-balancer); and (ii) applied content-popularity change-point detection driving VM orchestration to scale the resources. In compar-

ison to traditional CDN solutions, our approach provides significant elasticity capabilities (i.e., flexible on-demand content provisioning) through utilizing tiny Unikernel-based VMs. Our experimental results demonstrated improvements in terms of system responsiveness and resource efficiency.

2. An extended UNIC platform orchestration with additional network features (i.e., multi-homing connectivity control), improving the mobile end-users' experience through utilizing lightweight virtualization technologies in high-mobile environments with operational Mobile Broadband (MBB) connectivity. Our results show that orchestrating network resources with lightweight clouds improve the QoE of mobile users.

3. 5G-CDN, a novel experimentation environment for large-scale, multi-domain E2E slicing, addressing the scalability and heterogeneity aspects of both physical and virtual resources. 5G-CDN supports: (i) appropriate abstractions and interfaces, that handle and hide the complexity of heterogeneous physical and virtual resources; (ii) a Graphical User Interface (GUI) that serves as the platform interface based on YAML schema to allow defining the service specifications and relations among service components; (ii) supports all the required functionality for slice instantiation, such as resource discovery, slice embedding, resource provisioning, slice stitching (intra and inter-Domain network configuration) and service deployment of alternative virtualization flavors; (iv) real-time monitoring for physical and virtual resources; and (v) a bespoke visualization tool.

4. A novel edge cloud experimentation environment that: (i) supports all basic edge cloud operations, including the deployment, removal and operation of virtualized web-based service; (ii) realizes web load prediction and balancing, as well as both horizontal and vertical elasticity actions; and (iii) supports common abstractions and APIs over heterogeneous virtual resources.

- Evaluated the performance dynamics of alternative edge cloud technologies. More specifically:

    1. We provided an extensive experimentation analysis study, which was missing for the literature, investigating the performance trade-offs of alternative edge cloud technologies, including different container builds and alternative unikernel flavors with respect to all basic edge cloud operations (i.e., resource allocation, removal, service operation, horizontal and vertical elasticity actions). We experimented from both service and infrastructure viewpoints, in terms of service fulfillment, service assurance and communication performance. Our extensive experimental results provided useful insights showing that each virtualization option is characterized by particular performance trade-offs.
    2. We consolidated the gained insights from our realistic experiments and provided essential key design guidelines of a conceptual edge cloud orchestration platform for 5G and beyond networks.

- Exploited the performance trade-offs of alternative virtualization options through novel cloud orchestration features, i.e., facilities targeting adaptability to dynamic network, service and cloud resource conditions. Along these lines:
    1. We introduced a new cloud orchestration strategy, called Virtualization Technology Shifting (VTS). VTS exploits the performance particularities of alternative virtualization options, including unikernels and containers, which can be dynamically switched between each other, aiming to increase the availability of edge cloud resources, improve adaptability (i.e., dynamic network and cloud resource condition) and handle cases with challenging resource requirements.
    2. To assess the benefits and validate the feasibility of our solution, we provided a first VTS cloud resource optimization framework, including an optimization model as well as a relevant system model.
    3. Our extensive simulation results showed the benefits of adopting VTS as well as the adaptability of the proposed system under different conditions (i.e., service performance goal, server configuration and network capacity)

towards maximizing the number of accommodated users and minimizing the utilization of server and network resources.

1.3 Structure of Dissertation

The remainder of this dissertation is organized as follows:

- **Chapter 2** presents in detail two novel edge cloud orchestration facilities, namely UNIC and 5G-CDN, including their main components, functionalities and novel mechanisms towards bridging the gap between Unikernel-based technologies and 5G Networks (i.e., Multi-Access Edge Computing), tackling associated research aspects, such as elasticity, heterogeneity, performance optimization and scalability.

- **Chapter 3** investigates the diverse performance trade-offs regarding different unikernel flavors, container builds, and implementations of the same service. We conduct an extensive experimental evaluation of the alternative options considering all basic edge cloud processes (i.e., resource allocation, removal, service operation, horizontal and vertical elasticity actions). Finally, we provide key design guidelines for conceptual edge cloud systems build-up on our research results.

- **Chapter 4** introduces and elaborates a new cloud orchestration strategy, called Virtualization Technology Shifting (VTS), backed by a cloud resource optimization framework. We conduct extensive simulation evaluations, investigating the benefits and validating the feasibility of our solution under different network, service and cloud resource conditions.

- **Chapter 5** concludes the dissertation and summarizes our research results, discussing also further improvements and future research directions derived from this research work.

1.4 Published Work

The scientific findings of this thesis have been published in the following journals and conference papers.

1.4.1 Scientific Journals

J.1 T. Theodorou, G. Violettas, **P. Valsamas**, S. Petridou, Lefteris Mamatas, "A Multi-Protocol Software Defined Networking Solution for the Internet of Things", *IEEE Communications Magazine, vol. 57, no. 10, pp. 4248, Oct. 2019;*

*J.2 **P. Valsamas**, L. Mamatas, and L. M. Contreras, "A Comparative Evaluation of Edge Cloud Virtualization Technologies", IEEE Transactions on Network and Service Management, **Accepted**, Nov. 2021, ISSN: 1932-4537, doi: 10.1109/TNSM.2021.3130792;*

*J.3 **P. Valsamas**, L. Mamatas, and L. M. Contreras, "Virtualization Technology Shifting for Resource-Efficient Edge Clouds", IEEE Access, **Submitted**, 2022.*

1.4.2 International Scientific Conferences

C.1 **P. Valsamas**, S. Skaperas and L. Mamatas, "Elastic Content Distribution Based on Unikernels and Change-Point Analysis", *in Proc. 24th Eur. Wireless Conf. (EW), Catania, Italy, 2-4 May, 2018, pp. 1-7;*

*C.2 **P. Valsamas**, S. Skaperas, G. Violettas, T. Theodorou, S. Petridou, D. Vardalis, A. Tsioukas and L. Mamatas, "Experimenting with Cloud and Network Orchestration for Multi-Access Edge Computing", Demo Paper, IEEE Wireless Commun. Netw. Conf. (WCNC), Marrakech, Morocco, April 2019;*

*C.3 **P. Valsamas**, I. Sakellariou, S. Petridou, and L. Mamatas, "A Multi-domain Experimentation Environment for 5G Media Verticals", in IEEE INFOCOM Workshop on Computer and Networking Experimental Research using Testbeds (CNERT), Apr 2019, pp. 461-466;*

*C.4 P. D. Maciel, F. L. Verdi, **P. Valsamas**, I. Sakellariou, L. Mamatas, S. Petridou, and S. Clayman, "A Marketplace-based Approach to Cloud Network Slice Composition Across Multiple Domains", in 2nd Workshop on Advances in Slicing for Softwarized Infrastructures (S4SI), co-hosted at the 5th IEEE NetSoft, Paris, June 2019;*

*C.5 **P. Valsamas**, P. Papadimitriou, I. Sakellariou, S. Petridou, L. Mamatas, S. Clayman, F. Tusa, and A. Galis, "Multi-PoP network slice deployment: A feasibility study", in Proc. IEEE 8th Int. Conf. Cloud Netw. (CloudNet), Nov. 2019, pp. 1-6;*

*C.6 Kalafatidis, V. Demiroglou, S. Skaperas, G. Tsoulouhas, **P. Valsamas**, L. Mamatas, and V. Tsaoussidis, "Experiments with SDN-based Adaptable Non-IP Protocol Stacks in Smart-City Environments", Demo Paper, in 27th IEEE Symposium on Computers and Communications (ISCC), Rhodes, Greece, June 2022.*

1.4.3 Awards

- In the scope of UNIC project, we developed and implemented a novel elastic CDN paradigm deployed over the Fed4FIRE+ infrastructure. Our solution utilized lightweight Unikernel-based Virtual Machines (VMs) and employed content popularity detection mechanisms that drive the deployment of the content delivery service. We participated in the 5th Fed4FIRE+ Engineering Conference, Apr. 19, Copenhagen, Denmark, where UNIC received the Best Demo Award.

2 Platform solutions for 5G Networks

2.1 Introduction

The exponential growth of Internet content, in size, quantity and network traffic demands, along with the wide adoption of powerful end-devices (e.g., smart-phones, tablets, end-nodes, wireless sensors) are driving forces for changing the way of development and deployment of new services. 5G networks are considered as the main enabler for new services (e.g., Media Entertainment verticals), allowing ultra-low delays, high numbers of mobile clients, resource-demanding operations, and large-scale ultra-high definition streaming. To boost this evolution is essential to enable new low-complexity and flexible network architectures that include advanced service orchestration and management mechanisms (e.g., efficient load balancing, AI/ML, optimal content caching, service elasticity, e.t.c) over multi-domain 5G ecosystems. Traditional solutions, especially in CDNs clouds, are characterized by inflexible and monolithic infrastructures unable to support heterogeneous and high-mobility environments, facing a number of difficulties including: (i) lack of common abstractions or APIs handling heterogeneity spanning from hardware to virtualization technology, operating system, and application; (ii) inefficient virtualization technologies used, including traditional virtual machines (VMs), that face slow times for deployment, downloading or scaling up of virtual resources; and (iii) software applications that are based on proprietary software tightly coupled to particular data centers, hardware, operating systems or virtualization technologies. Since such approaches are becoming inefficient for 5G networks, we suggest targeting ultra-low latency services through lightweight edge clouds that host (or cache) the content near the end-users.

In this context, network slicing is a central concept to the 5G success, since it declares a shift from existing monolithic cellular network architectures to the creation of virtual networks tailored to the performance requirements of each particular service. A 5G slice integrates core/mobile edge cloud and network resources in an isolated, guaranteed, in terms of performance, end-to-end (E2E) virtual network, with fast deployment, advanced network management and support for diverse service classes. Such slices should be E2E, involve heterogeneous cloud deployments in terms of physical and virtual resources, i.e., to utilize available edge-cloud options close to the users. For example, a CDN service may involve both edge and core clouds and deliver content through containers and OpenStack Virtual Machines (VMs), respectively. In such a setting, advanced *resource orchestration* capabilities, e.g., dynamic resource discovery, are necessary for deploying and operating multi-domain slices.

A main challenge of 5G networks is to demonstrate the opportunities they bring to vertical sectors, such as M&E. In this context, there is a need for flexible, realistic and holistic experimentation of CDN services, involving both *resource and service orchestration* aspects. It should be noted that content delivery, a key service for M&E, is also of high relevance to applications in other vertical markets, including the e-health, educational and advertising sectors.

2.2 Contributions and Chapter Organization

2.2.1 Contributions

In this chapter, we propose novel edge cloud orchestration facilities, namely UNIC and 5G-CDN, investigating different aspects such as elasticity, scalability, performance optimization, heterogeneity as well as novel mechanisms, that utilize lightweight Unikernel-based Virtual Machines (VMs) in edge cloud environments.

2.2.1.1 Unikernel-based lightweight cloud technologies and Change-Point Analysis in CDN environments

Here, we propose a novel CDN platform, called Unikernel-Based CDN (UNIC), which provides efficient content caching through placing Micro Content-Proxies (MCPs) with popular content near the users. In comparison to traditional CDN solutions, UNIC provides the following advantages: (i) it can operate over heterogeneous hardware devices with diverse capabilities, including lightweight edge clouds; (ii) it provides significant elasticity capabilities through tiny VMs orchestration; (iii) it supports modular extensibility with new mechanisms; and (iv) it defines new research problems emerging from bringing together the content-caching approaches (e.g., [17],[18]) with the VM orchestration proposals.

Furthermore, UNIC supports the following novel features:

- modular orchestration of Unikernel-based VMs hosting replicas of Internet content (i.e., the MCPs), such as for configurable VM placement;
- content popularity changes detection mechanisms that drive the MCPs deployment based on a novel Change-Point Detection (CPD) methodology tailored to the specific problem;
- dynamic load balancing using a bespoke DNS service attached to the VM orchestration; and
- real-time monitoring of server resource utilization and end-user performance.

We designed and implemented UNIC towards focusing on its two core features: (i) the modular VM orchestration (e.g., placement); and (ii) the detection of content popularity changes. We tested our CPD methodology and experimented with UNIC using real youtube popularity measurements [19] to drive end-user content demands, as an approach to early detect changes in traffic and server resource utilization.

2.2.1.2 Multi-homing with Unikernel-based lightweight cloud technologies in mobile environments

Next, we extended the UNIC platform towards integrating high-mobile environments with real operational Mobile Broadband (MBB) connectivity, investigating additional network orchestration aspects (i.e., multi-homing connectivity control and IoT routing protocol adjustments) to improve the mobile end-users' QoE through utilizing lightweight virtualization technologies. In summary, we extended the UNIC solution towards the following novel aspects:

- realized intelligent and modular orchestration for both cloud (e.g., efficient virtual machine placement) and network aspects (e.g., dynamic load balancing and multi-homing connectivity control);
- supported heterogeneous lightweight virtualization in the network edge;
- exploited mobile clients that reside at real moving buses; and
- performed multi-homing capabilities and protocol adjustments for the mobile nodes and IoT devices, respectively.

The UNIC platform realizes and supports such features through exploiting the capabilities of the unique Experimentation-As-a-Service facilities provided by the MONROE platform [20], which enables highly-mobile environments (i.e. moving bus, trains and tracks), access to real multiple Mobile Broadband (MBB) networks and extracts mobile node measurements with particular characteristics, e.g., from actively moving nodes.

2.2.1.3 A Multi-domain Experimentation Environment for 5G Media Verticals

Furthermore, we proposed the 5G-CDN platform, a novel experimentation facility for large-scale, multi-domain E2E slicing for media content delivery. It is built on top of the Fed4FIRE+ to exploit its assets of large-scale experimentation with heterogeneous hardware, resource-facing slicing, as well as enables relevant automation (e.g., dynamic resource discovery and service-aware E2E slice establishment), covering both service and virtualization aspects. It also abstracts multiple virtualization technologies and resource specifications (i.e., applying uniform representation of resources). Finally, the proposed platform refines a high-level service definition to lower-level slice specifications, while supporting modular CDN service orchestration.

In a nutshell, the 5G-CDN platform supports the following technical enablers:

- A GUI and YAML [21] schema realizing a high-level definition of CDN experiments and modular extensibility of resource and content-delivery service orchestration algorithms through the NodeRED tool [22].

- A novel architecture, appropriate abstractions and interfaces that: (i) bridge the gap between CDN-based services, important 5G features, and general-purpose test-bed federations through a holistic approach; (ii) handle and hide the complexity of heterogeneous physical and virtual resources; and (iii) perform scalable dynamic resource allocation and discovery over both federated and local test-bed resources.

- Its novel architecture, through utilizing Fed4FIRE+ facilities [23], enables: (i) a representation of the multi-domain infrastructure providers through the federated Fed4FIRE+ test-beds providing hardware resources around the globe, including physical servers and 5G networking equipment; (ii) a basis for 5G network slicing through the Fed4FIRE+ experimenters' slicing, in the sense of putting together physical resources from different test-beds, called slivers, to implement a particular service for experimentation; (iii) automations in the resource discovery and allocation through the Fed4FIRE+ test-bed control tools (e.g., jFed CLI).

We conducted a number of experiments with the 5G-CDN platform. Our experimental results highlight the capabilities of the 5G-CDN to: (i) realize large-scale experimentation involving regular and lightweight clouds; (ii) dynamically discover and allocate resources for the CDN service deployment; (iii) implement E2E slices over geographical distributed heterogeneous physical and virtual resources utilizing alternative Unikernel technologies; (iv) define, in a flexible and modular manner, new experiments with alternative resource and service orchestration algorithms for CDNs.

2.2.2 Chapter Organization

The remainder of the Chapter is organized as follows. Section 2.3 provides an overview of the related topics. Section 2.5 details the UNIC platform's architecture and highlights its core mechanisms. Section 2.6 elaborates on our experimentation methodology regarding the UNIC platform and Section 2.7 presents our experimental results, demonstrating the full UNIC system operation. Section 2.8 elaborates the extended features of the UNIC platform, along with the relevant experimental results. Next, Section 2.10 discusses the design and implementation details of the proposed 5G-CDN platform, while section 2.11 presents our evaluation results for each independent experimental scenario. Finally, section 2.12 concludes the chapter.

2.3 Related Works

In this subsection, we contrast our proposed facilities (i.e., UNIC and 5G-CDN) to related works mainly focused on relevant CDN platforms implementing: (i) resource-efficient operation utilizing unikernels; and (ii) E2E slicing over novel federated distrubuted infrastructures and experimentation for vertical services, especially media-related.

2.3.1 Relevant CDN Architectures for Resource-Efficient Allocation driven by CPD mechanisms with unikernels technologies.

A massive amount of the Internet traffic consists of content distributed by major providers such as YouTube, Netflix and Amazon. This bulky traffic is usually delivered to users through a Content Delivery Network (CDN) [24] [25], which is estimated to scale up to 71% of the Internet traffic by 2023 [26]. Furthermore, a crucial issue is on how 5G emerging opportunities would affect CDNs.

However, CDNs are usually tightly coupled with cloud providers (e.g., Akamai [27], Microsoft Azure [28], Amazon [29]) that use their own hardware, sometimes customized (e.g., NetApp's FlexCase). The CDN software may be proprietary, costly for SMEs and with specific hardware or OS requirements. Such approaches deliver transparently and efficiently content to the end-users. However, traditional CDN servers are typically far away from the content consumers and are unsuitable for the ultra delay-sensitive applications envisaged by the 5G networking initiatives [30]. Hence, we argue that there is a need for open, flexible, extensible, hardware-independent and resource-efficient CDN solutions hosting the content near to the users, and we suggest lightweight virtualization as an enabling technology for such unique features.

The main candidates for lightweight virtualization are the Containers and the Unikernels. The Containers (e.g., Docker [31] or LXC [32]) are standardized units implementing application packaging with all of its dependencies. Unikernels [14] (e.g., Mirage0S [33], Click OS [34], Rump Kernel[35], OSv[36]) are single-purpose VMs with only the essential part of the OS for the particular service, specialized at compile-time and sealed against modification when deployed in the cloud. The Unikernels have fewer MBs in size, boot up quicker and are more secure than Containers (e.g., the latter use shared kernel space). A Unikernel-based web server can even boot transparently in milliseconds with the reception of a DNS request packet [15]. A relevant thorough comparison between Unikernels and Containers can be found in [37].

The efficient VM placement is a challenging issue in cloud computing. However, existing relevant proposals do not consider the high-dynamicity of Unikernels. For example, the survey paper [38] studies and categorizes a large number of existing VM placement approaches, but none of them consider the high-dynamicity of Unikernels.

Furthermore, our approach applies CPD mechanisms [39, 40, 41] to provide an early signal of content popularity changes and deploy new MCPs. CPD is used extensively in network anomaly detection [42, 43], e.g., for intrusions detection [44, 45]. We proposed a theoretical CPD methodology suitable for content popularity change estimation, aligned to the CDNs context.

In our understanding, the only relevant CDN platforms to ours (i.e., UNIC) are [46],[47]. The MOSTO platform [46] deploys Unikernels as TCP proxies (i.e., to improve TCP's slow-start algorithm performance). An interesting CDN solution with impressive performance is [47], which provides content through Click OS Unikernels [34] and is evaluated with a CDN simulator [48]. In contrast to [47], which focuses on performance aspects, our apporace: (i) considers heterogeneity in terms of virtualization and Unikernel technology; (ii) conducts real experimentation; and (iii) achieves flexibility through Unikernel-oriented VM placement driven by novel early content popularity change detection. However, the two approaches could be potentially synchronized (i.e., support Click OS in our solution).

2.3.2 Relevant federated infrastructures for E2E Network Slicing

Relevant experimentation environments exploiting Fed4FIRE+ or GENI capabilities as 5G-CDN are: (i) FUTEBOL [49] integrating wireless and optical domains over European and Brazilian test-beds; (ii) SoftFIRE [1] an SDN/NFV test-bed supporting high-level service definition based on TOSCA [50] and resource discovery; and (iii) 5GinFIRE [51] a 5G test-bed targeting multiple vertical industries (one of the 5GinFIRE test-beds, the TNO 5G Media Vertical, focuses on the Media Vertical industry).

5G platforms with inherent 5G E2E slicing capabilities include: (i) the 5G-VINNI[2] targeting particular 5G KPIs and multiple verticals; (ii) the 5G-PAGODA [52] investigating NFV-based E2E slicing over two test-beds in Europe and Japan; and (iii) 5G-EVE [3] which is an experimentation-oriented platform for multiple verticals, supporting a cross-facility 5G catalogue and multi-domain orchestration. Other proposals focus on RAN slicing aspects (e.g., [53]). The 5G-MEDIA [54] project and platform targets the Media Verticals through multiple use-cases, focus on SDN/NFV aspects and supports a DevOps environment for media applications, hiding the complexity of service development and deployment over 5G infrastructures.

To the best of our knowledge, 5G-CDN, in contrast to the relevant approaches, is the first 5G platform, tailored for CDN services, exploiting the novel Fed4FIRE+ experimentation capabilities, while addressing multi-domain operation, modular service and resource orchestration, scalability and heterogeneity aspects. Our approach does not focus on RAN slicing, but it can incorporate relevant capabilities offered by existing Fed4FIRE+ test-beds, such as the LTE slicing feature of the NITOS test-bed [55].

2.4 Content Distribution Networks (CDNs) for 5G Networks

The emerging 5G networks call for new approaches to CDNs through addressing challenging issues, such as: (i) scalable and holistic resource management, spanning from large data centers to the user device, including edge clouds; (ii) incorporation of heterogeneous physical and virtual resources; (iii) extensibility to support new capabilities

[1]http://www.softfire.eu
[2]http://5g-vinni.eu
[3]http://5g-eve.eu

or mechanisms; and (iv) adaptability to dynamic user requirements, server resources and network capacity constraints. User-generated content is the driving force for new services, while edge cloud solutions are being proposed to host (or cache) content locally. However, it is costly to deploy traditional clouds near end-users and such virtual machines (VMs) are inefficient for dynamic network conditions (i.e., may boot-up in minutes).

2.5 Implementing elasticity CDN based on Unikernels

In the following subsection, we propose an elastic content distribution platform, that serves the Internet content using tiny Unikernel-based VMs. We named the latter Micro Content-Proxies (MCPs). Such VMs host one or a few videos, that can appear rapidly in nearby cloud deployments, serve users and then disappear. In other words, the studied environment provides content dissemination through very dynamic, almost "fluid" VM placement, since the content is packaged with the server software with just a minor increase in size. So, we reposition the content caching and provisioning as a VM orchestration problem. We demonstrate the complete implementation of the proposed platform with proof-of-concept experimental results.

2.5.1 The UNIC Platform Architecture

UNIC is an intelligent lightweight cloud orchestration platform, that provides efficient content distribution to the end-users through MCPs scattered to a cloud hierarchy (i.e., with both regular and lightweight clouds). The UNIC platform realizes flexible and scalable content distribution over heterogeneous virtual and physical resources. We focus on two main UNIC aspects here: (i) the modular VM placement algorithms considering real-time server resource utilization and content provisioning requirements; and (ii) a novel approach to early content popularity change detection driving VM orchestration, based on our CPD methodology, i.e., Cumulative Sum (CUSUM) procedures efficiently combining off-line and on-line CPD, mechanisms revealing the direction of changes and an improved time-series segmentation algorithm, to detect multiple changes.

Here, we give a bottom-up description of the UNIC platform architecture (i.e., Figure 1), which consists of the following three main layers:

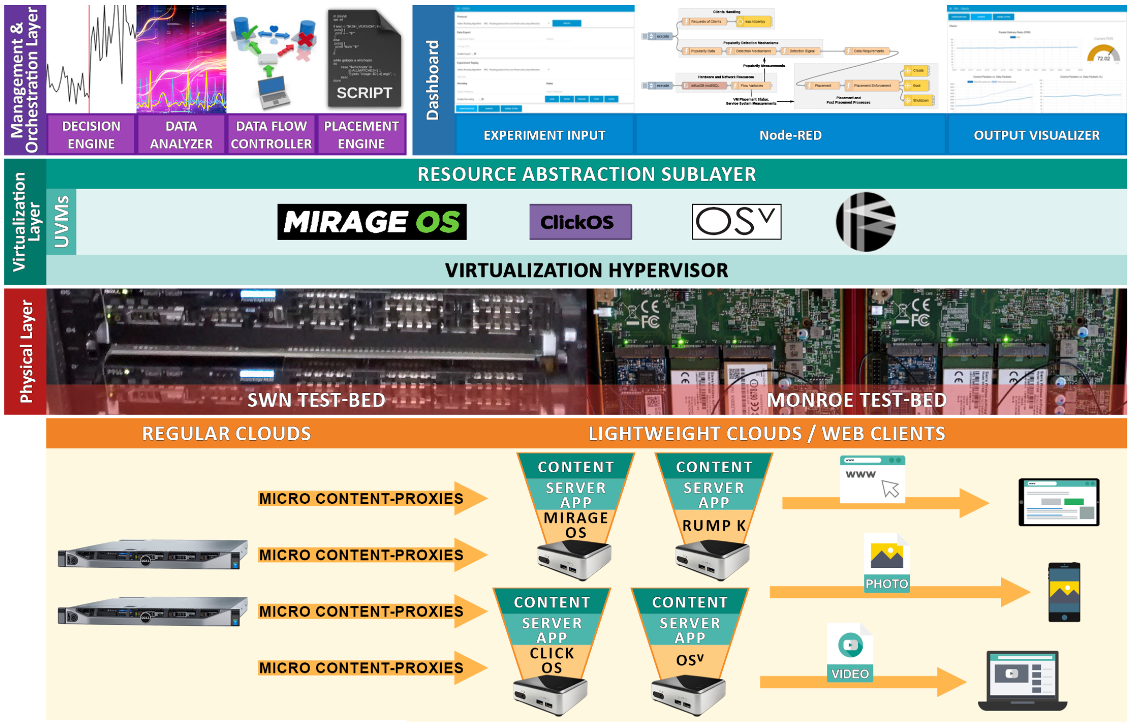


Figure 1: The Architecture of UNIC Platform

a) The *Physical Layer* utilizes federated hardware, providing the required heterogeneity and scalability aspects. More precisely, we use our own SWN test-beds [56]. The SWN test-bed provides the regular and lightweight cloud facilities and hosts the end-users.
b) The *Virtualization Layer* supports lightweight cloud capabilities and multiple Unikernel technologies (e.g., Mirage OS, Rump Kernel or Click OS). The UNIC architecture is independent from the virtualization technologies used; hence, carefully designed abstractions hide the virtualization heterogeneity (i.e., the *Resource Abstraction Sublayer* exports a uniform interface for VM control).
c) The *Management and Orchestration Layer* controls and orchestrates the UNIC platform and test-beds, including providing the efficient VM placement through the *Placement Engine*, the traffic control through the *Data Flow Controller* and network analytics, that enable intelligent network configuration decisions through the *Data Analyzer* and the *Decision Engine*, respectively.

As shown in Figure 1, the *UNIC dashboard* provides the experimentation input, results visualization and modular extensibility of the evaluated mechanisms through the Node-RED tool [22]. An important UNIC aspect is the ability to implement VM orches-

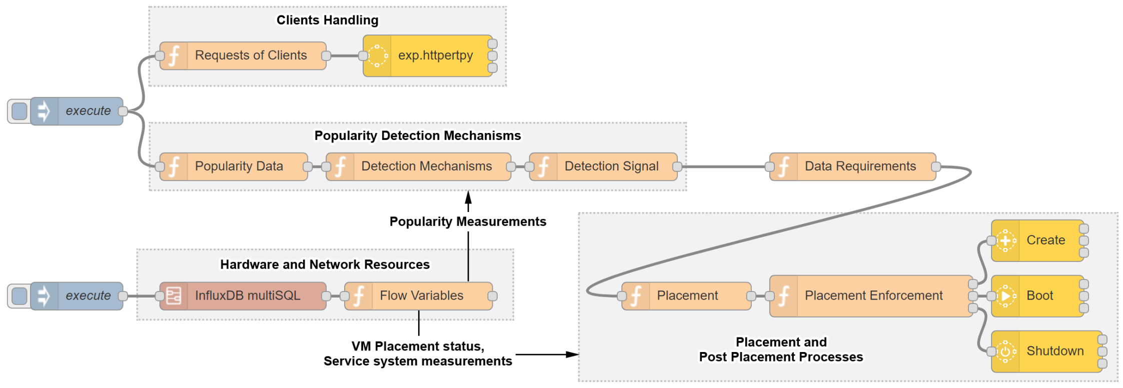


Figure 2: Platform's orchestration & management features implemented as Node-RED nodes

tration processes in the form of Node-RED work-flows, in a plug-and-play fashion. As shown in Fig. 2, all the UNIC *Management and Orchestration Layer* components have been implemented in the form of Node-RED nodes, such as: (i) the content popularity detection for the *Decision Engine*; (ii) the VM placement functions for the *Placement Engine*; (iii) the content popularity and web client performance monitoring for the *Network Analytics*; and (iv) the traffic load balancing (i.e., our own dynamic DNS server matching content replicas with web clients) for the *Data Flow Controller*. These nodes are standalone components that can be manipulated / configured independently of each other and be connected to form complete VM orchestration processes.

To further analyze the UNIC platform, we describe two of its core features in the following subsections, i.e., the content popularity changes detection and the modular VM placement mechanisms.

2.5.1.1 Content Popularity Changes Detection

The UNIC platform detects early changes in the content popularity and signals new MCP placements, in case of an upward qualitative change in the content popularity or a removal of MCPs in case of a downward change (i.e., handled from the VM placement algorithms of subsection 2.5.1.2). Such decisions are being communicated to a dynamic DNS server assigning end-users to the active content caches, in a round-robin fashion.

We apply a novel statistical change point analysis methodology to approach the content popularity detection problem introduced in [57]. Such mechanisms target the following requirements: (i) low-complexity and quick estimations to match the dynamic and resource-constraint nature of Unikernels; (ii) to rely on a non-parametric framework to avoid restrictive assumptions (i.e., no particular model or distribution can fit a 'mixed content' provisioning and a large number of model parameters may lead to high convergence times); and (iii) to operate in an on-line manner and be able to estimate the existence, the direction and magnitude of changes.

To address the above requirements, we detect the existence of a change in the historical sequence of content views' observations using a retrospective test statistic for the unknown time of change in the mean [58]. Such procedure consists of a CUSUM based detector and the Newey-West long-run variance estimator [59] to capture the serial dependence between observations. We combine the method [58] with a new heuristic, incorporating the two binary segmentation algorithms [60], [61], to be able to detect multiple change points (i.e., through a segmentation of the time-series). We apply the Moving Average Convergence/Divergence (MACD) trend indicator [62] to estimate the direction of detected changes.

Regarding our on-line CPD mechanism, we implement a stopping-time procedure [63], based on a mean CUSUM detector. The stopping rule depends on a sensitivity parameter $\gamma \in [0, \frac{1}{2})$. For example, a $\gamma \ll 0.25$ allows slower but more accurate detections, in terms of type I errors (i.e., incorrect rejection of the null hypothesis of no change) and $\gamma \gg 0.25$ leads to quicker but more sensitive to false alarms detections.

We outline our main content change-point detection algorithm, assuming that the monitoring period starts at the arbitrary time $t = m.s$, as follows:

- **Step 0:** Define *a priori* a finite monitoring window $l > 0$, and denote the monitoring horizon as $m.h = m.s + l$.

- **Step 1:** Apply the segmentation algorithm supplied by the retrospective statistic

R_h for the whole historical period $h = [1, m.s]$, or for the bounded interval (i.e., to experiment with smaller monitoring periods):
$h = [w, m.s],\ w > 0$,

- if no changes are detected, the training sample of the sequential procedure becomes $m = h$,

- else the training sample becomes $m = [cp_{last}, m.s]$, where cp_{last} is the detected time of change.

- **Step 2:** Apply the sequential procedure $S(m,k),\ k \geq m.s$,

  - if a change is detected for some $m.s \leq k_{cp} \leq m.h$, the procedure stops,

  - else if no change is occurred after $m.h$ observations set $k_{cp} = 0$ and the monitoring terminates, i.e., proceed to **Step 4**.

- **Step 3:** If $k_{cp} \neq 0$, define k_{cp} as a change-point and apply the trend indicator at the time of change $TI(k_{cp})$,

  - if $TI(k_{cp}) > 0$ then deploy a Unikernel (i.e., upward change),

  - else, displace a Unikernel (i.e., downward change).

- **Step 4:** Set a new starting point for the monitoring period,

  - if $k_{cp} > 0$, set $m.s = k_{cp} + d$, where d is a constant value defining a period assuming no change,

  - else, set $m.s = m.h$.

We maintain two parallel change-point detection processes with different significance level α and parameter γ values. We place one more Unikernel, in case the change is detected from both processes, otherwise, we may remove one or two, in a similar way. The MCP placements and removals are being handled from the algorithms described in the following subsection.

2.5.1.2 Modular VM Placement

As discussed above, UNIC supports modular extensibility of VM orchestration functionalities in the form of independent software entities, called Node-RED nodes. We exploit such capability to implement three alternative VM placement mechanisms: (i) a *Random Placement* algorithm, choosing randomly one of the available physical nodes and providing reference measurements; (ii) a *Quantity-Based*, choosing the physical node each time hosting the lower number of VMs; and (iii) an *Objective Weight Function (OWF)* approach, considering the real-time CPU, RAM and network utilization measurements of each node, weighted by particular coefficients (i.e., tuning the involved performance trade-offs).

Since the first two algorithms are self-explanatory, we only detail the *OWF* algorithm only. We consider as $PN = \{pn_1, ..., pn_i\}$ the set of available physical nodes (PN) to host MCPs, where i is the PN number. The variables cpu_{np_i}, mem_{np_i}, tt_{np_i} and rt_{np_i} represent the percentage of CPU utilization (i.e., the average values of available CPU cores), memory allocation(MEM), outgoing(TT) and incoming(RT) link utilization, respectively.

Furthermore, the assignment of MCPs to a particular PN cannot lead to exceeding the available capacity resource (i.e., they have available resources to host one more VM). Consequently, we define the following constraint:

$$0 \leq cpu_{pn_i}, mem_{pn_i}, tt_{pn_i}, rt_{pn_i} < 1 \tag{1}$$

The *OWF* executes periodically and triggers MPCs deployment or removals in the event that content popularity changes detection mechanisms detect a qualitative change in the content popularity. The *OWF* place the MPCs on the i^{th} Physical node PN that provides the minimum resource utilization, given by:

$$\min_{pn_i \in PN} \alpha cpu_{pn_i} + \beta mem_{pn_i} + \gamma tt_{pn_i} + \delta rt_{pn_i} \tag{2}$$

$$\text{s.t.} \quad (1)$$

The coefficients α, β, γ, $\delta \geq 0$ weight the importance of each resource type, e.g., to match particular application-level requirements.

2.6 Experimental Methodology

In this subsection, we describe our experimental methodology, including the details and configurations of main platform implementation aspects along with the test-bed we utilized to carry out our experiments.

We conduct real experiments that require the implementation and configuration of separate technical features, such as: (i) the VM orchestration; (ii) the content popularity detection mechanisms; (iii) the end-user traffic emulation and control; and (iv) the physical server resource utilization and end-user performance monitoring. We briefly outline each technical aspect below, i.e., configuration parameters, basic implementation details or open-source tools we used.

The VM Orchestration: We create lightweight web-servers delivering content with Mirage OS Unikernels. Such tiny VMs are being orchestrated according to a work-flow diagram created through the Node-RED tool. Such work-flow defines the communication of independent software entities, i.e., the Node-RED nodes. We create one node per VM orchestration process (e.g., the VM deployment, the placement decision making, etc). An experimenter can introduce new nodes (e.g., placement algorithms) or connect them in alternative ways (e.g., to create new orchestration work-flows). All processes communicate with the hypervisor through the *Resource Abstraction Sublayer (RAS)*, exposing a unified north interface to the orchestration features but virtualization-technology specific south interfaces. The latter communicates through ansible scripts [64]. *RAS* allows us to introduce heterogeneous virtualization technologies, in the near future.

The Content Popularity Detection Mechanism: We implement the CPD mechanisms in Matlab. To ensure an online operation, we create at a separate host a Matlab TCP server application that receives periodic content popularity measurements and returns a notification for each detected change-point and an estimation of its basic characteristics (i.e., direction and rough magnitude). The other end is a Node-RED node that triggers VM deployments or removals.

The end-user traffic emulation and control: We emulate the web-clients using the httperf open-source tool [65]. We create and deploy web-clients based on real content popularity measurements extracted from youtube with the tool [19]. In Figure 3, we show the content requests per minute that we use as an input for the emulation of web clients in these experiments. The duration of the particular measurements matches the duration of the experiments, i.e., 310 minutes for each run. We create a DNS-based load-balancing Node-RED node that keeps track and redirects the web requests to particular content caches (i.e., Unikernel VMs), in a round-robin fashion.

The physical server resource utilization and end-user performance monitoring: We are monitoring the servers' resource utilization, in terms of CPU, memory, incoming / outgoing traffic and the performance of web-clients using the open-source tool CollectD [66]. We store the measurements in InfluxDB [67], a time-series database, and visualize them with the Grafana tool [68]. The measurements reach to the orchestration processes that take informed decisions for the context environment, e.g., to the VM placement algorithms. The monitoring takes place at regular time intervals (i.e., every 10 secs).

We use our own SWN test-bed to conduct the experiments. More precisely, we use: (i) seven physical servers, five to host the MCPs, one as a Management and Orchestration server and one to host the CPD mechanisms; (ii) ten Raspberry PIs to emulate the web-clients requesting web content from the MCPs; and (iii) one L3 100Mbps switch.

2.7 Experimental Results

We group our experimental runs into two scenarios. The first scenario investigates the impact of early content popularity change detection driving by VM orchestration, while the second investigates the impact of placement algorithm utilizing real-time server resource utilization measurements. For both scenarios: (i) we assume a running operation of UNIC with three MCPs hosting particular web content; and (ii) we allocate web-clients based on the real content popularity traces illustrated in Figure 3. We see

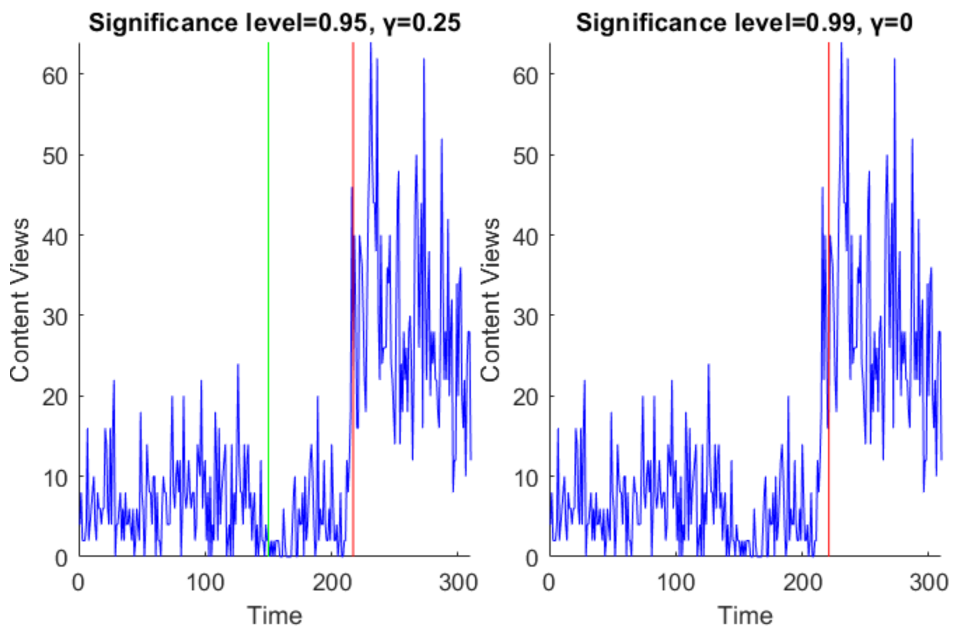


Figure 3: Content-views per minute of a particular youtube video and detected change-points for different α and γ values. Red lines symbolize the upward and downward change.

there is a small change (i.e., reduction) in the end-user demand to watch the particular video at around the 150 minute of the experiment and a significant change after the 220.

We apply two CPD processes in parallel with variable sensitivity, i.e., a more sensitive with parameters $\gamma = 0.25$, $\alpha = 0.95$ and a less sensitive with parameters $\gamma = 0$, $\alpha = 0.99$, as shown in Figure 3. In case both of them estimate a change point at the same time interval, we deploy two VMs, assuming a larger magnitude of change. In the typical case the sensitive approach detects a change but not the less sensitive one, we deploy one VM.

We set the *OWF* algorithm's coefficients α, β, γ, $\delta \geq 0$ to the values 60%, 30%, 5%, 5%, respectively. In the following figures (i.e., 4 to 9), the different colors represent measurements from different physical machines.

2.7.1 Scenario 1: Impact of change-point detection mechanisms

To evaluate the impact of the change-point detection mechanisms, we run the experiment twice, one with the CPD mechanisms enabled and one with them disabled.

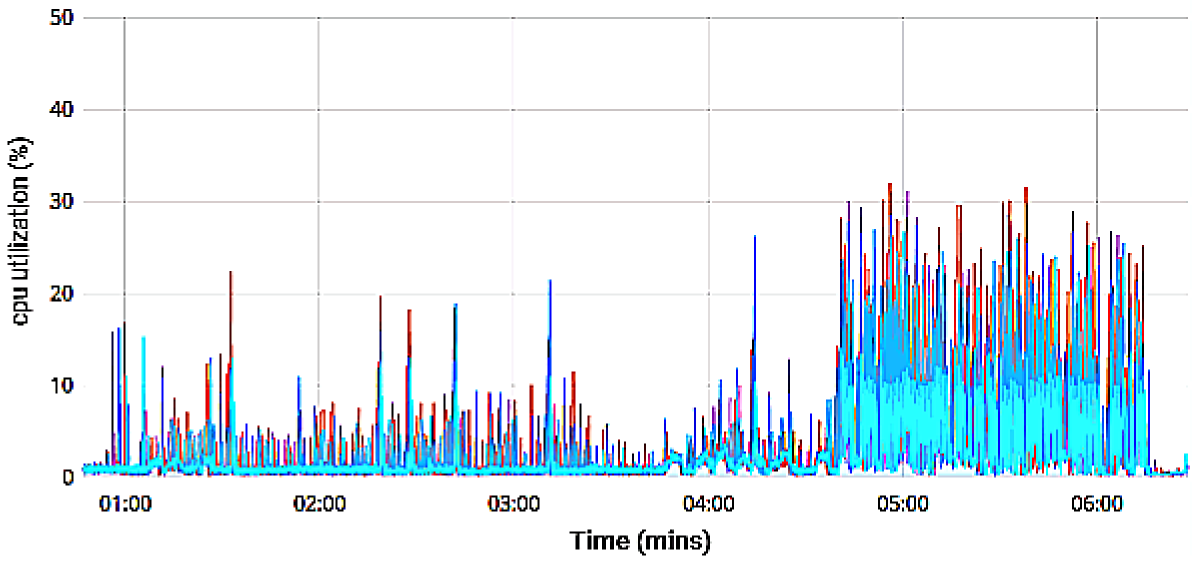


Figure 4: The servers' CPU utilization with the change-point detection mechanisms disabled, using the Objective Weight Function placement algorithm

The Figures 4 and 6 contrast the percentage of CPU utilization and Figures 5 and 7 the percentage of memory allocation per physical server, with the change-point

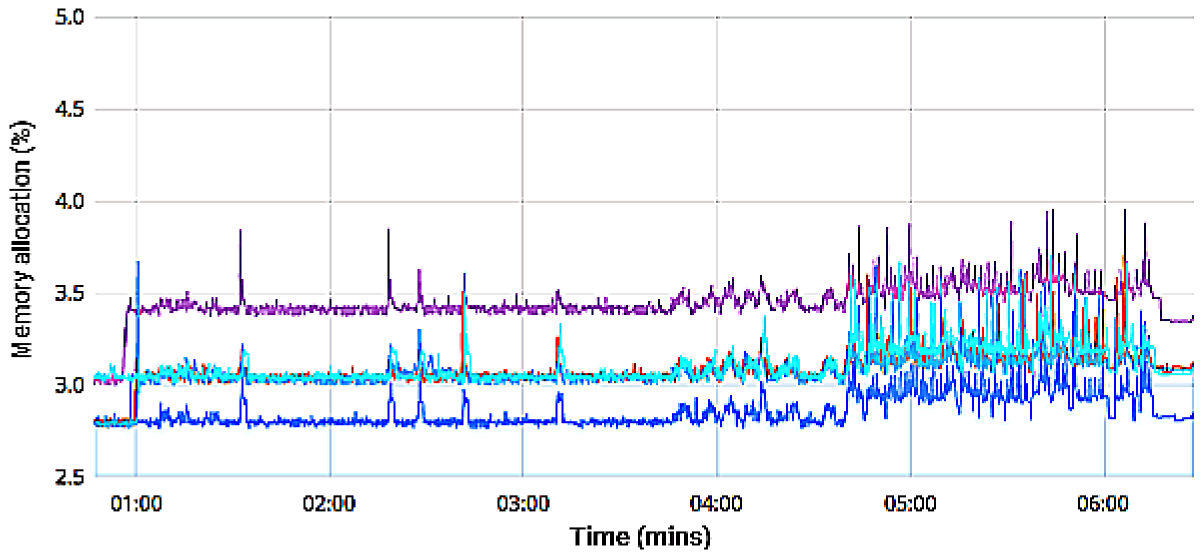


Figure 5: The servers' memory allocation with the change-point detection mechanisms disabled, using the Objective Weight Function placement algorithm

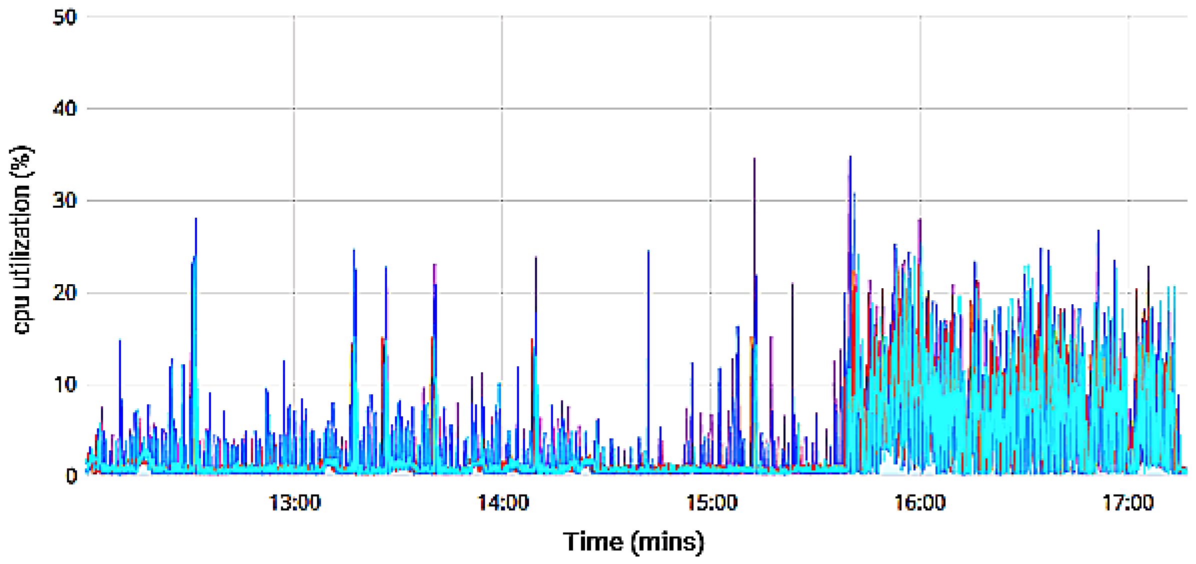


Figure 6: The servers' CPU utilization with the change-point detection mechanisms enabled, using the Objective Weight Function placement algorithm

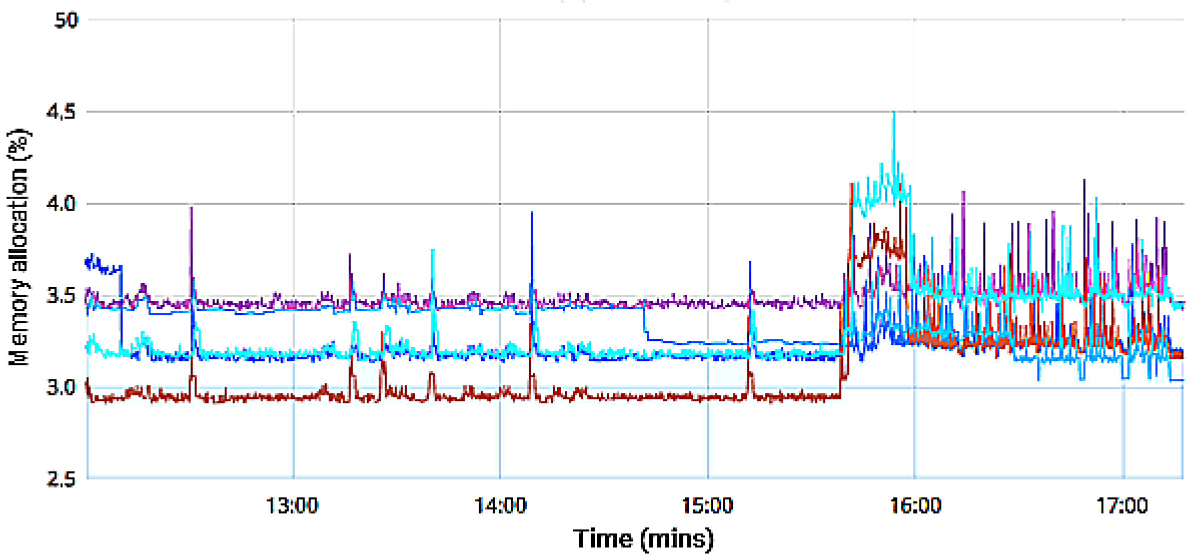


Figure 7: The servers' memory allocation with the change-point detection mechanisms enabled, using the Objective Weight Function placement algorithm

detection mechanisms disabled and enabled, respectively. According to these figures, we have the following observations:

- for the first time period (i.e., 0 to 220 minutes), the CPU and memory allocation are similar for both cases.
- for the second time period (i.e., 220 to 310 minutes), the maximum CPU utilization is reduced around 10%, while there is a 0.5% increase in the memory allocation.

Such outcome can be explained as follows. In the second experimental run, the change-point detection mechanisms take the decision to boot two more MCPs due to the abrupt increase of content views (i.e., both CPD processes detect the change, as shown in Figure 3). This decision reduces the CPU utilization, but has a minor impact on the memory allocation (i.e., due to the additional VM deployment). This is consistent with the content popularity traces used for the web-clients deployment, dictated by the real measurements, where there is a change-point at the same time-period (i.e., see Figure 3). We note that the smaller change-point, detected from the sensitive CPD process only, leads to the removal of an MCP, without significant impact on both CPU utilization and Memory.

2.7.2 Scenario 2: Impact of VM placement algorithms

To evaluate the impact of the placement algorithms, we execute a representative experiment three times, one for each placement algorithm discussed in Section 2.5.1.2. Due to the homogeneous nature of our SWN test-bed, we omit the results of the *Quantity-based* algorithm since it often produced similar results with *OWF* algorithm. The *OWF* mechanism considers the real-time measurements of all available physical machines. In all cases the change point detection mechanisms are enabled.

Figures 6 and 8 highlight the impact of the placement algorithm on the CPU utilization, while Figures 7 and 9 highlight the same impact on the memory allocation of physical servers. According to these four figures, we observe the following:

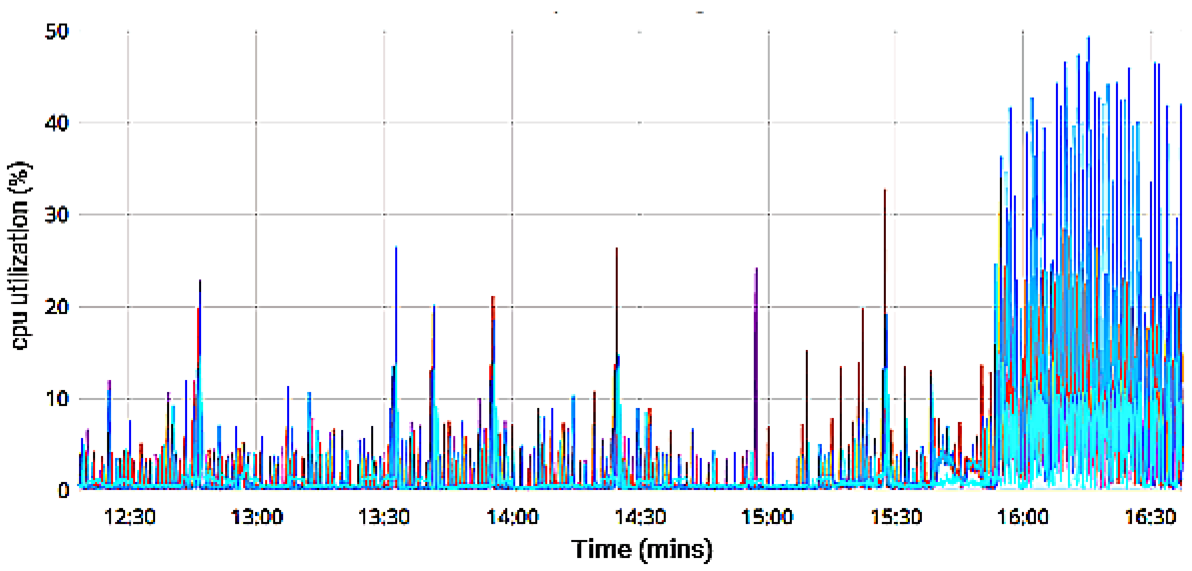


Figure 8: The servers' CPU utilization with the change-point detection mechanisms enabled, using the Random placement algorithm

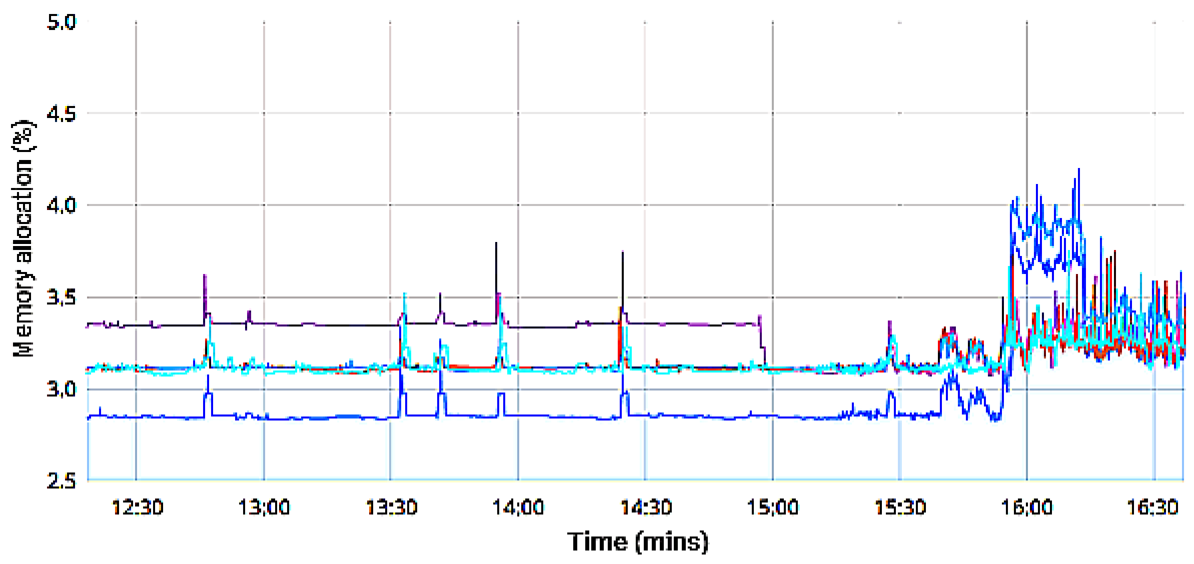


Figure 9: The servers' memory allocation with the change-point detection mechanisms enabled, using the Random placement algorithm

- for the first time period (i.e., 0 to 220 minutes), the CPU and memory measurements scale at the same low-levels, for both placement algorithms (i.e., there are not very many content requests, as shown in Figure 3).
- for the second time period (i.e., 220 to 310 minutes), the *OWF* placement algorithm causes the consumption of at least 10% less maximum CPU allocation, which even reaches the 30% at some points. This without significant changes in the memory allocation.

The above outcome can be justified by the choice of the *Random* placement algorithm to boot one additional MCP in a server that hosts other VMs as well. This result calls for further investigations in the sophistication of the placement algorithms.

Our results show that orchestrating network resources with lightweight clouds brings advantages in users' QoE.

2.8 Extended UNIC platform capabilities for mobile IoT environments

In the context of the MEC research project, we investigated and extended the UNIC capabilities towards integrating and optimizing together a real mobile environment with lightweight cloud virtualization resources (e.g., Unikernels). More precisely, the extended UNIC platform brings together: (i) heterogeneous lightweight cloud technologies hosting the content and IoT data; (ii) multi-homing capabilities in the mobile nodes, that select the best connectivity option based on the service requirements related to the service used e.g., for low latency or high throughput; and (iv) IoT routing protocol adjustments, that reduce the delays of measurements' collection.

Fig. 10 illustrates a high-level overview of the extended UNIC architecture. We briefly elaborate on the three-layer structure as well as the corresponding components (i.e., as a detailed description has been discussed in section 2.5.1). In a bottom-up approach, we highlight the following three layers: (i) the *Physical Layer* consisting of the lightweight edge clouds, IoT devices (i.e., federated infrastructure facilities utilizing Monroe and our SWN test-bed), that collect measurements, and mobile clients with

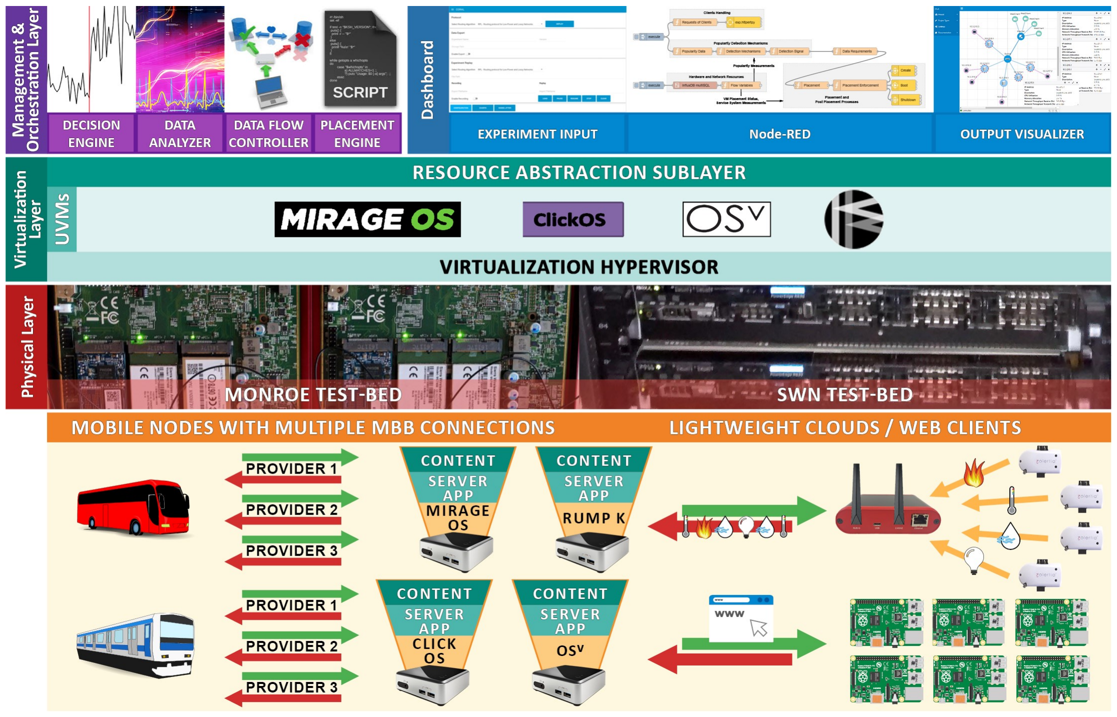


Figure 10: The extended Architecture of UNIC Platform

multi-homing capabilities; (ii) the *Virtualization Layer* enabling lightweight cloud capabilities through Virtual Machines (VMs) with “tiny” operating systems, such as the Mirage OS and Rump Kernel Unikernel technologies. A Resource Abstraction Sublayer (RAS) hides virtualization heterogeneity and exports a uniform interface for VM control; and (iii) the *Orchestration Layer* with the following features: a Data analyzer, which performs CP detection to early “track” changes in the content evolution, a Decision engine, which specifies either to deploy or remove lightweight VMs, accommodating content and IoT data near the end-users, a Data flow controller, that balances the traffic load among active VMs, and a Placement engine, which determines the optimal location of the VMs. The UNIC dashboard takes the experimentation input through the Node-RED tool and provides the corresponding visualization results.

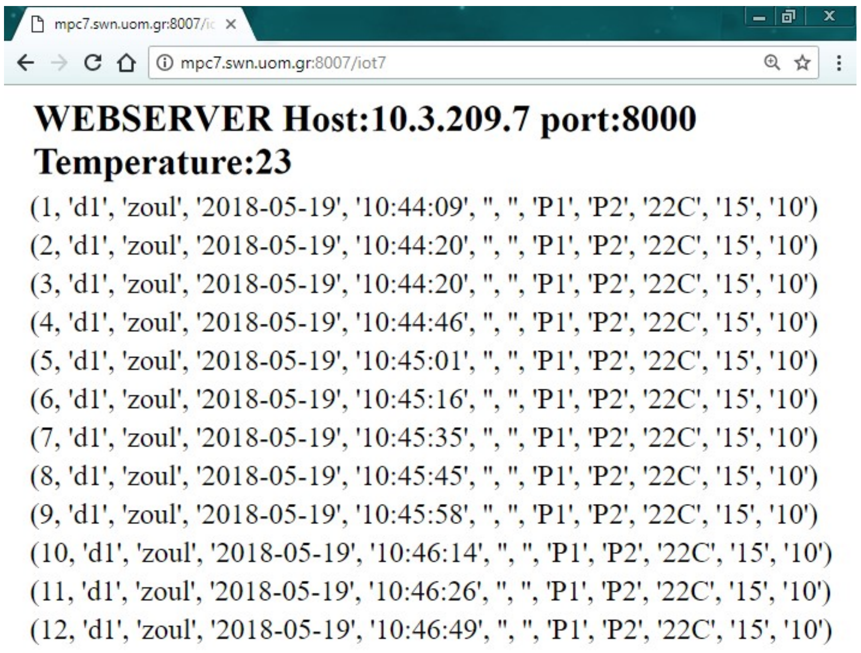
In practice, the novel new feature in the extended UNIC platform is the *Multi-homing mechanism* which is an independent software entity deployed on real movies vehicles (i.e., *Physical Layer*). To realize the *Multi-homing mechanism* features, we implemented a multi-threaded measurement tool that is constantly measuring all interfaces (i.e., alternative connectivity options) in terms of delay and bandwidth (e.g. by periodic "ping" and HTTP downloading), and it is equipped with a decision mechanism, that dynamically and transparently switches to the most appropriate provider to improve the mobile broadband (MBB) connectivity of mobile users. The selection is based on the recent average measurement and according to application requirements.

2.8.1 Experimental results of multi-homed network utilizing lightweight cloud resources

In this section, we evaluate the new novel multi-homing features as well as the lightweight capabilities, that UNIC platform brings through assuming the following IoT application scenario.

We assume that a medical doctor (i.e, mobile user) accesses up-to-date IoT vital-signs of his patients (i.e. fixed IoT devices) using his smartphone. The mobile user(s) (i.e., medical doctor) resides in MONROE mobile bus node equipped with our *Multi-homing mechanism* features. The latter mobile node may suffer from MBB connection inefficiencies and delays due to the overloaded cloud. To further reduce the involved delays and assist the network communication between users and IoT devices, we utilize nearby edge clouds with real caching IoT measurements (e.g., temperature, pressure, vital-signs, etc.). The latter in form of Unikernel-based VMs act as data collectors, that store measurements in a lightweight DB (i.e., Fig. 11 shows an example Unikernel caching real IoT measurements). In practice, the fog and edge cloud is dwelling in our facility involving five Low-end PCs, hosting the Unikernel-based VMs and five Zolertia RE-Mote 2 motes (i.e., IoT nodes produce real IoT measurements). The Unikernel based VMs are being deployed at the most appropriate physical host utilizing the management and orchestration layer through using the Placement Engine.

In such emergency service it is essential that the up-to-date measurements' are collected with ultra low-latency (i.e., crucial application requirement). With that perspective the *multi-homing mechanism* is switched to the best of the available MBB providers based on dealy measurements (i.e., ping measurements) rather than throughput (i.e., the throughput measurements will be ideal with an elastic content-distribute network case as described in section 2.7).



```
WEBSERVER Host:10.3.209.7 port:8000
Temperature:23
(1, 'd1', 'zoul', '2018-05-19', '10:44:09', '', '', 'P1', 'P2', '22C', '15', '10')
(2, 'd1', 'zoul', '2018-05-19', '10:44:20', '', '', 'P1', 'P2', '22C', '15', '10')
(3, 'd1', 'zoul', '2018-05-19', '10:44:20', '', '', 'P1', 'P2', '22C', '15', '10')
(4, 'd1', 'zoul', '2018-05-19', '10:44:46', '', '', 'P1', 'P2', '22C', '15', '10')
(5, 'd1', 'zoul', '2018-05-19', '10:45:01', '', '', 'P1', 'P2', '22C', '15', '10')
(6, 'd1', 'zoul', '2018-05-19', '10:45:16', '', '', 'P1', 'P2', '22C', '15', '10')
(7, 'd1', 'zoul', '2018-05-19', '10:45:35', '', '', 'P1', 'P2', '22C', '15', '10')
(8, 'd1', 'zoul', '2018-05-19', '10:45:45', '', '', 'P1', 'P2', '22C', '15', '10')
(9, 'd1', 'zoul', '2018-05-19', '10:45:58', '', '', 'P1', 'P2', '22C', '15', '10')
(10, 'd1', 'zoul', '2018-05-19', '10:46:14', '', '', 'P1', 'P2', '22C', '15', '10')
(11, 'd1', 'zoul', '2018-05-19', '10:46:26', '', '', 'P1', 'P2', '22C', '15', '10')
(12, 'd1', 'zoul', '2018-05-19', '10:46:49', '', '', 'P1', 'P2', '22C', '15', '10')
```

Figure 11: Unikernel-based web server caching real IoT measurements

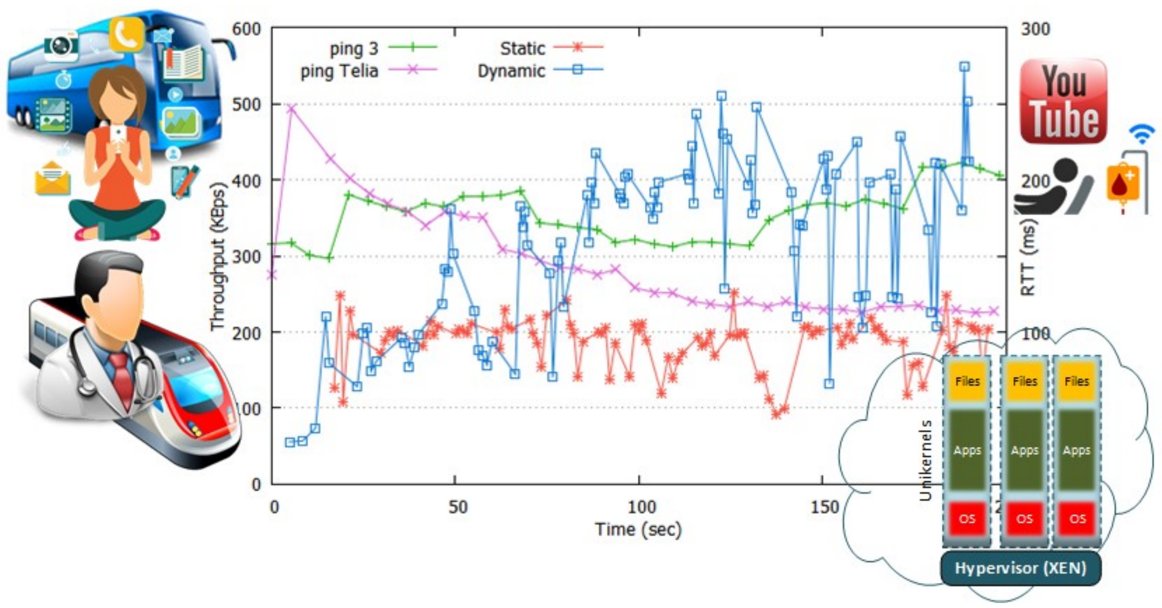


Figure 12: Multi-homing for mobile users

In practice, the mechanism periodically collects ping measurements and compares the average value of the ten recent ping measurements of the available MBB connectivities and selects dynamically the provider, that performs better in terms of delay. As we show in Fig. 12, the green and the pink curves show the periodic ping measurements of the '3' and 'Telia' Swedish providers, respectively. The mobile node starts communicating using the '3' provider and as soon as the average ping measurements of the '3' provider are higher than those of the 'Telia' provider, the mechanism switches to the latter. As the blue curve shows, this dynamic behavior achieves constantly the best performance. Using one provider for the whole communication (i.e., red curve), is not the most efficient choice.

As a bottom line, our results show that our proposed platform brings advantages to mobile users' QoE by exploiting multi-homing capabilities along with lightweight virtualization resources. A live demo, highlighting the novel aspects of the extended UNIC platform, can be found here[4].

2.8.2 Synergy of multi-homing capabilities along with adaptable network protocols in IoT environments

Our experiment results show that multi-homing capabilities improve the connectivity of mobile users. In the context of the MEC project, our research team initiates an early investigation study involving mobile IoT devices in our IoT application scenario. However such networks topologies (i.e., with fixed and mobile IoT devices) may suffer from delays, reliability and resource constraints. Since reliability and overhead have a direct impact on the delay, we focus on relevant performance aspects involving a more logically-centralized approach influenced by the SDN approach. The idea is to have a flexible network that adapts the communication protocols as well as multi-homing capabilities in diverse network conditions. Initial results investigating the previous research aspect, i.e., the extended UNIC Platform adapts on-the-fly important parameters of the IoT routing protocol (i.e., RPL) as well as utilizes the multi-homing capabilities, are reported in the final deliverable on MEC project[5].

[4]https://bit.ly/2KSO6HF
[5]https://www.monroe-project.eu/mec/

Our initial investigation in the MEC project regarding the multi-homing capabilities as well as the adoption of IoT protocols was the precursor, in cooperative work involving the developing, designing and implementing an advanced SDN Platform, namely MINOS [69]. MINOS is a multi-protocol SDN platform for IoT, that implements service awareness utilizing appropriate SDN abstractions and interfaces for logically centralized network control of diverse and resource-constrained IoT environments involving two network protocols that are deployable and configurable on demand.

2.9 The 5G-CDN Platform solution for heterogeneous distributed infrastructures

In the previous subsections, we elaborate on a novel elastic CDN paradigm and a relevant change point mechanism utilizing lightweight Unikernels based VMs. Although the first experimental results extracted from our relevant UNIC platform have been promising, we faced difficulties to conduct experimentation resembling real CDN deployments in order to tackle aspects such as scalability, performance efficiency, heterogeneity of physical / virtual resources, and flexibility. To bridge the gap between CDN-based services and tackle essential 5G features, we built a relevant experimentation environment, namely 5G-CDN platform on top of the Fed4FIRE+ facilities, which leverages the UNIC solution introduced in section 2.5.1. Our new novel platform: (i) covers the service and virtualization aspects and enables relevant automation (e.g., dynamic resource allocation and discovery); (ii) abstracts test-bed control approaches, multiple virtualization technologies and resource specifications (i.e., through applying a uniform representation of resources); and (iii) refines a high-level service definition to lower-level slice specifications while supporting modular CDN service orchestration.

2.10 Design and Implementation of 5G-CDN Platform

The architecture adopted by the platform, depicted in Fig. 13, consists of two planes, the *Management and Orchestration* and the *Multi-Domain Experiment Engine*. The former provides all the necessary experimenter GUI, experiment definition and specification refinement, slice creation, control and management features, including advanced

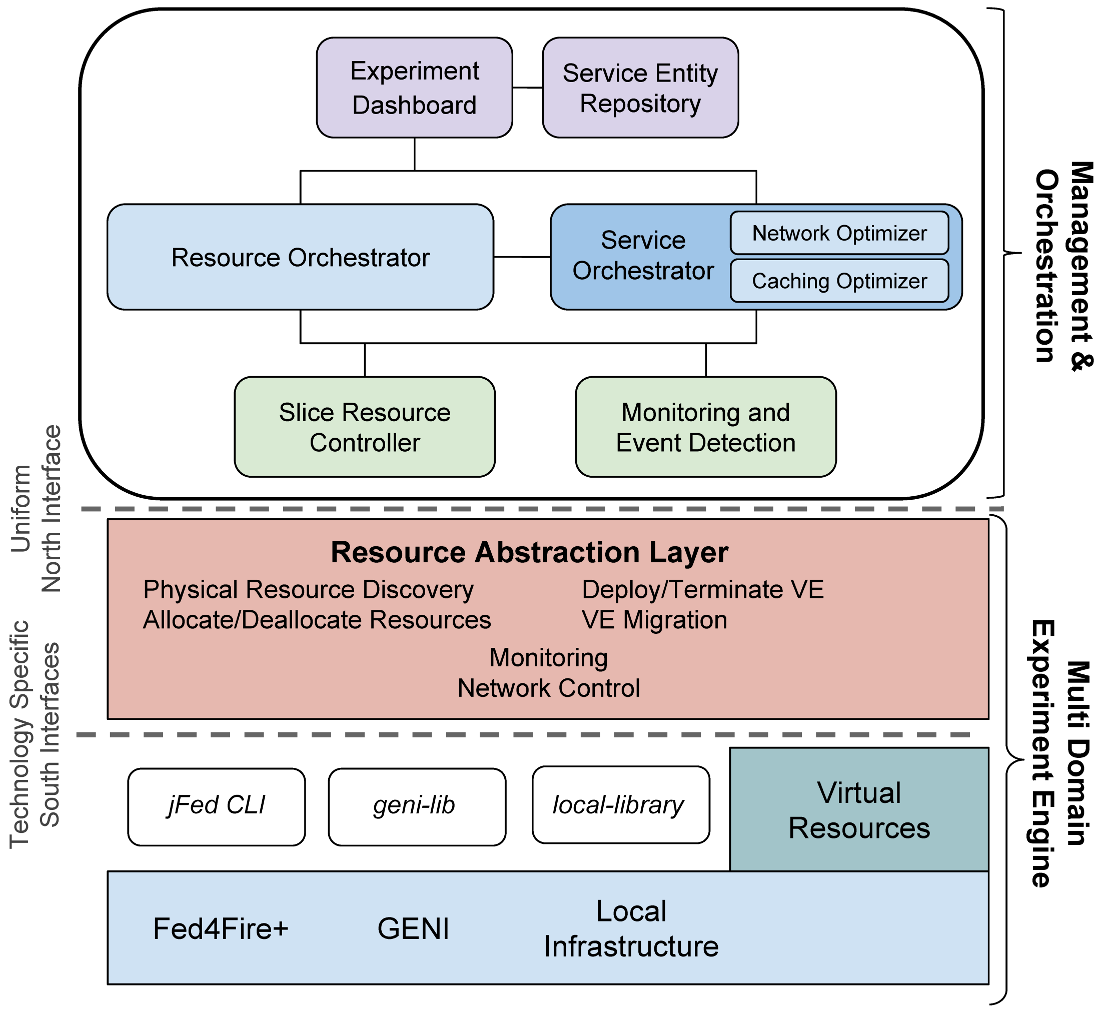

Experiment Dashboard
Service Entity Repository
Resource Orchestrator
Service Orchestrator
Network Optimizer
Caching Optimizer
Slice Resource Controller
Monitoring and Event Detection
Management & Orchestration
Uniform North Interface
Resource Abstraction Layer
Physical Resource Discovery
Allocate/Deallocate Resources
Deploy/Terminate VE
VE Migration
Monitoring
Network Control
Technology Specific South Interfaces
jFed CLI
geni-lib
local-library
Virtual Resources
Fed4Fire+
GENI
Local Infrastructure
Multi Domain Experiment Engine

Figure 13: The architecture of proposed 5G-CDN platform

monitoring and results aggregation functionalities. The Engine's role is to provide uniform access to the federated test-bed resources, alleviating the need for low level, technology specific code provisioning on the experimenters' side. These two planes are explained in detail in the following subsections.

2.10.1 Multi-Domain Experiment Engine

The Multi-Domain Experiment Engine consists of two layers, namely:

- The bottom layer of the *Multi-Domain Experiment Engine* accommodates the physical resources located either on remote federated test-beds (i.e., currently Fed4FIRE+ and GENI [70]) or on our local infrastructure through custom relevant libraries. Accessing local resources through a federated experiment is important, since many important pilot 5G deployments are not part of large-scale federated test-beds (e.g., 5TONIC [71]).

- The *Resource Abstraction Layer (RAL)* hides the resource and test-bed heterogeneity from the top architectural plane. *RAL* offers a technology agnostic uniform *North interface* to the upper *Management and Orchestration* plane, while translating incoming "north" operations to test-bed specific commands via its technology specific *South interfaces*, i.e., one for each diverse test-bed control approach. Moreover, *RAL* serves as a resource catalogue/provisioning layer for the experiments to be conducted, where resources are accessed by their corresponding test-bed control interface (e.g., jFed CLI or geni-lib). In practice, we maintain a local representation of the resources (in json format). This happens because both Fed4FIRE+ and GENI represent their resources through Resource Specifications (called RSPECs [70]), but not in a uniform manner between the test-beds, e.g., the resources may have incomplete details or present different attributes.

The Virtual Entities (VEs), i.e, VMs and containers, are treated similarly to physical ones and are also accessible via the *RAL*. We support services operating over multiple Virtual Infrastructure Managers (VIMs), since it is common for edge and core clouds to use different virtualization technologies (e.g., OpenStack for core and Docker/Kubernetes for edge clouds). Finally, the *RAL* provides abstractions for diverse monitoring technologies for both slice resource and content delivery aspects, as well as SDN network control operations.

In short, the Engine provides a "universal" (i.e., physical & virtual), uniform (i.e., technology agnostic) and flexible (i.e., extendable to include new classes of resources) *North interface* to the high-level management and orchestration components. Due to these features, this plane enables experiments on dynamically discovered, heterogeneous resources.

2.10.2 Management and Orchestration

The *Management and Orchestration* plane follows a modular architecture that consists of the components depicted in Fig. 13 and detailed below.

- The *Experiment Dashboard* is the platform's interface to the experimenter. It includes a GUI and along with the *Service Entity Repository* allow defining the service specifications and the details of each experiment (e.g., slice geographic constraints, KPIs, monitoring technology). In particular, the repository: (i) provides a set of visual components (for instance a Content Server VM), that can be added (drag-n-drop) to the *service graph* description, and (ii) assists the conversion of the *service graph* to a *slice graph* based on initial slice configuration details, that include but are not limited to the VIM requirements of the selected service entities, their resource requirements and network connectivity constraints. For example, a *service graph* represents the relations among service components, e.g., a load-balancer and a number of content servers are connected with "edges" that are annotated with requirements regarding service hosting components (e.g., VIM) or geographical and connectivity constraints. In the same example, the *slice graph* is the allocation of service components to slice parts, along with any network connectivity or physical/virtual resource requirements (e.g., number of physical machines, CPU / RAM demands), that will be populated during the slice creation phase. Graphs' representation is based on a custom YAML schema, inspired by TOSCA [50] and the ESpec [6]. The Dashboard also supports visualization of results and a bespoke visualization tool for large-scale CDN deployments

- The *Resource Orchestrator (RO)* component handles the population of the experiment definition (i.e., represented as a CDN service graph) with physical resources. This slice building process involves the selection of appropriate resources among those available, requesting the allocation of resources and the necessary test-bed stitching from the *Slice Resource Controller*. Operations involving slice resource elasticity, i.e., addition / removal of resources to existing slice parts (or test-bed-level) and new slice part (or test-bed) allocation along with the necessary stitching, also fall under the responsibility of the *RO*, that decides the new slice configuration.

- The *Service Orchestrator (SO)* offers service Virtual Entity (VE) deployment (i.e., Unikernel-based VM or a container) on the allocated slice resources, based on configurable placement algorithms, VE termination and operations involving service elasticity, e.g., VE migration or deployment, as a response to service request events. The service request events are detected via the corresponding component (*MED* - see below) in situations that require a graceful service elasticity operation. The two sub-components that handle this kind of events are the *Network Optimizer*, that deals with placement of VEs related to load balancing and traffic redirection in the slice, and the *Caching Optimizer*, that decides on the deployment of new VEs to respond to increased service requests. Finally, the *SO* handles the experiment scenario execution, i.e., starts/stops VEs emulating service clients' requests for video offered by a CDN service running on the slice. To perform these tasks the *SO* interacts directly with the *RAL*.

- The *Slice Resource Controller (SRC)* discovers available resources with particular specifications through the *RAL* and handles the requests for physical resource allocation in the supported test-beds. In general, the *RO* takes the slice-level

[6]http://jfed.ilabt.imec.be/espec

configuration decisions, while the *SRC* the slice-part-level (or test-bed-level) decisions. The *RO* and *SRC* are considered to be the main enablers for the E2E slice creation.

- The *Monitor and Event Detection (MED)* module accesses monitoring data for physical and virtual resources via the *RAL* and detects traffic and other network events that one or both of the orchestrators should respond to. For instance, we carry out experiments on content popularity detection employing change point analysis [72, 73] triggering the *Caching Optimizer* component of the *SO* to deploy additional content-serving VEs. This component is also responsible for the results aggregation presented at the *Experiment Dashboard.*

All the above components besides the *RO* have been implemented through the NodeRED tool and according to the microservices paradigm [74]. NodeRED is a browser-based flow editor wiring together independent software entities. Each entity is represented as a NodeRED node, i.e., a standalone Node.js component. For example, the addition of a new VE placement algorithm requires only a simple GUI task and minor changes in the placement decision mechanism through the embedded code editor.

2.11 Evaluation Results

In this section, we demonstrate the functionality of the proposed 5G-CDN platform through experiments on different aspects of the studied large-scale CDN deployment of Unikernel-based content provisioning over heterogeneous resources. In particular, we investigate the following aspects:

2.11.1 Performance and scalability of 5G-CDN platform

Here, we evaluate our experimentation facility realizing large-scale Unikernel-based content provisioning over heterogeneous (i.e., virtual and physical) Fed4FIRE+ resources.

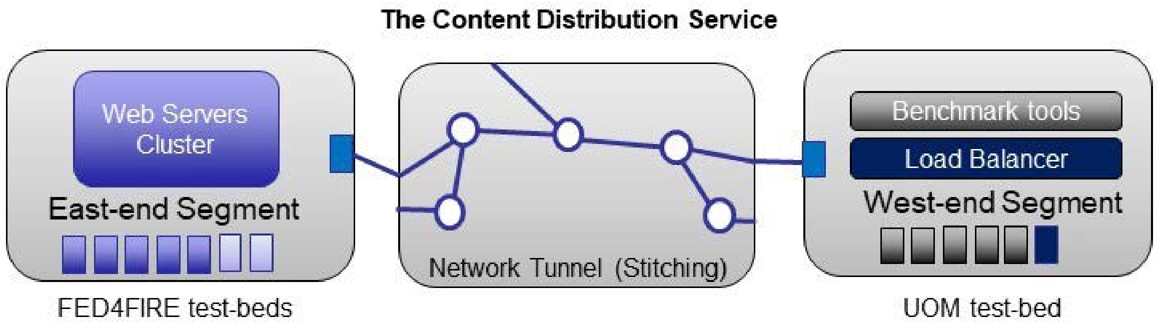


Figure 14: Content distribution service

2.11.1.1 Experimentation Setup

We validate the prototype implementation of our 5G-CDN architecture, acting both as our main experimentation facility built on top of Fed4FIRE+ and as a novel proposal for large-scale orchestration of CDN services. For our experiments, we consider a particular CDN service geographically spanning over Europe and USA. We configure the experiment through a YAML-based schema, that represents a high-level definition of a CDN experimental setup. This setup consists of the following services: (i) a cluster of Web servers; (ii) a load balancer (LB), which distributes (i.e., in a round robin fashion) the Web traffic of request content from clients; and (iii) benchmarking tools (BT) emulating the clients' behaviour. The same input of the experimenter specifies each service to be allocated in a particular geographic location, and thus the whole experiment spans in two different segment (test-beds) as depicted in Fig. 14. In particular, the East-end segment contains physical resources located in the Fed4FIRE+ test-beds, whereas the West-end segment consists of physical resources located at our own UOM test-bed. Furthermore, we deploy an additional physical machine at each side, acting as an edge router. In practice, we deploy GRE or VXLAN tunneling between the test-beds. To secure intra-connectivity, we configure each physical node's routing table to allow the communication with the edge router of the other side. For inter-domain connectivity, we assume a star topology, where each physical node is connected to the edge router (central node) and through the edge router to the remote test-bed segment (physical remote nodes).

To proceed with the CDN service deployment, we define the service and resource requirements as a YAML-based input that defines two CDN segments. More specifically, the west-end CDN segment at the UOM test-bed consists of six physical nodes hosting: (i) a service load balancer, which distributes (*i.e.,* in a round robin fashion) the Web traffic of a number of clients to the Web servers located at a remote CDN segment (*i.e.,* the east-end); and (ii) the benchmarking tools emulating the clients' behaviour. The east-end segment acting as a Web servers' cluster is physically located in the USA (*i.e.,* at the CloudLab Utah test-bed) and is accessed through the Fed4FIRE+ facility. The number of physical machines in this segment is expressed by the parameter *Nodes* of our experiment, which is in the range of $[5 \ldots 30]$. We use two classes of *test-beds nodes*, *i.e.*, the *pc3000* class with 3.0 GHz processor, 2 GB DDR2 RAM and 300 GB storage, and the *d430* class with two 2.4 GHz $8-$core processors, 64 GB DDR4 RAM and 2.2 TB storage (a detailed description of hardware specifications can be found here[7]). The first class represents a server deployed at an edge cloud and the latter at a core cloud. In this particular east-end segment, we assume that each physical node hosts one Web server, supports a particular virtualization technology (*i.e.,* ClickOS) and is assigned with specific service resource flavour (*i.e.,* CPU, RAM utilization and storage usage).

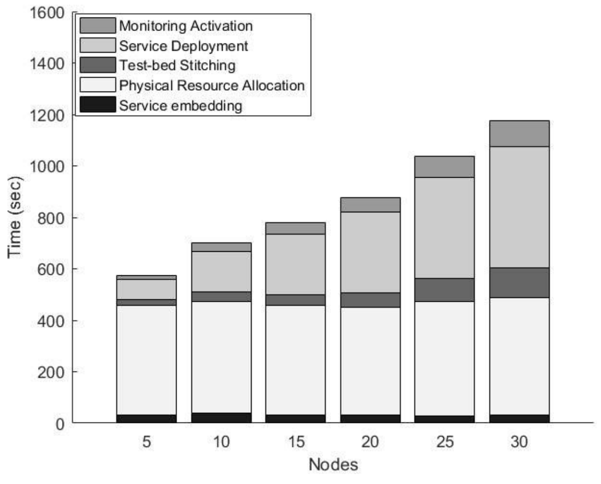
To monitor the behavior of the CDN deployment, we use the CollectD open-source monitoring tool, as well as the InfluxDB and the Grafana tool. For the allocated physical resources, we enable the following KPIs: (i) CPU and RAM usage; and (ii) incoming/outgoing traffic. The tool collects the KPI metrics every 20 sec (time interval).

2.11.1.2 Experimental Results

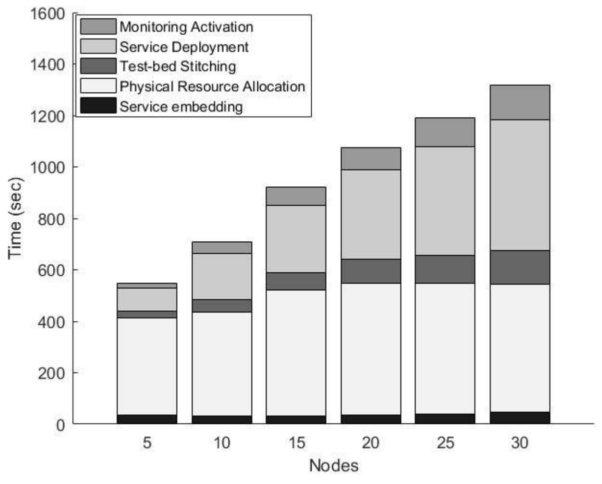
Our evaluation results show the different delays involved in the deployment of the CDN service, while considering both edge and core cloud nodes. Fig. 15 provides both a general view of the total delay incurred for the whole instantiation, as well as the time spent for each individual step, i.e., Service Embedding, Physical Resource Allocation, Test-bed Stitching, Service Deployment, and Monitoring Activation. Delay is expressed

[7]https://wiki.emulab.net/wiki/UtahHardware

as a function of the number of the physical Nodes deployed at the CloudLab Utah testbed. We report the results across five runs.



(a) Core cloud nodes

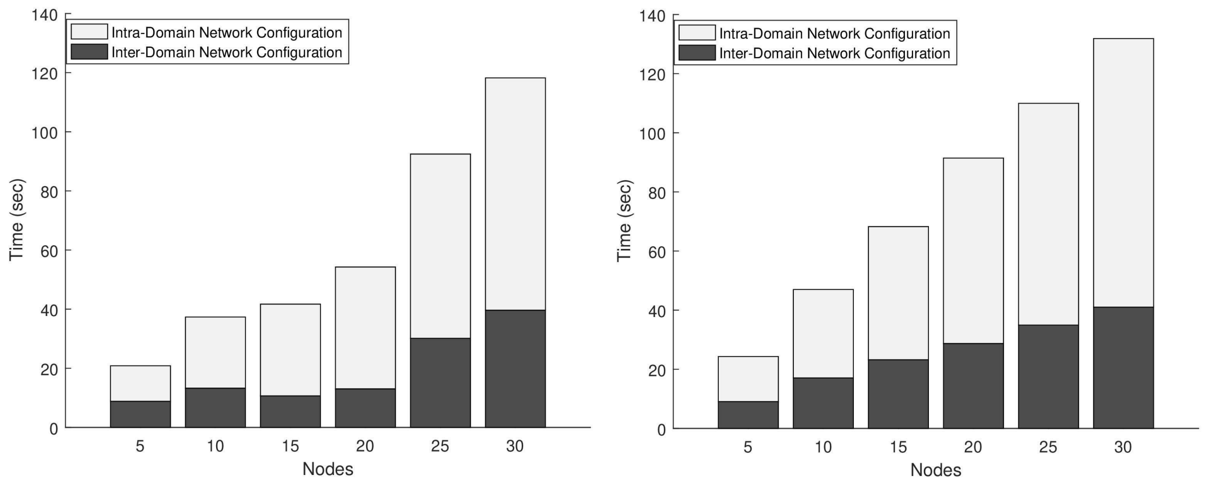


(b) Edge cloud nodes

Figure 15: Deployment time of CDN service.

Fig. 15(a) and 15(b) indicate that the less time-consuming steps are *Service Embedding*, *Test-bed Stitching* and *Monitoring Activation*. *Service Embedding*, in particular, is almost fixed at around 30 secs in the case of core cloud nodes, and at around $35-40$ secs in edge clouds. We discern that this step does not yield any scalability limitation, at least in the scale of our experimental setup, since it refines the experiment input to further details regarding the requested physical resources and the service deployment. The *Test-bed Stitching* step ranges from almost $20-120$ secs and $25-130$ secs, for core and edge cloud nodes, respectively. This step along with the monitoring increase linearly with the number of nodes. The *Monitoring Activation* starts at 16 sec and reaches almost 100 secs (Fig. 15(a)), whilst it ranges from 21 to 133 secs in Fig. 15(b). These results show slightly higher delays when edge cloud nodes are allocated for the east-end segment.

On the other hand, the *Physical Resource Allocation* and the *Service Deployment* steps are the most time-consuming. Resource allocation involves the servers' boot-up time, which entails the prolongation of the total deployment time. However, in case of the core cloud nodes, the resource allocation requires as much as 74% of deployment time when $Nodes = 5$, which significantly decreases at 30% when $Nodes = 30$. The



(a) Core cloud nodes (b) Edge cloud nodes

Figure 16: Network stitching time.

corresponding percentages range from $69-37\%$ in case of edge cloud nodes. This decrease is due to the fact that the time remains almost stable with the increase of the nodes. In contrast to this observation, the delay incurred for *Service Deployment* increases with the number of nodes. As a result, service deployment attributes $14-40\%$ and $16-38\%$ (in respect to the number of nodes) of the total experiment deployment time when core and edge cloud nodes are employed, respectively.

In Figs. 16(a) and 16(b), we further elaborate on the time required for *Test-bed Stitching*. As described in subsection 2.11.1.1, the 5G-CDN platform configures connectivity both inside the segments, and between the two geographically remote segments, as well. In these figures, the light dark and grey areas correspond to the intra- and inter-domain network configuration, respectively. According to these plots, intra-domain network setup incurs longer delays compared to the inter-domain network configuration, irrespective of the type of cloud nodes. This stems from the fact that in *Test-bed stitching*, physical node configuration takes place sequentially, whereas in the intra-domain case configurations are applied to a larger number of nodes (compared to inter-domain).

Finally, the exact delay values along with the standard deviations are reported for clarity reasons in Tables 1 and 2.

Table 1: Standard deviation of deployment time for core cloud nodes

Step	SD (σ) in sec	Nodes					
		5	10	15	20	25	30
Monitoring Activation	Time	16.07	31.83	43.25	57.48	82.44	102.34
	SD	1.51	2.87	0.06	0.11	5.92	1.02
Service Deployment	Time	80.56	158.19	235.86	312.97	391.61	469.95
	SD	0.24	0.93	1.19	1.65	1.33	0.76
Intra-domain network configuration	Time	12.01	24.06	31.04	41.22	62.32	78.59
	SD	1.54	3.18	0.09	0.04	6.06	0.97
Inter-domain network configuration	Time	8.84	13.28	10.67	13.05	30.15	39.64
	SD	2.75	4.54	0.14	0.43	8.34	0.24
Physical Resource Allocation	Time	426.91	434.49	428.50	419.09	442.61	453.93
	SD	5.04	10.88	0.52	28.84	4.44	17.18
Service Embedding	Time	31.31	37.79	30.02	32.83	29.17	32.56
	SD	3.06	10.08	2.28	3.90	1.51	1.96

Table 2: Standard deviation of deployment time for edge cloud nodes

Step	SD (σ) in sec	Nodes					
		5	10	15	20	25	30
Monitoring Activation	Time	21.57	44.91	69.43	86.74	111.19	133.74
	SD	1.9	2.43	2.22	2.37	1.98	5.84
Service Deployment	Time	89.41	178.49	262.38	346.66	420.19	508.21
	SD	3.18	4.92	10.77	11.12	3.01	7.81
Intra-domain network configuration	Time	15.24	29.91	45.04	62.72	74.99	90.85
	SD	2.63	0.66	0.35	2.36	1.17	2.44
Inter-domain network configuration	Time	9.07	17.07	23.23	28.72	34.96	41.03
	SD	2.81	0.59	0.72	0.85	0.22	0.55
Physical Resource Allocation	Time	379.73	406.53	488.59	512.8	508.44	497.95
	SD	26.83	35.22	29.65	10.99	1.36	30.13
Service Embedding	Time	34.36	30.33	32.4	36.19	39.54	45.44
	SD	5.05	4.54	4.66	10.36	8.44	19.66

2.11.2 Dynamic resource discovery for scalable CDNs

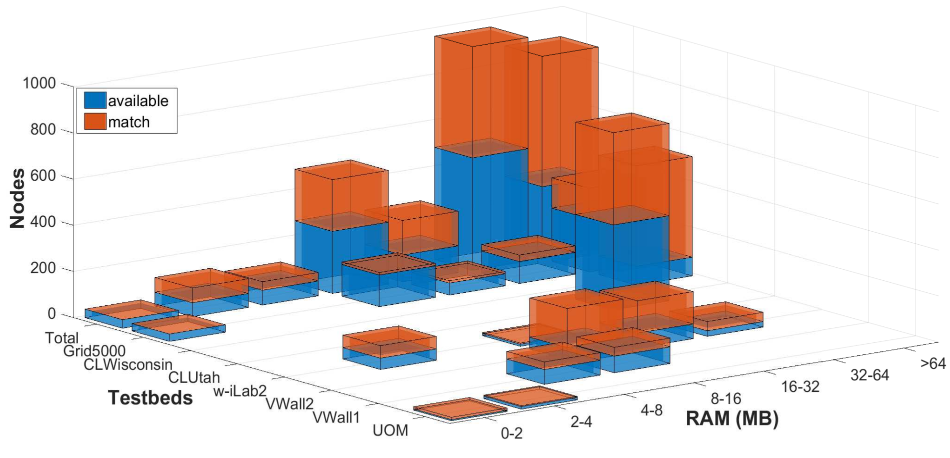
In this experiment, we validate the capabilities of our dynamic resource discovery mechanism to discover and allocate the appropriate resources for large-scale CDN deployments, over six different test-beds participating in the Fed4FIRE+ federation [23]: (i) w-iLab2, Virtual Wall 1 (VWall1), Virtual Wall 2 (VWall2) and Grid5000 test-beds, which are located in Europe; (ii) CloudLab test-beds in Utah (ClabUtah) and Wisconsin (CLabWisconsin), i.e., located in the USA and (iii) our local test-bed resources. This is particularly essential, since for example, when clients' interest for media content suddenly increases, new VMs should be deployed to deliver the content; however, the decision for their placement should be tailored to the available resources.

Practically, the *Slice Resource Controller* component of the 5G-CDN platform dynamically discovers the appropriate resources and exposes their specifications in respect to the search criteria related to the memory, number of cores, disk storage and NIC bandwidth information. To retrieve in real-time the status of resources, e.g., the available RAM, the *Resource Abstraction Layer (RAL)* is equipped with a *Python Translator*, which is responsible to directly communicate with the corresponding test-bed control interface (e.g., jFed CLI) and translate the response message into a uniform format. In practice, we maintain a local representation of the resources in JSON format. This treatment of the resources' features is critical because Fed4FIRE+ represents their resources through Resource Specifications (called RSPECs), but not in a uniform manner throughout the test-beds, e.g., the resources may have incomplete details or present different attributes.

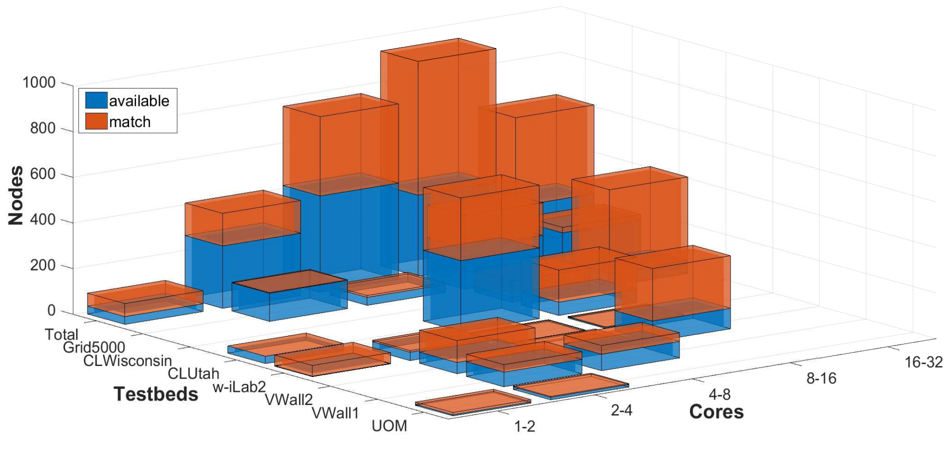
We organize our experimental evaluation in three series of trials (i.e., the first two with quantitative and the third with qualitative results) to evaluate the feasibility of conducting scalable CDNs experiments on the aforementioned infrastructure. We gradually increase the specifications of the resources demanding in each series of trials.

2.11.2.1 Experimental Results

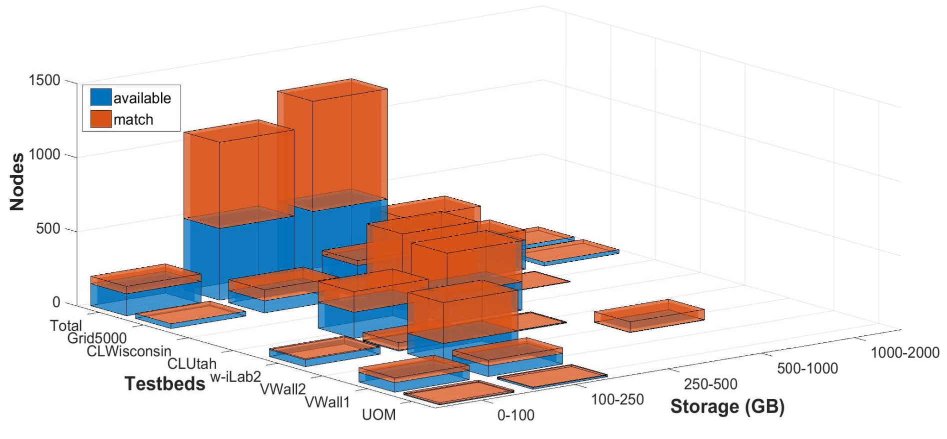
In the first set of results, we illustrate the span of the available resources that can be utilized once the resource discovery process has been completed. Fig. 17 presents the outcome of the *SRC* discovery process, for physical resources, satisfying specific memory, CPU, disk storage and NIC bandwidth criteria, on the different aforementioned test-beds. Hardware that simply matches these criteria and resources that are actually available during the discovery process is depicted in different color. The data plotted in Fig. 17(a)-17(d) provide information necessary for *SO* VM placement decisions. Additionally, such information also assists in experiment setup, indicating test-beds that can act as core clouds, since they contain high-end resources, and



(a) Memory



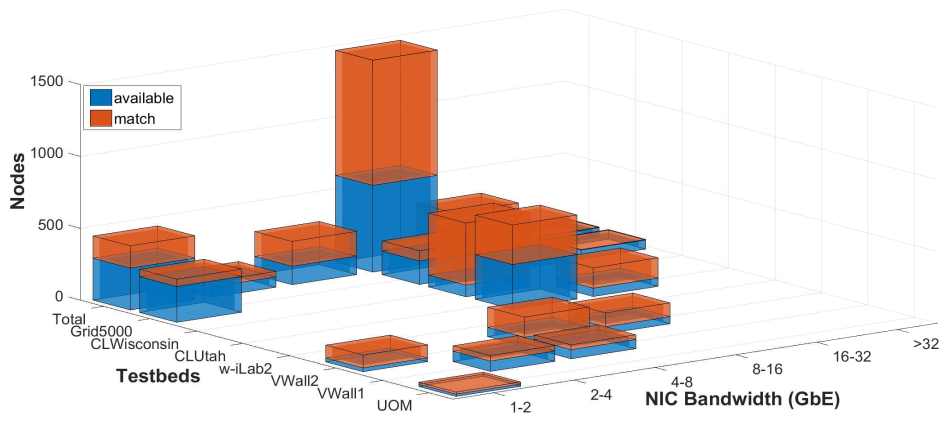
(b) Number of cores



(c) Disk storage

low-end resource test-beds that could operate as edge-clouds.

In the second set of results, we further validate the overall functionality of our dynamic resource discovery mechanism. We utilize coarse-grained queries, where we



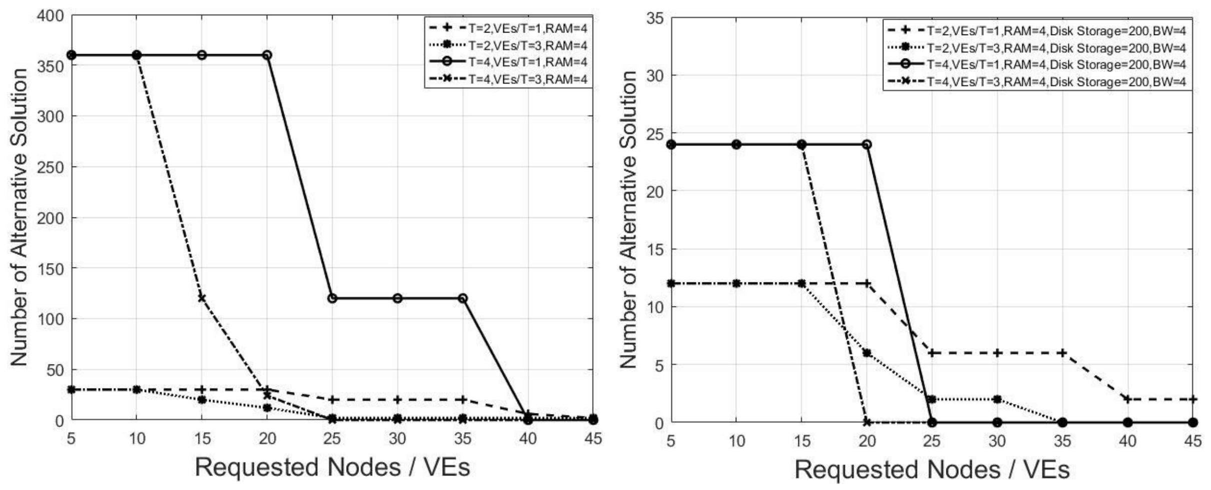
(d) NIC specifications

Figure 17: Resource discovery results in different testbeds

adjust the resources' request by increasing the number of physical nodes within a VE, the latter annotated with a single resource specification, i.e., the memory size (RAM). In addition, we carry out fine-grained queries for alternative solutions, which are differentiated not only according to the number of the requested test-beds *(T=[2,4])* and the VEs number per test-bed *(VEs/T=[1,3])*, but also in respect to the ability to define multiple resource specification per VEs (i.e., RAM, disk storage and bandwidth capabilities).

Coarse-grained queries provide the results of Fig. 18(a), where the number of alternative solutions decreases as the requested nodes per VEs increases. Such behaviour is expected, as the number of providers (i.e., test beds), that can accommodate the VEs decreases. Note that the number of VEs plays a significant role in the solutions, since it allows for more combinations of providers (i.e., alternatives) to be generated. Similar results deduce from fine-grained queries as depicted in Fig. 18(b). It is notable that resource requests with a high number of VEs and an increased set of resource requirements lead to a sharp decrease in the number of alternative solutions. This is attributed to the fact that very few test-beds can accommodate resource-demanding VEs.

It is important to emphasize again that, the curves in both graphs of Fig. 18 exhibit similar trend in the sense that the number of alternative solutions decreases as the



(a) One resource requirement

(b) Three resource requirement

Figure 18: Number of alternative solutions with different resource requirement per VEs.

demand for requested nodes increases. Even with the increase in the number of the test-beds to be involved in the experiment setup, which theoretically provides more alternative combinations, the number of possible solutions drops significantly from 20 requested nodes per VEs onwards. Finally, as also expected, strict requirements for resources (three specifications instead of one) further decrease the number of alternative solutions.

A final experiment in relevance to the dynamic resource discovery aims to demonstrate the allocation of a CDNs experiment setup across multiple test-beds, where the test-beds act either as core or edge cloud providers. We generate a resource request that demands any four available different geographically test-beds, each hosting two VEs. Each VEs has diverse resource demands, reflecting their role in the whole CDN setup, i.e., higher demands for the core cloud resources and lower demands for the edge cloud resources. Tables 3 and 4 summarize the different resource requirements according to the test-bed role as a core or edge cloud provider. Obviously, the resource requirements for an edge cloud test-bed are less demanding compared to that of a core cloud.

The outcome of this experiment provides the nine alternative solutions depicted in Table 5. Since the request did not define any geographic constraints regarding the VEs,

Table 3: Resource requirements for a test-bed to host a core server

Core Servers have higher resource demands

```
 Test-bed(location:'undefined',
['VE_1'([
  'available-nodes'>=2,'memory-gb'>64,
  'min-storage-gb'> 250,'nics-bw'> 2]),
 'VE_2'([
  'available-nodes'>=1,'memory-gb'>=128,
  'min-storage-gb'>=500,'nics-bw'>=5])
])
```

Table 4: Resource requirements for a test-bed to host an edge server

Edge Servers have low resource demands

```
 Test-bed(location:'undefined',
['VE_1'([
  'available-nodes'>=1,'memory-gb'>=8,
  'min-storage-gb'>= 80,'nics-bw'>= 1]),
 'VE_2'([
  'available-nodes'>=1,'memory-gb'>=8,
  'min-storage-gb'>=80,'nics-bw'>=1])
])
```

Table 5: Alternative solutions

Test-bed 1	Test-bed 2	Test-bed 3	Test-bed 4
Grid5000	CLabUtah	CLabWisconsin	VWall2
Grid5000	CLabUtah	VWall1	VWall2
Grid5000	CLabUtah	w-iLab2	VWall2
Grid5000	CLabWisconsin	CLabUtah	VWall2
Grid5000	CLabWisconsin	VWall1	VWall2
Grid5000	CLabWisconsin	w-iLab2	VWall2
Grid5000	VWall1	CLabUtah	VWall2
Grid5000	VWall1	CLabWisconsin	VWall2
Grid5000	VWall1	w-iLab2	VWall2

the resource discovery mechanism generated all possible combinations, distributing VEs between Europe (Grid5000, w-iLab2, VWall1 VWall2) and USA (CLabUtah CLab-Wisconsin). This demonstrates that such a loosely coupled resource discovery model can manage a diverse set of geographically distributed test-beds (i.e., infrastructure providers).

2.11.3 Heterogeneity on E2E slicing

In this class of results, we demonstrate the functionality of the 5G-CDN platform in the 5G multi-domain E2E slicing context, while focusing on the exploitation of the Unikernel technology as a means of implementing lightweight VMs, which are called Micro-Content Proxies (MCP), and highlight flexibility, scalability, and content provision under virtualization and hardware heterogeneity.

2.11.3.1 Experimentation Setup

Three different test-beds are utilized through the bottom layer of the *Multi Domain Experiment Engine*, namely: (i) the *Virtual Wall 2* (VWall2) test-bed which is part of the Fed4FIRE+ federation and located in Europe; (ii) the *Cloudlab Utah* (CLUtah) test-bed, in Utah, USA; and, (iii) our local *UOM* test-bed in Greece, Europe. We build multi-domain E2E slices on top of these pieces of hardware considering two clusters of nodes, as illustrated in Fig. 19. The first east-end cluster contains six physical nodes

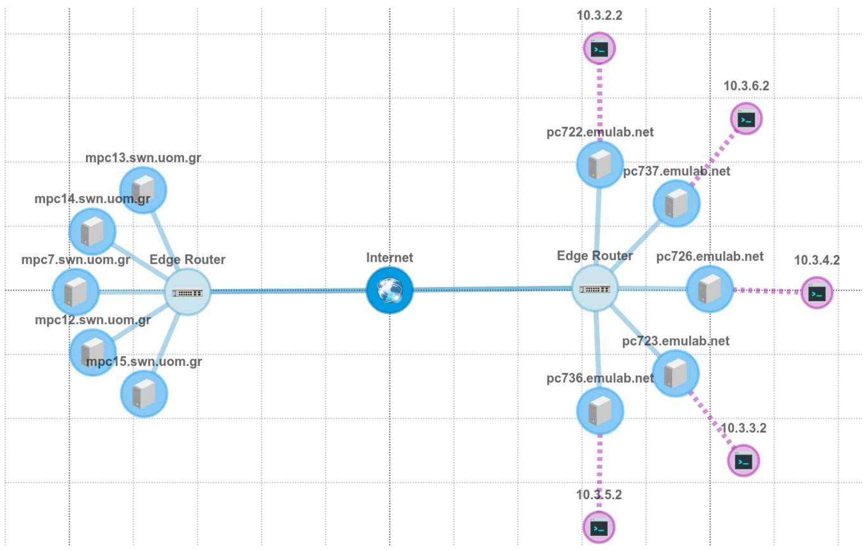


Figure 19: Abstract view of the E2E slicing in our experiment

– always part of the *UOM* test-bed – which five out of the six nodes emulate the clients' behaviour initiating media service requests. The second west-end cluster consists of six physical servers, five to host the MCPs. In both clusters, the extra physical node (i.e., 6th) serves as an edge router. By allocating the west-end cluster in different test-beds, we obtain experimental results for geographically distributed slices, e.g., *UOM-UOM*, *UOM-VWall2* and *UOM-CLUtah* slices. To make the above infrastructure available for our experiments, the *SRC* performs dynamic resource discovery and exposes specifications, such as the memory, number of cores, disk storage and NIC information of hardware found.

Apart from hardware, heterogeneity in our experimental setup is also reflected in the *RAL*, where different lightweight virtualization technologies are offered, namely ClickOS [34], Rump Kernel [35] and MirageOS [33]. Fig. 19 depicts an instant of such VMs placement in the MCPs' side (west-end cluster) orchestrated by the *SO Caching Optimizer* mechanism.

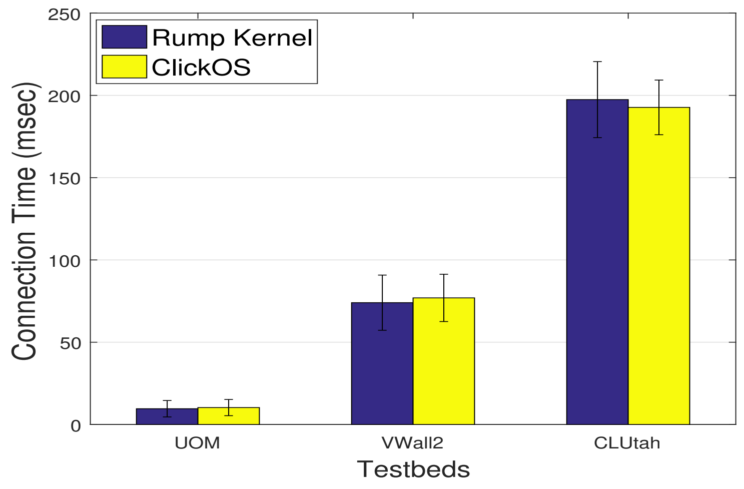
2.11.3.2 Experimental Results

We organize our experiment in two scenarios to demonstrate different aspects of heterogeneity: (i) the diversity in hardware types (i.e., nodes) as well as on test-beds' geographic location; and (ii) the diversity in lightweight virtualization technology and the content size being delivered as a response to the Web clients' requests.

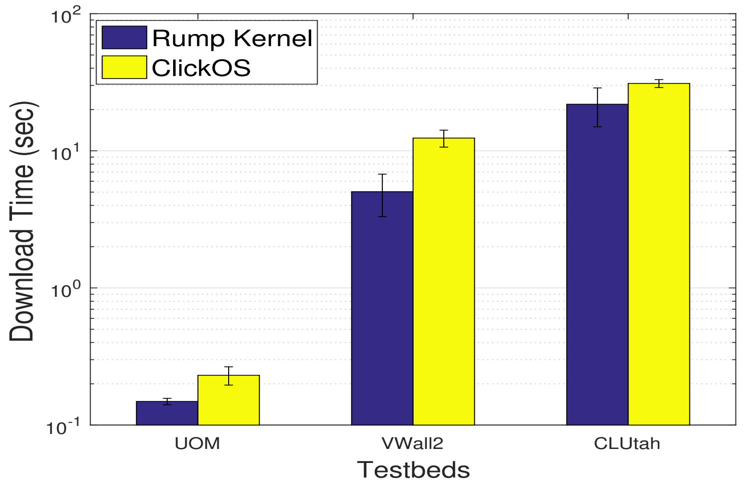
In the first set of results (i.e., Fig. 20 and Fig. 21), we evaluate the performance of MCPs hosting and launching data of 8 MB size, in terms of clients' *connection time* (i.e., the time elapsed between sending a media service request and receiving the first response byte), *download time* (i.e., the time required for downloading to be completed), *network throughput* (i.e., the ratio of the amount of data downloaded to the download time) and *CPU utilization*. We conducted the same experiment three times assigning the west-end cluster to different, gradually geographic isolated test-bed, i.e., UOM, VWall2 and CLUtah, while each running uses either ClickOS or Rump Kernel as virtualization technology.

Fig. 20(a) and 20(b), we validate the communication performance, as expected the geographically remote test-beds deteriorates users' experience. Indicatively, clients' connection time is roughly 10 *msec* for requests being served locally, 75 *msec* in the Greece-to-Europe slice and 200 *msec* in the Greece-to-USA slice. The "geographical" distance is also reflected in the data download time which ranges from 0.1 *sec* to 32 *sec*. Obviously, for a fixed data size of 8 MB, lower download time entails higher network throughput, which is illustrated in Fig. 20(c). Regarding the Unikernels' technology, we observe deviation in the performance due to differential implementation approach for each unikernel. We notice that Rump Kernel outperforms ClickOS in download time and network throughput, while they exhibit similar performance in respect to the clients' connection time.

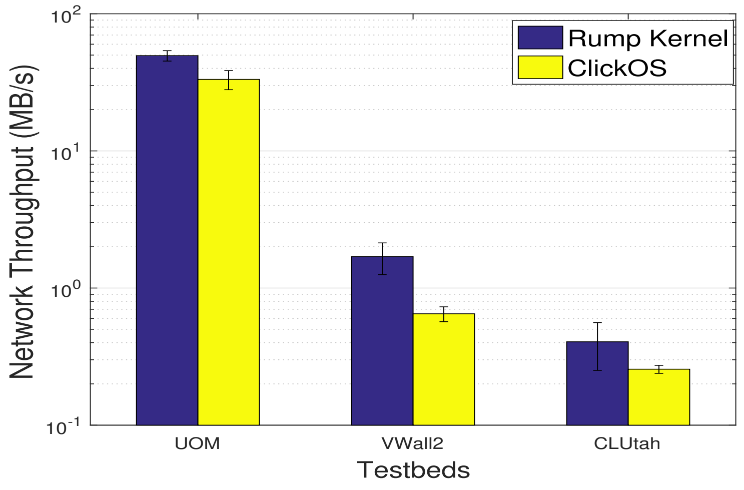
Clients' connection time is influenced among others by the CPU utilization in physical servers which is evaluated with respect to the virtualization technology used in the MPCs. Results of Fig. 21, in particular, show that Rump Kernel technology utilizes more CPU resources than ClickOS, especially in the case that the end-device has a low resource pool. In detail, the CPU usage for low (i.e., Mpc), medium (i.e., pcgen03-p1) and high (i.e., d430) resource pool for ClickOS is 0.89%, 0.03% and 0.01%, respectively, while in case of Rump Kernel is $1,11\%$, 0.03% and $0,02\%$, correspondingly. Such results provide us with insights that could nourish our estimations regarding



(a) Clients' connection time



(b) Clients' download time



(c) Network throughput

Figure 20: Communication performance with different Unikernel technologies in E2E slices

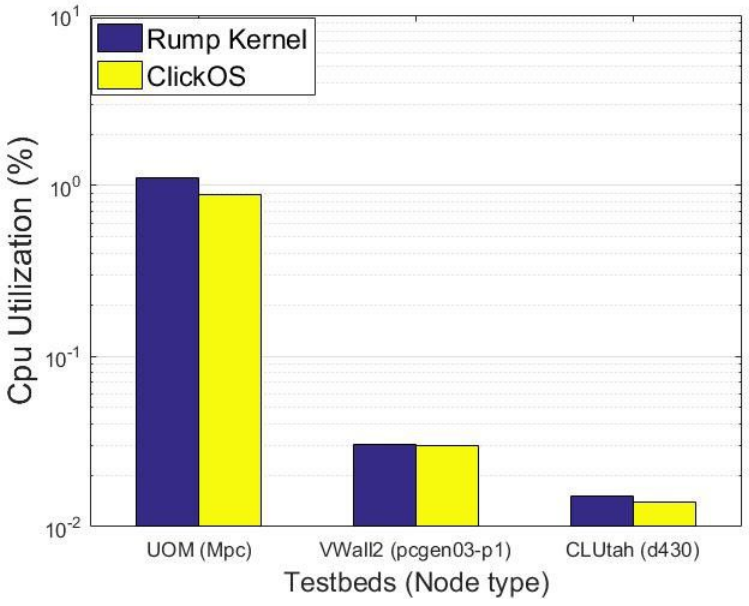


Figure 21: CPU utilization with different Unikernel technologies and node types.

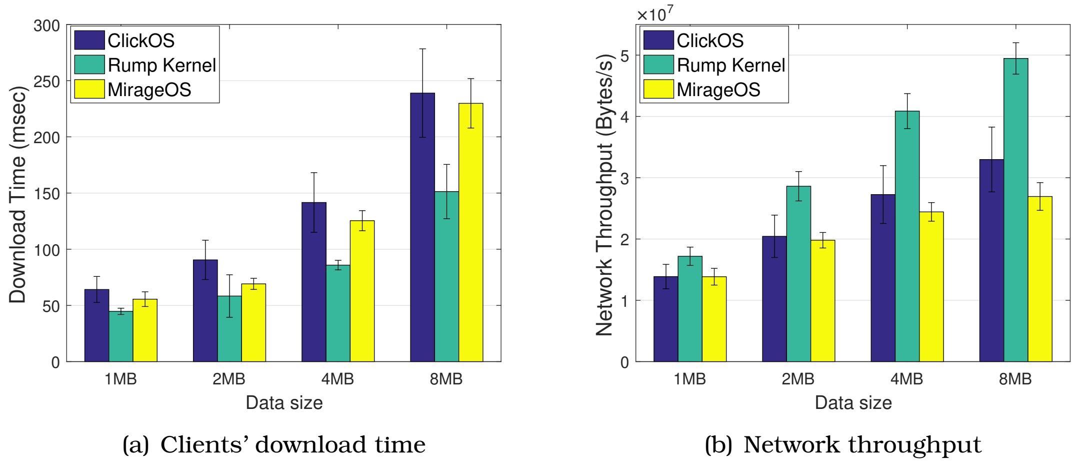


Figure 22: The different unikernel technologies are evaluated in line with the content size (MB)

the CPU usage percentage being consumed by each client.

In the second set of results, we further elaborate on VMs technology in Fig. 22, where we test the performance of ClickOS, Rump Kernel and MirageOS technologies assuming that both MCPs and clients reside on the *UOM* test-bed and the MCPs serve data of $1, 2, 4$ and 8 MB. Our results show that the download time (Fig. 22(a)) and network throughput (Fig. 22(b)) are straightly associated with the data size, independently

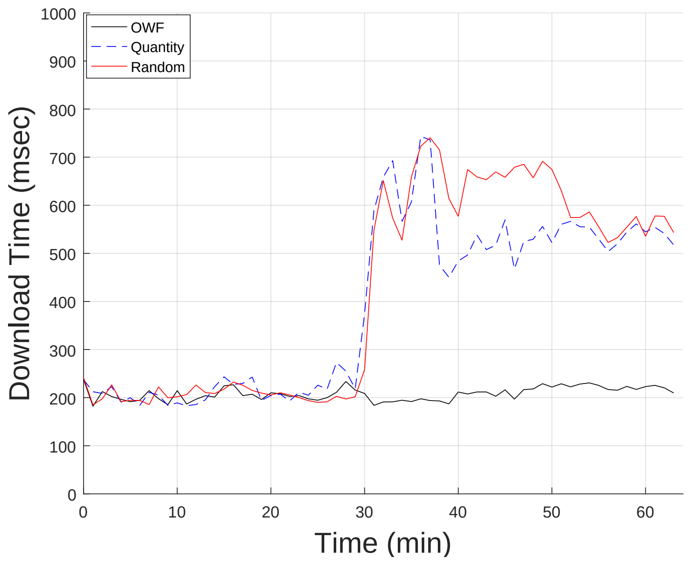


Figure 23: The placement algorithms' performance

of the Unikernel technology used, while Rump Kernel outperforms other technologies in respect to both metrics, especially for high data volumes. This outcome could lead to a VM placement strategy selecting the Rump Kernel technology, when data size is augmented. However, such behavior requires further investigation as we assess more thoroughly in chapter 3.

2.11.4 Modular service orchestration

Our last outcome demonstrates in real-time an example of a modular service orchestration mechanism available in our platform. We consider a similar experimentation setup as described 2.11.3.1 and deploy RumpKernel technology. More precisely, it is a preliminary result depicting the performance of three placement algorithms the *OWF* (i.e., as described in subsection 2.5.1.2), *quantity* and *random*, in respect to the download time experienced by the end-users.

Once the *RO* informs the *SO* for a pool of available resources, a placement algorithm (part of the *Caching Optimizer*) should decide to deploy additional MCPs on the allocated resources. Obviously, the *random* and *quantity* choice cannot perform better compared to one that takes into account real-time CPU, RAM and network utilization measurements (i.e., RT and TT) already running on the candidate resources (Fig. 23). However, it is required to extend our experiment analysis to define and consider further realistic parameters in order to formulate a more sophisticated optimization algorithm for the resource allocation problem, including investigating it under stress conditions and multiple infrastructure providers, as we present in chapter 4.

2.12 Conclusions

In this chapter, we introduce two relevant edge cloud orchestration facilities, namely the UNIC and 5G-CDN, along with relevant experimental results highlighting the benefits of utilizing virtualization technologies, motivated by the unique requirements of the 5G networks evolution.

The UNIC is a novel experimentation platform, that orchestrates both cloud and network aspects and supports heterogeneous lightweight virtualization in the network edge. Our use case scenarios of elastic content distribution and IoT measurements' collection as well as the experimental results demonstrate the above two aspects and its full system operation. Furthermore, the 5G-CDN platform, a novel experimentation facility for large-scale, multi-domain E2E slicing, focuses on the next-generation Content Delivery Networks (CDNs), a paradigm for hosting and launching M&E services. It is an evolution of UNIC solution and built on top of the Fed4FIRE+, to enable aspects such as: (i) large-scale experimentation with unikernel-based technologies, utilizing multiple geographically distant facilities focus on the feasibility of multi-domain slice deployment; (ii) dynamic resource discovery and allocation for flexible CDN experimentation, while validating the proposed facility and highlight the magnitude of the solution space for the complete slice offerings; (iii) end-to-end network slicing over multiple infrastructure providers utilizing heterogeneous hardware and virtualization resources; and (iv) novel media service orchestration mechanisms for content-popularity detection, resource allocation and load balancing.

3 A Comparative Evaluation of Edge Cloud Virtualization Technologies

3.1 Introduction

5G networks and beyond (5GB) [75] are being characterized by significant network performance and capacity advantages, especially at the radio level. They devised a promising agenda targeting new applications that enable a radical transformation of vertical sectors, including manufacture, media & entertainment, health, energy and automotive industry. Such services should adhere to stringent requirements, e.g., ultra-low delay or high throughput, scalable operation, and increased adaptability to dynamic contexts, in terms of service needs or resource availability. In other words, there is a need for systemic adaptations of 5GB ecosystems towards these goals. For example, virtual resources contribute significantly to end-to-end (E2E) service performance. Indicatively, for a 50ms E2E delay documented in paper [76] on 5G networks, radio and transport aspects caused the 17% of delay, while cloud dimension the 83%.

Along these lines, edge cloud computing is an important enabling technology for 5GB ecosystems, bringing computation close to end-users to improve service performance, reliability and data privacy of users. Thus, how to properly exploit the available computing substrate is a major concern for network operators with respect to the overall network and service management. Several aspects can be considered to take informed decisions for the usage of this infrastructure. Clearly, the performance improvement that can be obtained from using distinct compute facilities, the availability of resources as the services become deployed, the need for elasticity to accommodate a variety of application requirements (as anticipated in 5G), as well as the particular virtualization solutions that can be put in place to account for all of this.

For instance, large-scale service deployments usually serve a vast amount of users spread throughout the globe, which require the involvement of edge cloud resources in many different places. However, it is challenging to deploy resources near every user, consequently, there is a need for optimized cloud facilities not only towards particular performance requirements, but also mitigating a potential limited resource availability. Such infrastructures should be able to mobilize any available deployment, even with alternative server configurations, as well as network or virtualization technologies.

Given the fact that user demands or application requirements may be dynamic, edge clouds should also support a quick deployment or removal of virtual resources, implementing horizontal and vertical elasticity processes, i.e., adapting the service deployment and cloud resources to these requirements [16], respectively. For example, legacy cloud deployments may be using inefficient virtualization technologies, including traditional virtual machines (VMs), that face slow times for deployment, downloading or scaling up of virtual resources.

There is an on-going effort towards adopting lightweight virtualization approaches for edge clouds, such as containers [13] and unikernels [14]. For example, the European 5G-PPP initiative investigates alternative container and unikernel approaches to be used for edge computing [12]. In our experience, different container builds or unikernel flavors exhibit diverse performance capabilities. For example, containers can achieve a robust operation, while unikernels have rapid manipulation capabilities, e.g., they can boot up just in ms, even with a TCP SYN or DNS lookup request packet [15].

We argue that there is no single best virtualization solution for edge clouds, but the choice depends on the particular requirements, suggesting that the selection of the most appropriate approach should become an important overall management and operation task. Furthermore, a number of papers (e.g., [77], [78]) perform comparative evaluations of different virtualization technologies, but, in our understanding, none considers all basic tasks of edge cloud deployments, including the elasticity of service nodes and physical resources, as well as a service operation.

3.2 Contributions and Chapter Organization

3.2.1 Contributions

In this chapter, we conduct a systematic experimental evaluation of alternative container and unikernel builds of exemplary web services towards improving performance and adaptability of edge clouds. We bring a number of novelties, including on:

- introducing *a novel edge cloud experimentation environment*, supporting load prediction and balancing, horizontal and vertical elasticity, as well as common abstractions and APIs over heterogeneous virtual resources;
- providing an *extensive experimentation analysis* of different lightweight virtualization options for edge clouds, considering all basic edge cloud processes (i.e., resource allocation, removal, service operation, horizontal and vertical elasticity actions).
- presenting *basic design guidelines of edge cloud orchestration systems*, backed by our results and highlighted through a relevant conceptual facility that supports container and unikernel-based virtualization technologies as well as alternative service node implementations, while exploiting their diverse performance characteristics.

Our experimentation exercise was challenging, since it required the implementation of a complete edge cloud orchestration solution that supports heterogeneous virtualization choices, including the production-ready containers and alternative experimental implementations of unikernel flavors, often with bugs and instabilities. We require coding in many environments, i.e., C, Python, NodeJS, OCaml, the implementation of a bespoke DNS server, as well as a number of non-trivial OS, network, hypervisor and test-bed configurations.

3.2.2 Chapter Organization

The remainder of the Chapter is organized as follows. Section 3.3 provides an overview of the related investigations. Section 3.4 details our experimentation environment.

Section 3.5 elaborates on our methodological approach and its relevant assumptions. Section 3.6 provides our experimentation analysis and produced insights, focusing on the deployment, removal, operation and elasticity of edge cloud resources and services. Section 3.7 provides basic design guidelines for a novel edge cloud orchestration system that benefits from our findings. Finally, Section 3.8 concludes the chapter.

3.3 Related Works

Next-generation services call for network and cloud paradigms that enable ultra-low latency or high-throughput communication and bring elasticity in the service operation. For example, a number of approaches particularly focus on speeding-up packet processing, including the novel in-kernel proposals of extended Berkeley Packet Filter (eBPF) and Xpress Data Path (XDP) [79]. Furthermore, the edge cloud infrastructure StarlingX [8] targets at achieving ultra-low latency of network services through operating on top of real-time linux, employing Time-Sensitive Networking [80] capabilities.

Regarding the cloud viewpoint, 5G networks are gradually employing virtualization [81] and edge cloud technologies [82], as well as the microservice paradigm [83]. However, edge clouds may be associated with limited resource availability or dynamic service demands or network conditions, highlighting the need for flexible, lightweight virtualization technologies [12] at the edge, being efficiently orchestrated.

The main candidates for lightweight virtualization in edge clouds are containers and unikernels. Containers (e.g., Docker [31] or LXC [32]) are standardized units implementing application packaging with all of its dependencies, providing robust performance and adaptability to dynamic application requirements. Unikernels [14] (e.g., MirageOS [33], ClickOS [34], RumpKernel [35], or OSv [36]) are single-purpose appliances specialized at compile-time into standalone kernels, characterized by very low resource usage and rapid deployment capabilities, even at the range of ms [15].

Several studies conduct performance comparisons of alternative lightweight virtualization options, being suitable for edge cloud deployments. An overview of such works is shown in Table 6, highlighting the aspects being evaluated (i.e., on service

[8]http://www.starlingx.io/

Table 6: Related works comparing alternative virtualization technologies that are suitable for edge cloud environments

	Edge Cloud Aspects						Service Type	Number of considered container technologies	Number of considered unikernel technologies	Alternative flavors of the same application
	Service Operation		Elasticity or Fault-tolerance							
	Server Resource-efficiency	Service Performance	Resource Allocation Time	Resource Removal Time	Server Resource-efficiency Impact	Service performance Impact				
[84]	✓						Web-service & Database	2	2	
[77]	✓	✓					Web-service & Database	1	1	
[85]	✓	✓					Firewall	1	1	
[86]	✓	✓					DNS & Web-service		2	
[87]	✓	✓					HPC	1	1	3
[78]	✓	✓					Web-service & Database	1	4	2
[88]			✓				Virtual Entity Instantiation	1	1	
[89]			✓				Network Memory Server	2	1	
[90]	✓	✓	✓				Firewall	1	1	
[91]	✓	✓	✓				Web-service	2	1	
Ours	✓	✓	✓	✓	✓	✓	Web-service	1	3	2

operation and elasticity / fault-tolerance behavior), the considered services, as well as the numbers and types of contrasted virtualization options.

As enlisted in the Table 6, a number of considered works (i.e., [77] -[87]) assess the performance of alternative lightweight virtualization technologies and focus on investigating service operation aspects only. For example, the comparative analysis [84] focuses mainly on server resource-efficiency aspects (i.e., memory footprint and network latency) and contrasts alternative unikernel flavors with containers hosting an Nginx web-server or a Redis database. Most unikernel flavors perform at least equally or better than containers, especially in cases that require the transfer of unikernel or container images.

Other proposals consider both server resource-efficiency and service performance. Paper [77] compares KVM VMs, RumpKernel unikernels and docker containers hosting an Apache web-server or a Redis database with different numbers and sizes of requests. According to their results, containers achieve the best performance, in terms of communication delay and server resource utilization. Proposal [85] evaluates the performance of a unikernel-based firewall service against corresponding container and Linux-based solutions, concluding that the first option achieves a higher number of TCP requests served per second and a lower network latency. Similarly, paper [86] compares the network performance (i.e., requests served per second and latency) of unikernels against linux-based solutions offering both DNS and web-based services.

Furthermore, paper [87] evaluates the performance of unikernels versus containers for the same REST service, implemented in Java, Go, and Python. This study measures memory consumption and execution / response times and concludes that unikernels perform at least equally or outmatch the corresponding containers, however, the former consumes significantly more memory compared to the latter. Along the same lines, the authors of [78] compare the HTTP and database access performance of OSv, RumpKernel, MirageOS, and IncludeOS unikernels as well as docker containers, reporting a higher request rate in the case of containers, but a lower latency when employing unikernels.

A number of proposals consider aspects of elasticity or fault-tolerance processes. In [88], the authors compare KVM-based VMs, Docker containers and OSv-based unikernels in an OpenStack cloud platform, documenting that OSv outperforms the other virtualization technologies in terms of service provisioning time. A Vehicle Ad-Hoc Network realizing a service migration scenario and utilizing KVM-based VMs, both LXD and docker containers, as well as OSv-based unikernels is considered in [89]. The same work assesses the alternative virtualization options in terms of allocation time of a simple Network Memory Server and demonstrate the lower service allocation time of the unikernel option.

In our understanding, two of the related works consider both service operation and elasticity, however taking into account only the resource allocation aspect of the latter. In [90], the authors conduct a performance comparison between unikernel and container based implementations of a firewall, in the context of a fault-tolerance scenario, reporting that containers exhibit the best network performance (i.e., in terms of latency and throughput) and service instantiation time. Paper [91] compares a traditional VM with alternative container technologies and a unikernel-based implementation, all hosting a web-service. The last option achieves the lower boot-up times and higher network throughput, while the container option the best CPU and memory consumption efficiency.

As we see in Table 6, all the aforementioned studies consider one or two virtualized network services (or application functions) with the web-service being the most commonly used, since it may represent a number of REST-based services. We also observe that almost all of them investigate containers, mostly docker, while three of them (i.e., [84], [89] and [91]) consider alternative container virtualization technologies. Furthermore, papers [78], [86] and [84] assess different unikernel flavors, while works [78] and [87] consider multiple implementations of the same application.

In summary, the relevant papers provide comparative evaluations of alternative lightweight virtualization technologies with respect to service operation and/or assurance aspects. However, our approach is the only one that considers all of these aspects, including virtualized service deployment and removal times, as well as the impact of elasticity processes on server resource-efficiency and service performance. In contrast to the related works, our analysis (i) clearly targets the specific context of edge clouds; (ii) focuses on all basic relevant service and cloud operations; (iii) considers containers, multiple unikernel flavors and implementations of the same service; and (iv) targets to identify the performance trade-offs of alternative lightweight virtualization options, so they can be appropriately tuned, even by mixing multiple technologies.

3.4 Edge Cloud Experimentation Environment

Here, we detail our experimentation infrastructure, including its basic components and interactions.

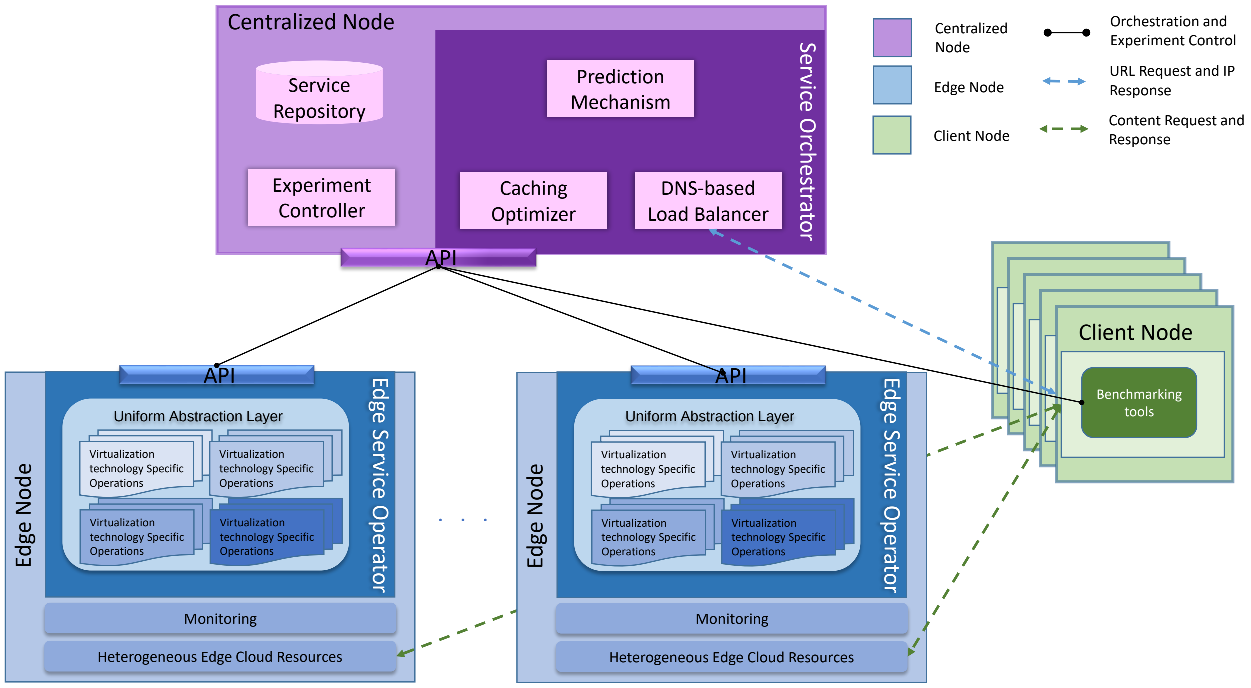


Figure 24: Experimentation environment

Our edge cloud experimentation environment is investigating the proposed edge cloud paradigm and its core design requirements, while demonstrating the following novelties: (i) implements all basic edge cloud operations, including the deployment, removal and operation of virtualized web-based services; (ii) realizes web load prediction and balancing, as well as both horizontal and vertical elasticity; (iii) employs heterogeneous lightweight virtualization technologies as well as different implementations of particular applications to realize adaptability of real deployments in various circumstances, e.g., rapid changes in users requesting content or resource availability; and (iv) is extendable to support new services, virtualization technologies, orchestration workflows and corresponding mechanisms.

Our experimentation facility (i.e., Fig. 24) comprises of three node clusters: the centralized (i.e., purple colored), the edge (i.e., blue colored), and the client nodes (i.e., green colored), all residing at our SWN test-bed [56]. We detail the three parts of our infrastructure and their basic components below, as well as their basic interactions.

The centralized node accommodates the *Service Orchestrator*, the *Experiment Controller* and the *Service Repository*. The *Service Orchestrator* is implemented in Node-

RED [22], a modular programmable environment utilizing a browser-based flow editor. As shown in the top of Fig. 24, it comprises of three standalone components: (i) the *DNS-based Load Balancer*; (ii) the *Prediction Mechanism*; and (iii) the *Caching Optimizer*. A brief presentation of these components follows:

- The *DNS-based Load Balancer* realizes load balancing over heterogeneous virtual resources, i.e., assigns client requests to particular servers, in a round-robin fashion. It supports heterogeneous virtual resources through different IP subnets. For simplicity, the URL indicates the content requested and the type of virtual resource. It maintains a list of active nodes serving content in cooperation with the *Caching Optimizer*, i.e., the latter provides notifications on all additions or removals of virtual resources. *DNS-based Load Balancer* is also tracking the content requests over fixed time intervals, i.e., 30 sec, in our case. Such information is the input of the *Prediction Mechanism*.

- The *Prediction Mechanism* utilizes the client requests' status from the previous component to predict the forthcoming load, based on historical information it maintains. Currently, we support two alternative load prediction mechanisms, the *Exponential Moving Average (EMA)* and the *Seasonal Autoregressive Integrated Moving Average (SARIMA)* [92]. *EMA* considers the recent measurements as more significant and its formula is: $EMA_t = \frac{2}{n+1} * R_t + (\frac{n-1}{n+1}) * EMA_{t-1}$ (R_t expresses the current requests' value). *SARIMA* extends *ARIMA* to consider seasonal trends, defined as *SARIMA(p,d,q)(P,D,Q)m*, where the parameters in the first (i.e., p, d, q) and the second set of brackets (i.e., P, D, Q) indicate the trend and seasonal parameters (i.e., autoregression, difference, and moving average orders), respectively. The mechanisms are implemented as NodeRED modules, so it is straightforward to introduce new models, however this work is not focusing on prediction aspects.

- The *Caching Optimizer* is responsible for controlling the virtual resources, including determining the quantity and location of resources to be deployed or removed, depending on the input of *Prediction Mechanism*. In our experimental analysis, we assume that each VM or container can serve up to a fixed number of requests. Consequently, the number of resources to be deployed or removed is calculated from the *Caching Optimizer* based on the estimation of the upcoming content requests and the total existing capability of the edge nodes to handle the web load, i.e., by subtracting the latter from the former and then dividing the result by the above fixed number. In case the result is negative or positive, a scale up or down event is triggered, respectively. The *Caching Optimizer* decides to deploy new service entities to the servers with the lowest number or remove existing ones from the nodes with the maximum number of entities, i.e., due to the homogeneous hardware of our test-bed servers. However, the *Caching Optimizer* receives not only the status of virtual entities but also recent resource allocation monitoring information from all nodes, which could be the basis for more sophisticated placement mechanisms. In case of a new deployment, it first confirms that the new virtual resource is up and running (i.e., through the *Edge Service Operator*) and then communicates the new IP address to the *DNS-based Load Balancer*. Whenever it removes an existing resource, it first notifies the latter, so no new users request content, waits for existing communication to complete, and then removes the particular virtual entity.

The centralized node also accommodates the *Experiment Controller* and the *Service Repository*. The former component utilizes custom scripts specifying the configuration of experiments (i.e., initiates cloud resources and client requests) and collecting the results from both client and edge nodes through the API interface. The latter stores, locates and communicates both unikernel and container images based on a common *Service Repository* API for heterogeneous virtual resources. It is built on top of a private Docker repository and a custom script handling unikernel images. The repository is utilized whenever a horizontal elasticity event is triggered, i.e., virtual resources are being deployed in new edge nodes, otherwise images are already hosted by the latter.

The second part of our facility contains the edge nodes' cluster. Each edge node is equipped with an *Edge Service Operator* and a *Monitoring* component. The former is a standalone software entity being responsible for the manipulation and configu-

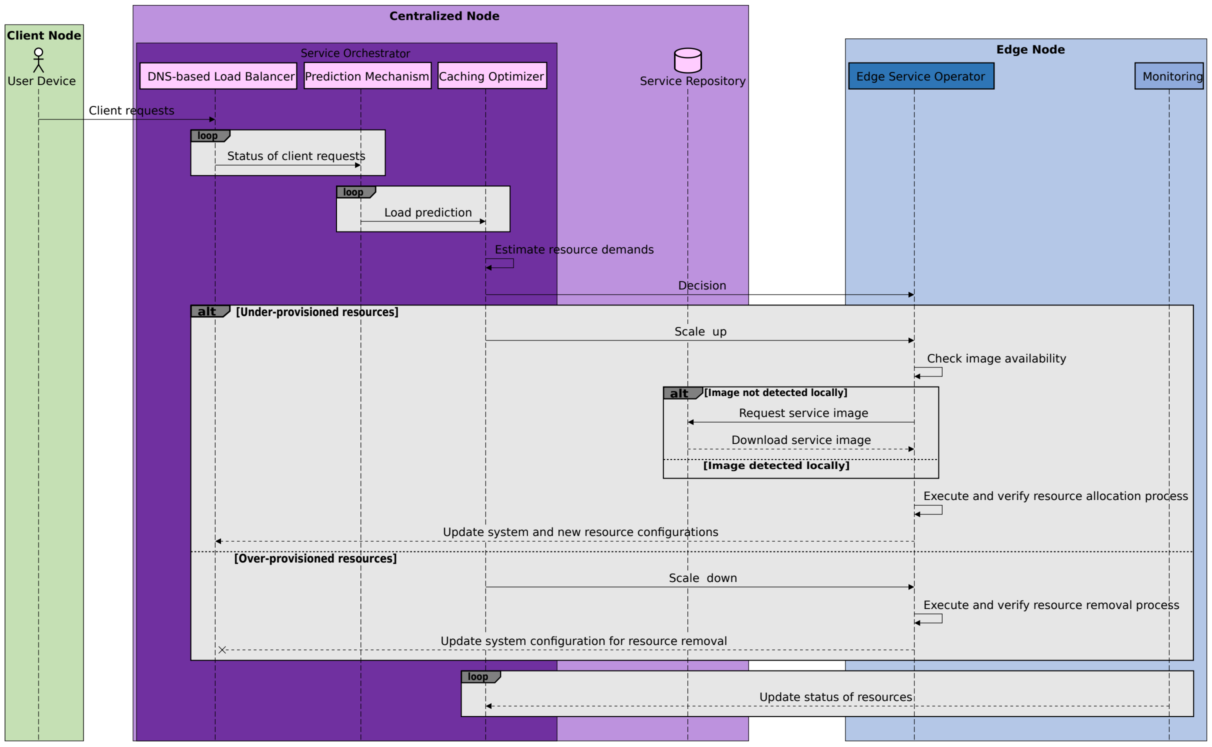

Centralized Node
Client Node
User Device
Service Orchestrator
DNS-based Load Balancer
Prediction Mechanism
Caching Optimizer
Service Repository
Edge Node
Edge Service Operator
Monitoring
Client requests
loop
Status of client requests
loop
Load prediction
Estimate resource demands
Decision
alt [Under-provisioned resources]
Scale up
Check image availability
alt [Image not detected locally]
Request service image
Download service image
[Image detected locally]
Execute and verify resource allocation process
Update system and new resource configurations
[Over-provisioned resources]
Scale down
Execute and verify resource removal process
Update system configuration for resource removal
loop
Update status of resources

Figure 25: Elasticity workflow

ration of edge cloud resources. It uses a uniform abstraction layer that hides the heterogeneity of virtualization resources. For example, the *Service Orchestrator* communicates general requests to deploy virtual resources to the *Edge Service Operator*, which in turn performs the following tasks: (i) locates corresponding virtualization-technology-specific APIs; (ii) identifies the location, i.e., being local or remote, and the details of required images; (iii) allocates the particular resources and assigns IPs to them; (iv) confirms the completion of deployment through a frequent polling process by requesting a tiny-sized content (i.e., every 0.1 sec); and (v) notifies the *DNS-based Load Balancer* upon the completion of the operational task through the API. Finally, the *Monitoring* component collects real-time data information about the physical and virtual resources in terms of resource availability and the status of edge cloud virtual resources.

The last part of our experimentation environment consists of the client nodes hosting our *benchmarking tools*, a custom multi-threaded workload generator tool being responsible for emulating the clients' behavior as well as assessing their performance with different metrics.

In Fig. 24, we also highlight the basic connectivity among the components of the experimentation facility. The centralized node communicates with both edge and client nodes through the API interface for orchestration, monitoring information and experimentation control processes, e.g., the implementation of elasticity events or changes in the configuration of the experiment.

For example, message exchange sequence diagram of Fig. 25 illustrates the interactions between the platform components for the realization of an elasticity process. In this workflow, *DNS-based Load Balancer* keeps track and notifies the *Prediction Mechanism* for the status of content requests, which in turn may request a scale up or down event from the *Caching Optimizer*, i.e., through an *Edge Service Operator*. The latter component downloads the required service image from the *Service Repository*, in the case it is not available locally, and then boots up and verifies a corresponding resource allocation or removal process. The elasticity event completes with the necessary configurations. Lastly, the *Monitoring* component informs periodically the *Caching Optimizer* for the status of cloud resources.

In the next section, we proceed describing our methodological approach.

3.5 Methodology and Assumptions

Here, we detail our methodological approach and provide the relevant assumptions, including on the considered service and virtualization technologies, networking aspects, as well as on the metrics used and the statistical evaluation of our results.

In our investigation, we consider a service that resembles a content delivery platform (e.g., distributing videos, music or other content) or a microservice / network service that hosts a local database and transmits messages via a REST interface. For simplicity, we deploy web servers able to transmit content or messages of different sizes. We also assume that content is embedded in the VMs or containers, so content size impacts relevant image sizes. Web servers are typically used from microservices and are also supported by all virtualization approaches, e.g., even from unikernel flavors at their early implementation stages, consequently serving as a good basis for our comparisons.

Table 7: Image sizes of different virtualized services

Content	C_Nginx	C_Flask	ClickOS	RumpKernel	MirageOS
1MB	22.5MB	85.2MB	6.2MB	34MB	13.4MB
5MB	26.5MB	89.2MB	9.4MB	37.8MB	16.7MB
10MB	31.5MB	94.2MB	14.6MB	42.6MB	20.6MB
20MB	41.5MB	104MB	24.5MB	52.1MB	26MB

Currently, the edge nodes support three common unikernel options, i.e., ClickOS, RumpKernel, and MirageOS. We integrate Nginx web servers to the first two, due to its relative simplicity, high-performance and minimalistic size. We use a simple OCaml-based web server for MirageOS, since it does not support Nginx. Furthermore, we build two lightweight Docker container images, the one hosting an Nginx web server and the other the Flask Python web framework, i.e., to be able to assess also the impact of web server type, marked as *C_Nginx* and *C_Flask*, respectively. Table 7 illustrates the VMs and containers' image sizes we utilize in our experiments, concerning the particular content cache sizes. In our experiments, all unikernel and container technologies are resource-limited to one vCPU and 256MB of RAM. We also set the maximum number of content requests served by all virtualization technologies as 20.

We are currently assuming content requests equally spread among client nodes based on particular patterns and organized in periodic batches. Due to space constraints, we leave a more thorough study on the impact of different user patterns on the performance of the proposed paradigm as a future work, since the current assumption suffices to highlight the particular novelties identified in this chapter.

We implement E2E communication between client and edge nodes, which spans over both physical and virtual networks. Each client retrieves content after looking up the particular URL through the *DNS-based Load Balancer*, which connects the client to an appropriate virtual server node. The E2E path is configured through static routes generated by a bespoke script and the virtual network is implemented through both XEN and Docker bridging. For better performance, we disable the Spanning Tree Protocol (STP) in the virtual network bridges. We also configure all network interfaces with the traffic control (tc) tool to align the network configuration with the scale of our

Table 8: Association of considered metrics with related works' categorization criteria

	Edge Cloud Aspects					
	Service Operation		Elasticity or Fault-tolerance			
Metrics	Server Resource-efficiency	Service Performance	Resource Allocation	Resource Removal	Server Impact	Service Impact
Resource Allocation Time			✓			
Resource Removal Time				✓		
CPU Service and Infrastructure Utilization					✓	
CPU Peak	✓	✓				
Response Time		✓				✓
Download Time		✓				✓
Total Delivery Time		✓				✓
Network Throughput		✓				

experiments.

Furthermore, we extract measurements from both client and edge nodes on service fulfillment, service assurance and communication performance, as summarized below:

- *service fulfillment* with the following metrics: (i) *Resource Allocation Time* is the total duration, in seconds, required for the deployment of a VM or a container, i.e., the amount of time from the *Service Orchestrator* triggering its deployment until it is ready to serve content; and (ii) *Resource Removal Time* expressing the time in seconds to remove a VM or container.

- the *service assurance* metrics apply here are: (i) *CPU Service and Infrastructure Utilization* measuring the CPU utilization of VM or container providing the service and of the edge node, respectively; and (ii) *CPU Peak* quantifying the highest point in CPU utilization in the life-span of the corresponding task, i.e., focusing on worst-case CPU utilization.

- *communication performance* with the metrics: (i) *Response Time* expressing the time, in seconds, pass from the client request to the receipt of the first byte of the response; (ii) *Download Time* representing the total duration, in seconds, required for the transfer of content to the client; (iii) *Total Delivery Time* reflecting the aggregation of *Response* with *Download Time*; and (iv) *Network Throughput* which is the ratio of transmitted data to download time, in MB/s;

Service fulfillment and assurance performance metrics are tightly associate with the service lifecycle that ensures utilizing the necessary resources and maintaining service quality, i.e., also reflected in the communication performance, focusing on the content delivery aspect, in our case. We use a different categorization for the metrics from the considered edge cloud operations, since some metrics are used to assess more than one operation. For clarity, in Table 8 we associate the considered metrics with the categorization criteria of our related works investigation, as presented in Table 6.

Finally, we evaluate our results' statistical accuracy by executing each experiment for an indicated number of times that produces low standard deviations between the replicated runs. All provided figures represent the average values over these runs as histogram plots. For simplicity, we provide the upper bounds of the associated standard deviations, whilst in particular cases, i.e., further supporting the statistical accuracy of our findings, we illustrate our results as box plots that provide supplementary statistical information, such as mean, median, outliers, lower and upper quartiles.

Next section provides our experimental analysis that is based on the presented experimentation platform and the discussed methodology.

3.6 Experimentation Results

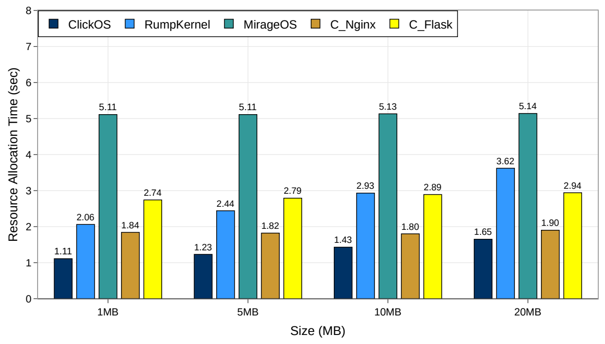
In this section, we validate the main concepts behind the proposed edge cloud paradigm, including the utilization of lightweight and heterogeneous virtual resources, as well as identify strategies for efficient resource allocation and service performance. Our analysis is organized into three scenarios covering essential edge cloud processes: (i) the *Deployment and Demobilization* scenario assessing the deployment and removal of diverse virtualized service entities; (ii) the *Service Operation* scenario quantifying the content delivery performance of alternative unikernel flavors, containers and web server technologies; and (iii) the *Elasticity* scenario considering a complete elasticity workflow that involves the prediction of client demands and a proactive strategy deploying content at the edge cloud. We measure *service assurance* in all scenarios, *service fulfillment* in the first one and *communication performance* in the other two. Our results and produced insights follow.

3.6.1 Scenario 1: Deployment and Demobilization

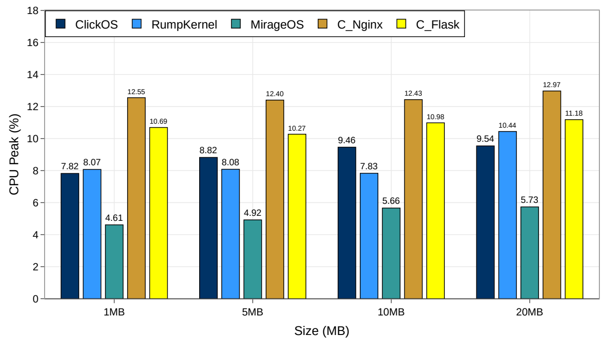
In the first scenario, we assess the individual performance characteristics of diverse virtualization and web server technologies during their deployment or removal, while ranging content size from 1 to 20MB, i.e., impacting VM or container image sizes. We deploy and remove one virtual resource at a time and wait for 3 minutes between different deployments, i.e., to cool off the CPU. We validate the statistical accuracy of our results by replicating each experiment 30 times.

The scenario includes the following two groups of results on (i) *vertical involvement of cloud resources*, assuming the existence of unikernel or container images for the service provisioning, at allocated physical servers; and (ii) *horizontal involvement of cloud resources*, representing the outcome of a horizontal elasticity process allocating new physical resources that should also download service images from the *Service Repository*, residing at the *centralized node*, i.e., assuming that new resources are available instantly. The resource allocation and removal processes concern the booting up or removal of both corresponding virtual entities and web servers.

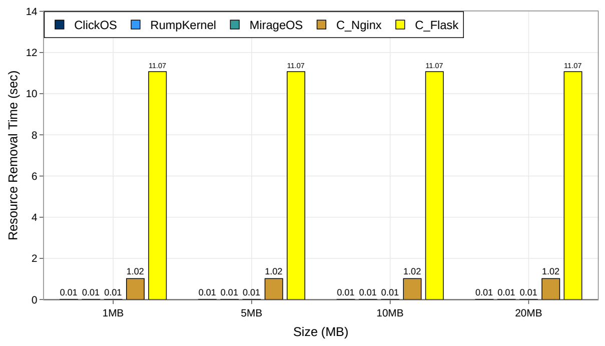
In the case of vertical involvement of cloud resources (i.e., Fig. 26), ClickOS outperforms other approaches in terms of *Resource Allocation Time*, succeeded by C_Nginx (i.e., Fig. 26(a)). For example, ClickOS, in the best case (i.e., of 1MB), can be deployed 4.6 times faster than MirageOS, while in the worst case (i.e., of 20MB) achieves around 15% lower *Resource Allocation Time* from C_Nginx. The improvement ratio of ClickOS is slightly canceled as the content size increases, besides the comparison with RumpKernel. In all circumstances, MirageOS faces the worst performance, however, associated with the most efficient CPU utilization (i.e, Fig. 26(b)). We also notice that



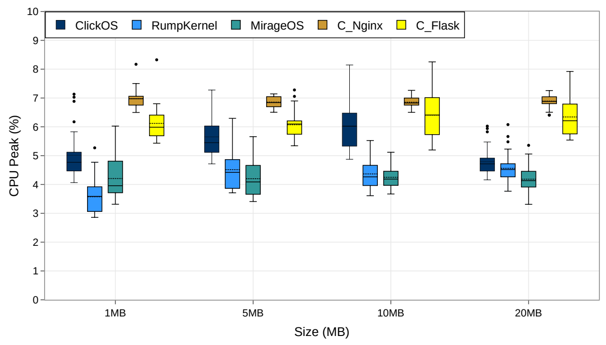
(a) Resource Allocation Time



(b) CPU Peak during resource allocation



(c) Resource Removal Time

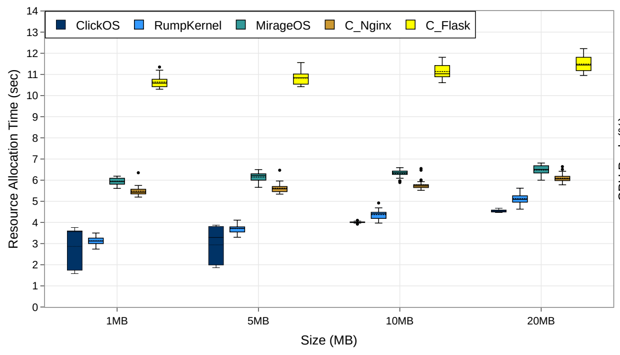


(d) CPU Peak during resource removal

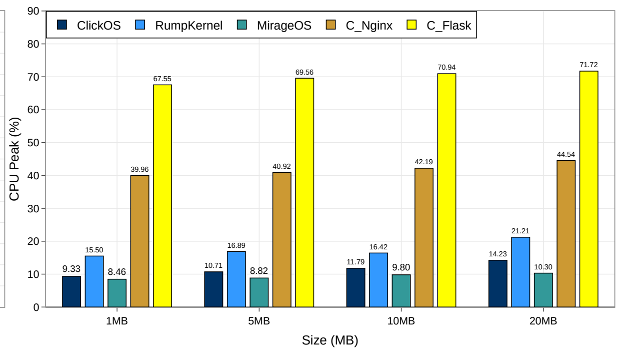
Figure 26: Service fulfillment of alternative virtualized services - vertical involvement of cloud resources

Resource Allocation Time of MirageOS is not affected by content size. The lowest *CPU Peak* value of MirageOS can be justified from its prolonged boot time. We also see in Fig. 26(b) a *CPU Peak* difference between unikernels and containers ranging between 1% and 8%, in favor of unikernels. MirageOS and ClickOS have an almost steady increase of *CPU Peak* per content size increment, while in the other options appears a more fluctuated behavior. The standard deviation between the runs regarding the *Resource Allocation Time* and *CPU Peak* does not exceed the value of 0.19 and 0.36, respectively in all cases.

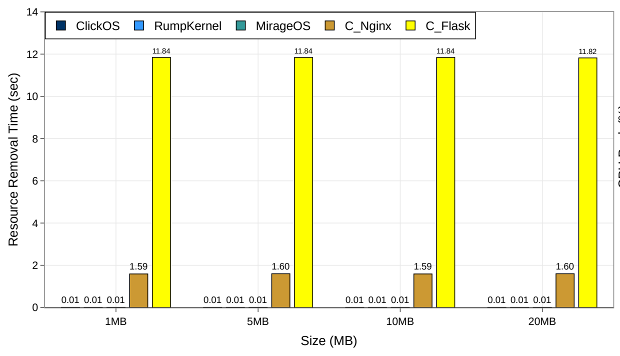
In Fig. 26(c), we depict the *Resource Removal Time* with an equivalent experiment to the above. We see that such metric is not influenced by the content size, in all virtualization and web server options. Furthermore, all unikernel flavors can be removed at almost zero time, highlighting their significant performance advantages in scaling down processes. Fig. 26(d) illustrates as a box plot the associated *CPU Peak* values with the resource removal process. We observe a few outlier values, mostly in the case of ClickOS, and mean / median values that are roughly similar. The results appear symmetrically distributed in all virtualization options, where MirageOS prevails with lower *CPU Peak* values for 5 to 20MB content sizes. Complementary, we observe that unikernels are approximately characterized by 1% to 4% lower mean *CPU Peak* values, compared to container options, mainly caused by their more lightweight processes.



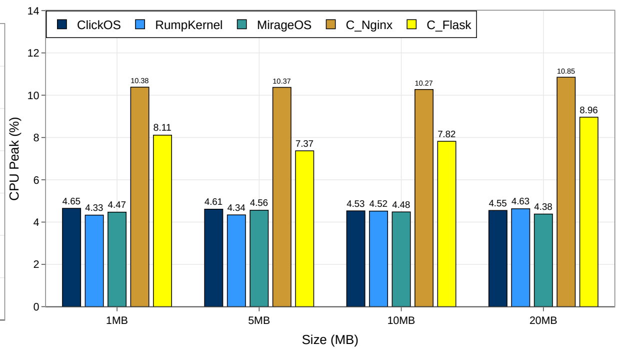
(a) Resource Allocation Time



(b) CPU Peak during resource allocation



(c) Resource Removal Time



(d) CPU Peak during resource removal

Figure 27: Service fulfillment of alternative virtualized services - horizontal involvement of cloud resources

Regarding the horizontal involvement of cloud resources, we observe that ClickOS achieves overally the best performance for all content sizes, in terms of *Resource Allocation Time* (Fig. 27(a)). For example, ClickOS is characterized by a 3.7 times lower boot-up time compared to C_Flask, in the best case (i.e., with 1MB content), while RumpKernel attains close results (i.e., the 91%) with a 10MB content size. We also notice a symmetrical distribution of results with ClickOS being associated with a higher variability (i.e., with 1 and 5 MB content), i.e., reflected in a inter-quartile range around 2. We also notice a steady improvement ratio reduction with the gradual increase of content size, i.e., when comparing ClickOS with all other solutions besides Rumpkernel. The standard deviations of results on *Resource Allocation Time* do not exceed value 0.92.

Furthermore, as shown in Fig 27(b), MirageOS contributes to a lower *CPU Peak*, in each case. This can be justified again due to the prolonged booting up time of MirageOS. Additionally, the high *CPU Peak* of C_Flask may be attributed to its large image size. We also observe that unikernels are more resource-efficient than containers in terms of CPU utilization, as reflected in their at least 20% lower *CPU Peak* values. This difference is much higher than the case of vertical involvement of resources and is attributed to the CPU consumption required for the image downloading process. We also notice an almost steady increase in *CPU Peak* with content size, for all virtualization cases. The standard deviation of *CPU Peak* values ranges from 0.6 to 1.5.

Here, we see that VM or container image download time impacts these results, gradually canceling the advantages of tiny VM sizes (or virtualization solutions with rapid boot up times) as content size increases, i.e., with content that is either embedded or downloaded together with the images.

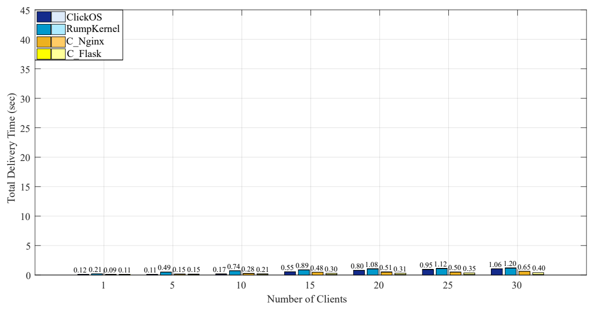
As we show in Fig. 27, the *Resource Removal Time* has an almost identical behavior as in Fig. 26(c), highlighting once more the significant performance advantages of unikernels in scaling down processes. Furthermore, RumpKernel is characterized by the lowest *CPU Peak* value (i.e., Fig. 27(d)) for removing resources with lower-sized content, while MirageOS slightly outperforms RumpKernel for larger-sized content. We also note that all unikernel options have equivalent *CPU Peak* values, which are independent of the content size, while *CPU Peak* in containers marginally oscillates. Complementary, we observe that unikernels achieve up to 6% lower *CPU Peak* values compared to containers. This could be attributed to the different shutting down processes between Docker and XEN, rather than to the content size. In overall, the standard deviations of *Resource Removal Time* and *CPU Peak* do not exceed the values of 0.01 and 1.1, respectively.

Here, we summarize our findings from the first scenario: (i) both unikernel and container-based options can be quickly manipulated in the context of a vertical involvement of cloud resources, with ClickOS achieving the best resource allocation time (i.e., at least 15% lower in contrast to C_Nginx); (ii) unikernel options consume around 1% to 8% less CPU resources during their deployment compared to containers, with MirageOS consuming at least 3% and 6% lower CPU resources in contrast to other unikernel flavors and containers, respectively; (iii) for horizontal elasticity allocation processes, the required image downloading brings further advantages to unikernels with respect to containers, in terms of *CPU Peak* and *Resource Allocation Time*, due to their tiny sizes, e.g., MirageOS achieves an at least 31% lower *CPU Peak* value and ClickOS boots-up at least 1.35 times faster from C_Nginx, respectively, however these are gradually canceled with the content size; (iv) in both vertical and horizontal involvement of cloud resources all unikernel options can be instantly removed, i.e., in 0.01 sec, independently of the content size, while achieve up to 6% lower *CPU Peak* values compared to containers.

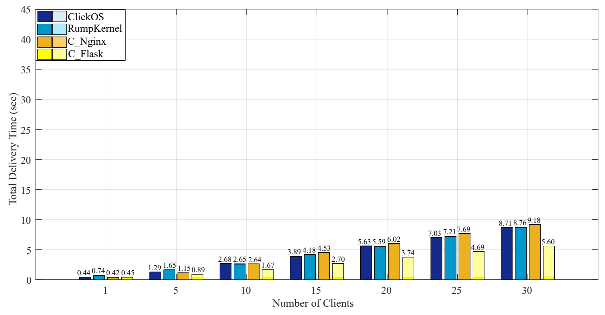
3.6.2 Scenario 2: Service Operation

In the second scenario, we evaluate the operation of the assumed content delivery service in terms of resource efficiency and performance with all considered virtualization and web server options, despite MirageOS. We omit the latter for two reasons: it did not perform well in the previous scenario and it faced stability issues. We also range the content sizes from 1 to 20MB. 10 replications of the runs sufficed for statistically accurate results. In the case of *Response* and *Download Times*, we calculate their standard deviation among different batches rather than of individual client requests, since they exhibit relatively high fluctuations due to the network. All metrics have an equivalent behavior for 5MB, 10MB and 20MB content.

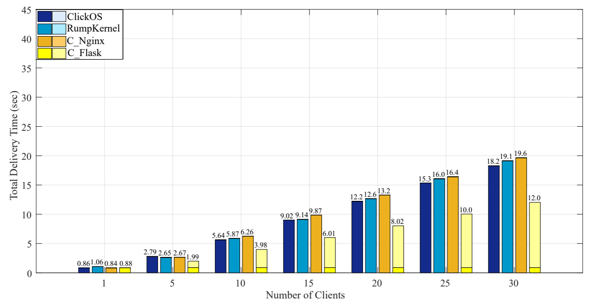
In Fig. 28, we observe that, in general, C_Flask outperforms other approaches in terms of *Total Delivery Time*, succeeding by ClickOS, RumpKernel, and C_Nginx. The same performance order is detected in the majority of cases, i.e., between 10 to 30 clients and for content sizes 5, 10 and 20MB. In the same figure, we notice an almost steady increase of *Total Delivery Time* with the number of client requests. For



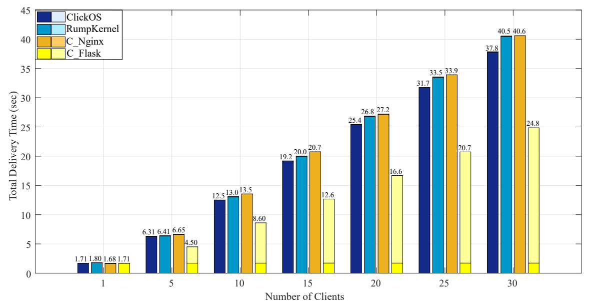
(a) Clients' Total Delivery Time with content size 1MB



(b) Clients' Total Delivery Time with content size 5MB



(c) Clients' Total Delivery Time with content size 10MB

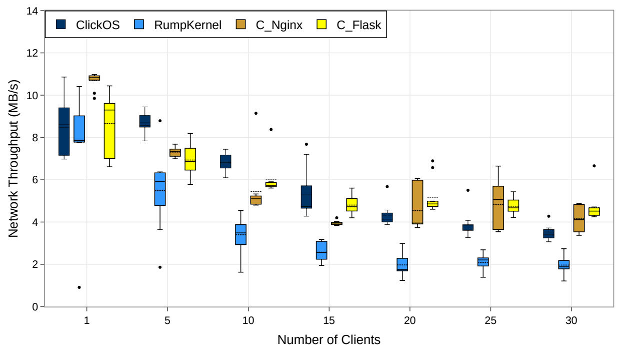


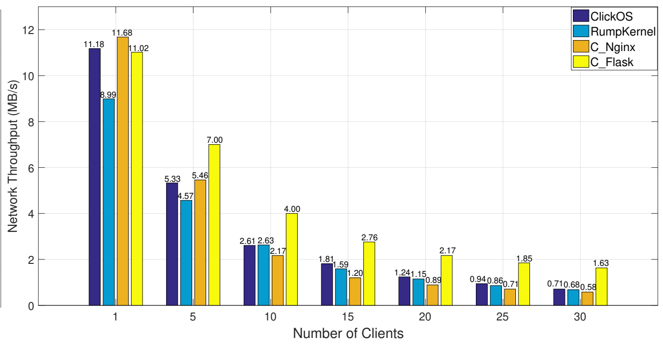
(d) Clients' Total Delivery Time with content size 20MB

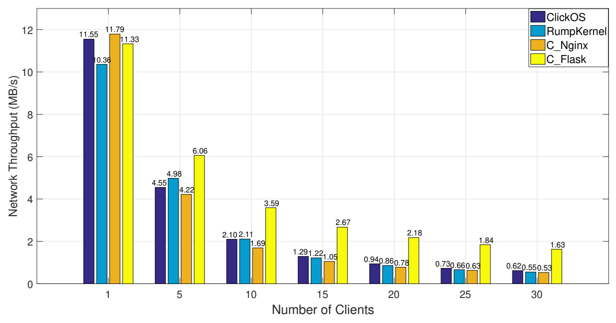
Figure 28: Communication performance - Total Delivery Time as Response (light colored) plus Download Time (dark colored)

example, such increments for 5MB to 10MB or 10MB to 20MB range from 1.4 to 2.4 times, for all approaches.

From an analysis of *Total Delivery Time* and its constituting *Response* and *Download Times*, we see that: (i) *Response Time* is independent of content size and the number of client requests for ClickOS, RumpKernel and C_Nginx, which does not exceed value 0.06 sec; and (ii) C_Flask *Response Time* increases with content size and the number of client requests, but with an almost non-fluctuating behavior regarding *Download Time*, which is not the case in the other solutions. This attitude can be justified by the different approaches of Nginx and Flask web servers in terms of requests' handling. Practically, we use the simplistic web server integrated with Flask, which struggles to handle all client requests (i.e., also reflected a high *CPU Peak* value, as shown in Fig. 30), however leading to a better *Download Time* because it desynchronizes the parallel web requests. The standard deviation values for *Download* and


(a) Clients' Network Throughput with content size 1MB


(b) Clients' Network Throughput with content size 5MB


(c) Clients' Network Throughput with content size 10MB

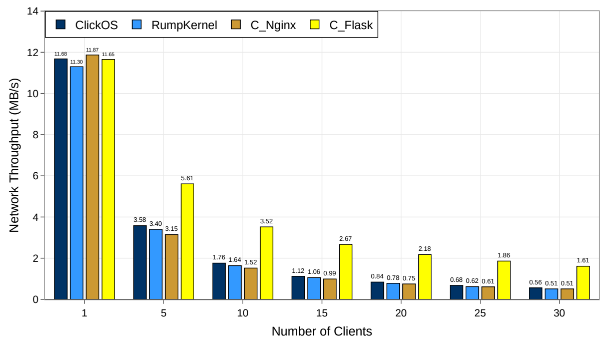
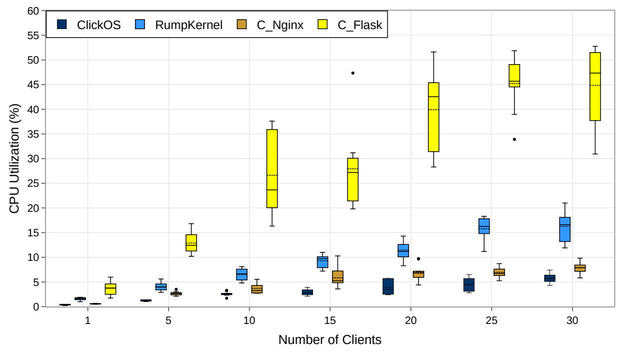

(d) Clients' Network Throughput with content size 20MB

Figure 29: Communication performance - Throughput

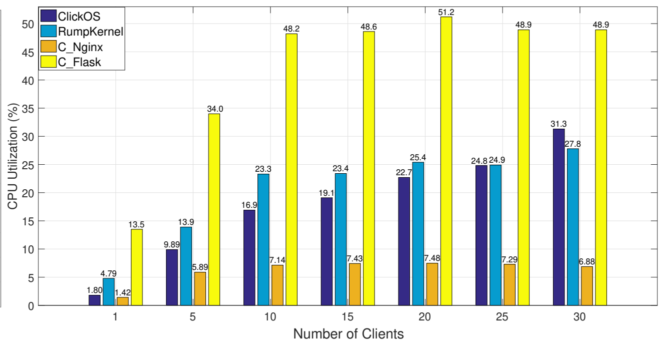
Response Time range from 0.004 to 1.65 and 0.003 to 0.21, respectively, i.e., causing borderline differences between the third and fourth-placed approaches only.

As depicted in Fig. 29, in the majority of runs, C_Flask achieves the higher *Network Throughput* per client. We identify an almost resembled performance pattern among *Total Delivery time* and *Network Throughput*, i.e., due to the significance of the networking aspect in the considered application. The figure also indicates a decline of *Network Throughput* with client requests' number, but with higher fluctuations, compared to *Total Delivery Time's*, i.e., around 6% to 73% and 9% to 69%, for C_Nginx and ClickOS, respectively. The standard deviation of *Network Throughput* ranges from 0.07 to 3.83, but without impacting the above arguments.

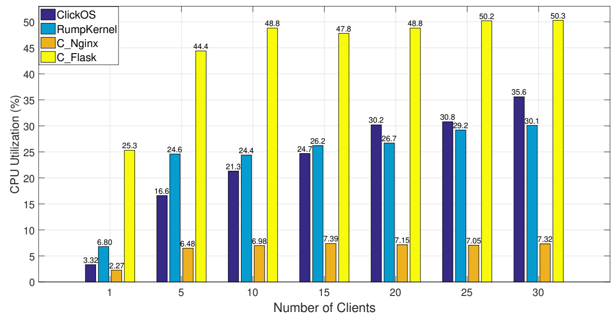
In Fig. 30(a) and 30(d), we see that in most cases C_Nginx outmatches the other solutions in terms of lowest *CPU Peak*, i.e., at least by 8% to 10%. However, in all circumstances C_Flask faces the highest *CPU Peak* value and the greatest variability,



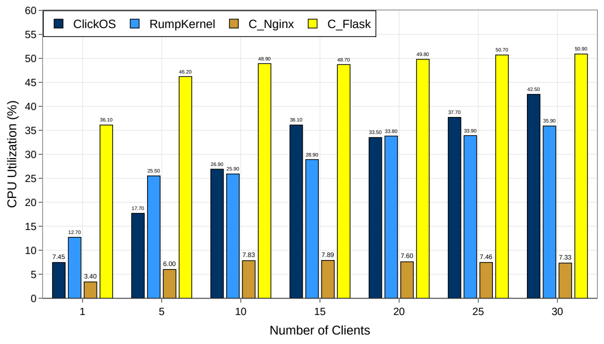
(a) CPU Peak utilization with content size 1MB



(b) CPU Peak utilization with content size 5MB



(c) CPU Peak utilization with content size 10MB



(d) CPU Peak utilization with content size 20MB

Figure 30: Service assurance - CPU Peak

highlighting that web service technology mattered more for *CPU Peak* than the virtualization technology, since both C_Nginx and C_Flask use containers. We also see a diverging behavior between unikernels and containers. For containers, *CPU Peak* values are increased until a particular value and then fluctuate, approximately, 48% to 50% and 7% to 8% for C_Flask and C_Nginx, respectively. Moreover, we see that *CPU Peak* increases more rapidly with a lower number of requests, as content size increases. Such attitude of C_Flask relates to its struggle to cope with high loads. Regarding unikernels, there is an increasing trend in *CPU Peak*, i.e., its increase ratio is gradually being reduced with the number of requests. The standard deviation of *CPU Peak* fluctuates from 0.04 to 9.34, without affecting the performance differences in each individual case.

We now summarize our findings as follows: (i) C_Flask achieves the best performance in terms of *Total Delivery Time* by distributing the content up to 3.4 times faster and consuming 3.18 times more *Network Throughput*, compared to RumpKernel, but it faces up to 40.7% higher *CPU Peak* values, in contrast to ClickOS, which aspect is important, especially for resource-constraint edge cloud deployments; (ii) ClickOS and C_Nginx are descent options for the operation of considered content delivery service, with the former being, in general, characterized by the best *Total Delivery Time* and *Network Throughput*, while the latter by the lower *CPU Peak* values; (iii) unikernels, compared to containers, have a more stable increasing of CPU utilization with clients' number, e.g., offering better conditions for a relevant prediction algorithm; and (iv) the web service technology used matters, especially for CPU resource consumption, with a *CPU Peak* difference between C_Nginx and C_Flask reaching up to 43%, in favor of the former.

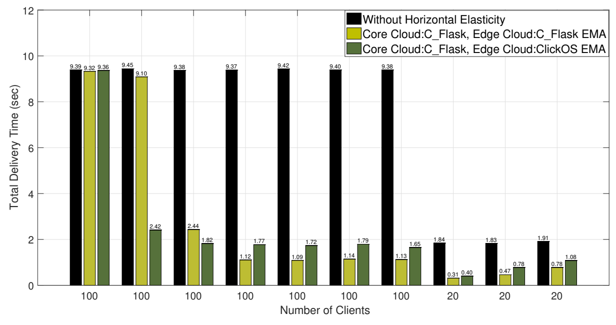
3.6.3 Scenario 3: Elasticity

After evaluating the behavior of heterogeneous virtualized web-based service entities on the basic edge cloud processes of resource allocation, removal and operation, we now assess their performance in a complete cloud environment, i.e., involving both core and cloud resources as well as scale up and scale down events triggered from predicted demands. We reproduce a cycled user pattern in the form of $\{100, 100, 100, 100, 100, 100, 100, 20, 20, 20, ...\}$, assuming a periodic rapid shift on client demands, alternating between high and low numbers of requests. Since the first two scenarios revealed that C_Flask achieves the lower *Total Delivery Time* in service operation but with a high *CPU Peak* and ClickOS very low *Resource Allocation Time* and *CPU Peak*, it comes natural to utilize the former technology at the core cloud and the latter at the edge. To evaluate the efficiency of this strategy, we contrast the performance of this setting against an equivalent one with C_Flask instead of ClickOS.

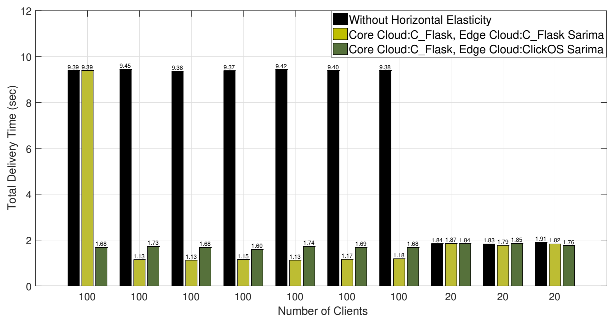
In practical terms, the core cloud solely serves the users up to points with rapid demand shifts. At these particular times, there is a rapid deployment of service nodes at the edge cloud, which are removed when the demand drops. Such elasticity events are guided from the assumption that each involved web server handles up to 20 requests, i.e., resources are being deployed for every 20 requests. For example, 100 clients should ideally utilize a C_Flask container at the core cloud and four edge nodes, i.e., either with ClickOS or C_Flask. However, this depends on the accuracy of the load prediction. The content size for the elasticity experiments is 5MB.

Our test environment assumes both an accurate and a moderate prediction mechanism, i.e., based on *Seasonal Autoregressive Integrated Moving Average (SARIMA)* and *Exponential Moving Average (EMA)*, respectively, so we also investigate the impact of demand prediction accuracy on both service fulfillment and assurance.

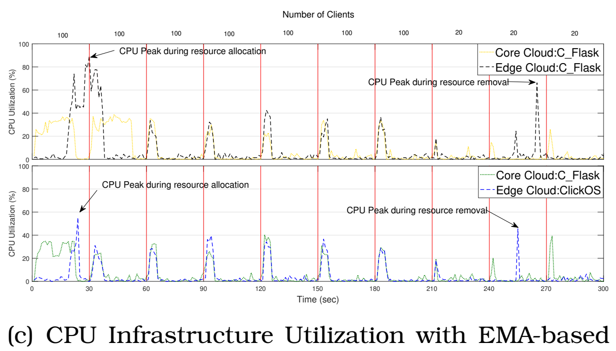
Here, we do not focus on the prediction technology, rather than on its importance in the studied context. For the same reason, *SARIMA* is selected as a model that perfectly predicts the considered user pattern. According to our stepwise validation scheme based on the *root mean squared error (RMSE)*, the obtained parameters are *SARIMA(0,1,0)(0,1,0)10*. The first user pattern cycle does not produce a response of the system, since it is used for the training of prediction mechanisms. Finally, we set parameter *n* equal to 3 in the *EMA* formula, i.e., takes into account the last three inputs.



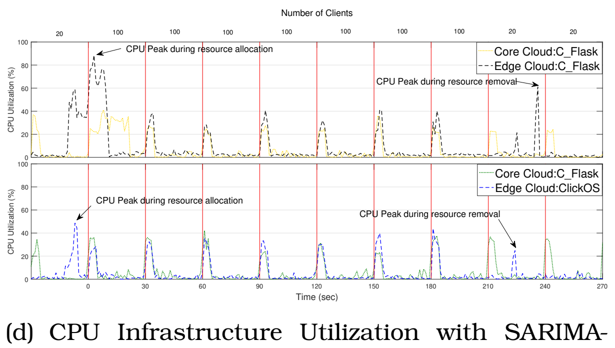
(a) Total Delivery Time with EMA-based load prediction



(b) Total Delivery Time with SARIMA-based load prediction



(c) CPU Infrastructure Utilization with EMA-based load prediction



(d) CPU Infrastructure Utilization with SARIMA-based load prediction

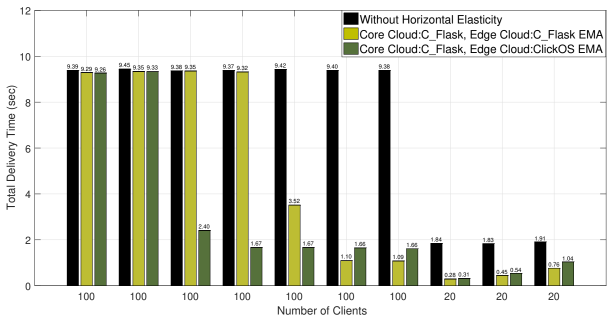
Figure 31: Horizontal elasticity with 100Mbps bandwidth for service orchestration

We have two sets of results, assuming both high (i.e., 100Mbps) and low (i.e., 10Mbps) bandwidth between *edge nodes* and *centralized node*, i.e., to evaluate the impact of network capabilities for service orchestration. Our goal is to assess both adequate and limited network resources, for the scale of our experiment, so we can get an indication whether the evolution of networking technology suffices (e.g., improving bandwidth) or strategies like the proposed should complement the latter, for the best outcome. In both cases, users request and download content over 200Mbps connections.

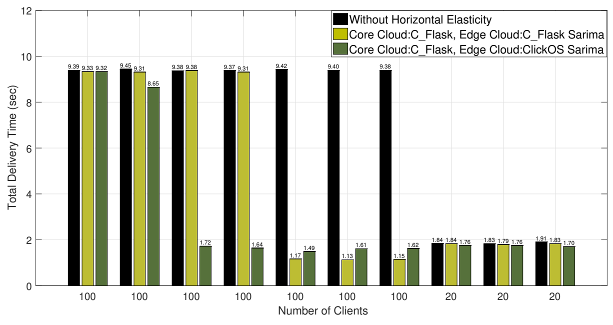
For clarity purposes, we depict the *CPU Infrastructure Utilization* figures with (i) indicative core and edge cloud nodes, since the visualization of all nodes produces a complex outcome, because the orchestration processes are not perfectly synchronized among the nodes; (ii) top and down subplots presenting the performance outcomes when using C_Flask and ClickOS at the edge nodes, respectively; and (iii) the cycled user pattern is visualized for one user batch sooner for *SARIMA* only (i.e., in Fig. 31(d) and 32(d)), so the impact of accurate prediction on *CPU Infrastructure Utilization* can be illustrated more clearly.

We start with the cluster of results assuming a high-bandwidth service orchestration. As expected, the involvement of edge cloud resources improves significantly the *Total Delivery Time*, i.e., as shown in Fig. 31(a) and 31(b). In the same figures, we observe that, independently of the load prediction strategy, ClickOS overally responds more rapidly compared to C_Flask, however C_Flask outmatches ClickOS in terms of *Total Delivery Time*, once the new edge cloud resources have been deployed. We also notice that *EMA*, compared to *SARIMA*, leads to further delays in the response of the system to the initial increased number of requests (switching from the 20 users of the training cycle to the 100 users of the regular cycle), i.e., takes one more round of users to respond. This first round is characterized by an equivalently high *Total Delivery Time* for the three approaches, indicating zero deployed edge cloud resources. The combination of accurate load prediction of *SARIMA* and rapid allocation capabilities of ClickOS leads to a very low *Total Delivery Time*, for all cases. It takes for C_Flask and *SARIMA* one round to respond to the high load, due to the less rapid resource allocation of the former, compared to ClickOS. Fig. 31(a) also highlights that *EMA* needs more rounds to predict load, leading to over-provisioning of resources in the last three user rounds, due to the associated gradual demobilization of resources.

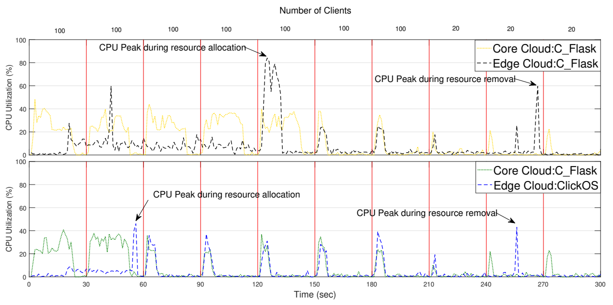
In Fig. 31(c) and 31(d), we notice the following: (i) ClickOS consumes less CPU resources compared to C_Flask for both allocation (i.e., about 30% and 35% less in *EMA* and *SARIMA*, respectively) and removal (i.e, 20% and 35% less in *EMA* and *SARIMA*, respectively) of new resources, both in terms of *CPU Peak* and duration of CPU involvement; (ii) *EMA* leads to a gradual response to the load changes, due to its slow prediction, while *SARIMA* responds rapidly, i.e., the latter allocates and removes resources almost 25 and 30 seconds sooner, corresponding to the first round of responses in the system, respectively; and (iii) the over-provisioning of resources due to inaccurate prediction from *EMA* is reflected to the lower *CPU Infrastructure Utilization* of core cloud towards the end of the corresponding figure curve, i.e., after the 210 sec of Fig. 31(c).



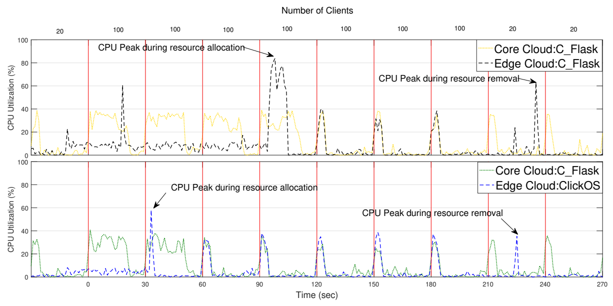
(a) Total Delivery Time with EMA-based load prediction



(b) Total Delivery Time with SARIMA-based load prediction



(c) CPU Infrastructure Utilization with EMA-based load prediction



(d) CPU Infrastructure Utilization with SARIMA-based load prediction

Figure 32: Horizontal elasticity with 10Mbps bandwidth for service orchestration

In the case of low bandwidth in service orchestration, there is an extension in image downloading time for the allocation of new virtual resources during the required vertical elasticity events, which is expected to impact the larger virtual entity sizes more. This behavior is manifested in the content delivery, since ClickOS is now allocated for the third user batch and C_Flask for the fifth, in the case of *EMA* (i.e., Fig. 32(a)). However, the accurate load prediction of *SARIMA* impacts marginally ClickOS, since it is ready to serve some users from the 2nd user batch (i.e., Fig. 32(b)). Consequently, the rapid booting up and small image size of ClickOS lead to a partial mitigation of performance issues caused by limited network resources in service orchestration.

An equivalent behavior can be seen in the *CPU Infrastructure Utilization* (i.e., Fig. 32(c) and 32(d)), where it takes more time for C_Flask to appear at the edge cloud and serve users, i.e., boots up almost 75 and 65 sec after ClickOS is up, for *EMA* and *SARIMA*, respectively. Such delay leads to an impact of around 30% to 40% on the core cloud's *CPU Utilization*, since resources are offloaded to the edge at a later stage. Things are better for resource removal, since there is no need to download images. Contrasting the impact of the two load prediction approaches on *CPU Infrastructure Utilization*, *SARIMA* leads to both sooner resource allocation and removal.

We now summarize our findings from the third scenario: (i) unikernels bring significant benefits in terms of responsiveness to rapid changes in the web load, e.g., ClickOS appears at the edge cloud and starts serving users up to 75 sec sooner than C_Flask; (ii) ClickOS is characterized by up to 40% and 35% lower *CPU Peak* values with respect to C_Flask, i.e., corresponding to the resource allocation and removal processes, respectively; (iii) inaccurate load prediction may lead to both over-provisioning or under-provisioning of resources, with *SARIMA* responding up to 25 and 30 sec earlier, in the respective instances of resource allocation and removal, compared to *EMA*, but such issues are more severe with containers, i.e., the system re-adjusts itself in a slower manner.

3.7 Design Guidelines for Edge Cloud Systems

The general deployment of computing facilities in telecom networks is introducing new challenges on their management, control and operation. Apart from interaction and integration with the underlying transport network [93], or the proper selection of the compute environment for deploying a given service [94], there are additional aspects of relevance to take into consideration, especially the possible virtualization approach [12] to follow, according to the network and service circumstances. This is even more evident with the general adoption of cloud-native approaches by different technological paradigms, in support of virtualized services [81] - [83].

In this context, we build-up on our research results towards providing design directions for 5G and beyond edge cloud infrastructures, which are aligned to the conceptual edge cloud platform illustrated in Fig. 33. The latter operates over multiple edge cloud Points of Presences (Edge PoPs), as well as a core cloud deployment (i.e., a next-generation central office - NGCO). A centralized orchestrator manages all network, compute and storage resources as well as service nodes, through Edge PoP managers that are deployed in every edge cloud. The traffic load is balanced between the Edge PoPs through a load balancer controlled from the same orchestrator. The platform supports alternative virtualization approaches and relevant packet processing optimization mechanisms, as well as implementations of particular application functions (AF) and virtual network functions (VNFs), i.e., exhibiting diverse performance trade-offs. The orchestrator receives service requirements through a Northern Interface (e.g., like in NECOS platform [95]), selects and configures the most appropriate virtualization technologies, mechanisms and service nodes, i.e., to deliver the service with the expressed performance requirements.

As a bottom line, we envisage the following capabilities for our conceptual platform and other relevant systems: (i) a virtual resource abstraction layer that supports multiple virtualization technologies and relevant features (e.g., containers and unikernels, eBPF, XDP [79], etc.); (ii) abstract well-design APIs translating uniform primitives to technology-specific compute and network control processes; (iii) intelligent optimization mechanisms selecting and configuring the most suitable virtualization technolo-

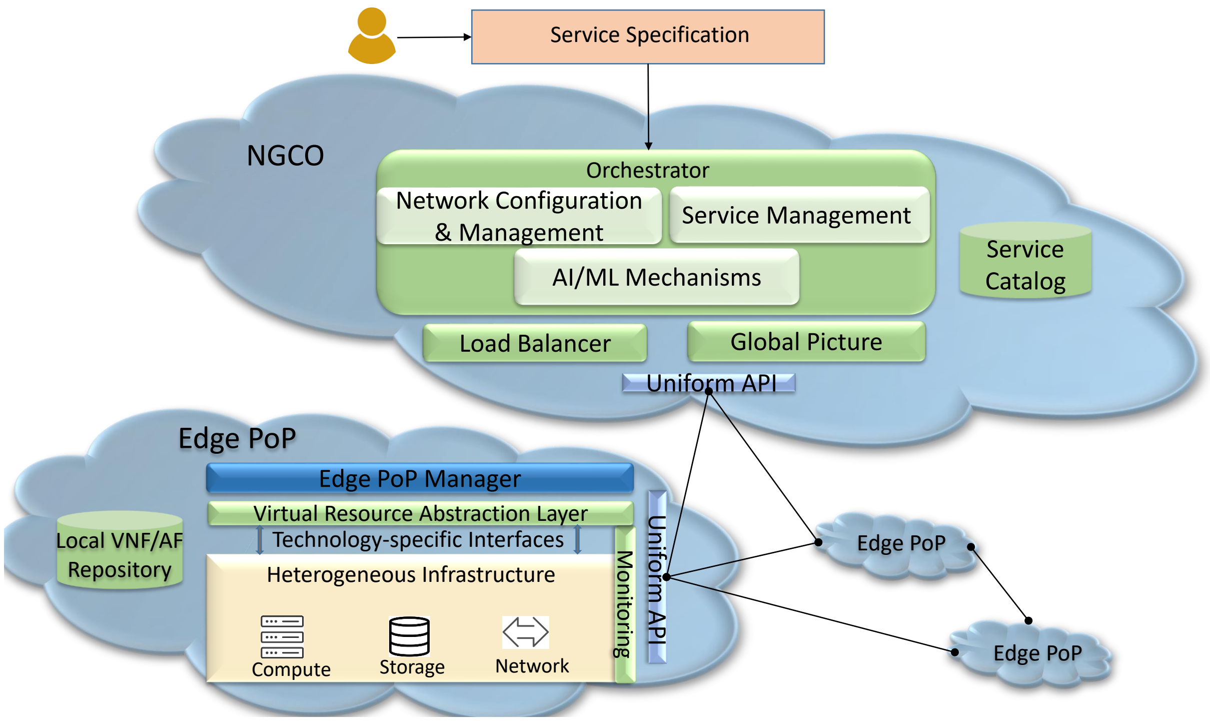


Figure 33: Conceptual edge cloud orchestration platform

gies and service nodes to particular service requirements and network conditions, e.g., targeting ultra-low latency; (iv) novel Artificial Intelligence and Machine Learning (AI/ML) capabilities enabling automation processes, including for fluid elasticity [72] and efficient resource allocation, scaling, load balancing / workload assignment; (v) a monitoring abstraction that provides a global picture to a centralized orchestrator, i.e., of virtual resources, network conditions and client behavior, backed by accurate prediction or rapid detection mechanisms; (vi) lightweight and high-performing service orchestration processes and interactions of involved components, adaptable to expressed service requirements; (vii) edge PoP managers being responsible for the control of virtualized network and application functions in the edge infrastructures, coordinated by the centralized orchestrator; (viii) local VNF/AF repositories hosting alternative implementations of particular virtual network or application functions with multiple virtualization technologies; and (ix) a service catalog enlisting available services, along with a description of the deployment, operational and performance characteristics of their constituting virtual network and application functions.

3.8 Conclusions

In the context of this chapter, we conduct an extensive comparative evaluation of alternative builds of virtualized exemplary web-services, involving both unikernels and containers, and identify that each option is characterized by particular performance trade-offs. Our experiments utilize a novel bespoke edge cloud experimentation infrastructure that considers virtualized service deployment, removal, operation, as well as both vertical and horizontal elasticity processes. We consolidate the gained insights from our realistic experiments and define a conceptual edge cloud orchestration platform for 5G and beyond networks with its key design guidelines.

4 Virtualization Technology Shifting for Resource-Efficient Edge Clouds

4.1 Introduction

5G networks and beyond (5GB) [75] are targeting services with challenging requirements (e.g., ultra-low latency, high throughput or for increased channel capacity), such as holographic teleportation, extended reality, or ambient connectivity applications. The relevant research endeavours are mainly focusing on: (i) improved radio spectrum efficiency and the utilization of higher frequency bands to meet stringent performance requirements; (ii) a transformation of telecommunication network through the virtualization of physical network functions, improving its deployment and configuration flexibility as well as reducing the associated capital expenditure (CAPEX); and (iii) the employment of edge cloud deployments, bringing virtualized services and keeping data closer to users, i.e., to improve service performance, reliability, and address privacy issues.

In this context, orchestrating edge cloud resources is a complex and non-trivial task. Internet services operating at large-scales depend on the efficient allocation of server and network resources in many different locations. For example, edge cloud resource availability in given areas may be unfeasible, limited or associated with incompatible virtualization technologies. Furthermore, such systems target mitigating resource exhaustion conditions, data link and server failures, or sudden changes in the workload behavior, as well as meeting particular service performance requirements. Existing orchestration strategies include the manipulation of duplicated virtualized service nodes (i.e., horizontal elasticity), the allocation or removal of physical resources (i.e., vertical elasticity), or service migration to more suitable cloud infrastructures, e.g., closer to users. Such processes may be time-consuming (e.g., the allocation of new physical servers may take hundreds of seconds [96], [95]) or constrained by the performance capabilities of a particular virtualization technology. For instance, legacy cloud deployments may be using traditional virtual machines (VMs), that face slow times for deployment, downloading or scaling of virtual resources.

Applications should also be aligned to the evolution of 5GB ecosystems. The microservices paradigm [11] breaks down traditional software architectures to minimal, single-purpose, communicating service functions or nodes, scaled towards bespoke resource allocation and fault-tolerance. In the prior chapter 3, we propose that such functions could be utilizing heterogeneous virtualization approaches [97], e.g., alternative container [13] and unikernel-based [14] implementations, since each virtualization option is characterized by particular performance characteristics. For example, the robustness of containers makes them suitable for the core cloud side, while the rapid manipulation capabilities of unikernels for the edge, e.g., unikernels can boot up in few ms [15]. Towards this direction, the European 5G-PPP initiative for 5G networks is investigating the benefits of employing alternative container and unikernel-based approaches for edge computing [83].

In the same chapter 3, we conducted an extensive comparative analysis [97] between different container and unikernel builds of exemplary web-services with respect to basic edge cloud operations (e.g., resource allocation/removal, service operation and elasticity), revealing that particular builds match the performance requirements of given edge cloud operations or network conditions, in terms of resource allocation efficiency and service performance.

Here, we argue that edge clouds mobilizing alternative virtualization technologies is an attractive resource optimization strategy, since it can increase the range of edge cloud deployments that can be utilized to better cover a large-scale service deployment, as well as enrich the available resource control options for edge clouds, i.e., mixing and matching alternative virtualized functions with challenging cloud resource optimization requirements. For example, in the case of a service that cannot tolerate disruption, a rapid increase in the workload could be addressed from a quick deployment of unikernels, even by sacrificing run-time performance.

4.2 Contributions and Chapter Organization

4.2.1 Contributions

In this chapter, our approach, called *Virtualization Technology Shifting (VTS)*, targets the 5G networks and beyond agenda through complementing their advanced radio communication facilities with further improved end-to-end service performance based on a better exploitation of the network edge, in terms of cloud resource utilization efficiency. It augments existing cloud orchestration approaches with an additional cloud resource control strategy, tuning the performance trade-offs characterizing the virtualized application or network service functions.

The chapter contributes to the above vision through:

- introducing *Virtualization Technology Shifting*, a new cloud orchestration strategy that exploits the diverse performance and resource demands of heterogeneous virtualization technologies, as well as the corresponding service node builds, to: (i) implement adaptability to dynamic network, service and cloud resource utilization conditions; and (ii) address challenging application requirements;

- detailing a first *VTS cloud resource optimization framework*, as defined from an optimization model and a corresponding system, being able to realize particular performance goals over heterogeneous virtualization and hardware technologies, distributed edge and core cloud deployments;

- providing an *extensive set of simulations* highlighting, from both service and infrastructure viewpoints, (i) the impact of *Virtualization Technology Shifting* against a typical edge cloud orchestration strategy (i.e., horizontal elasticity) utilizing a single virtualization technology; and (ii) the efficient implementation of particular performance goals for a given amount of network and cloud resources; and

- giving an *example integration* of the proposed resource control strategy with a typical Kubernetes-based edge cloud orchestration deployment.

4.2.2 Chapter Organization

The remainder of the chapter is organized as follows. Section 4.3 gives an overview of the related investigations, in contrast to the novelties of the current work. Section 4.4 presents a system model of the introduced cloud paradigm. Section 4.5 details the proposed VTS cloud resource allocation optimization model. Section 4.6 elaborates on our methodological approach and its relevant assumptions. Section 4.7 provides our evaluation results, quantifying the impact of proposed paradigm and model. Section 4.8 presents an example integration of our solution with a typical cloud orchestration framework. Finally, Section 4.9 concludes the chapter.

4.3 Related Works

In this section, we discuss related works sharing common characteristics with our proposal, including alternative cloud resource optimization and scaling strategies, solutions adopting lightweight virtualization, and relevant optimization models.

Cloud resource optimization or autoscaling is typically involving horizontal / vertical elasticity or virtualized service migration. For example, Kubernetes [98], the most commonly used container orchestration framework, utilizes the *Horizontal Pod Autoscaling* (HPA) and *Cluster Autoscaler* (CA) components autoscaling the number of application replicas and server nodes, respectively.

HPA implements a control loop being periodically executed (i.e., its default period is 15 sec). It drives a horizontal elasticity process that adjusts the number of service node replicas according to the following equation: $DesiredReplicas = \lceil (CurrentReplicas * \frac{CurrentMetricValue}{DesiredMetricValue})$. The $DesiredReplicas$, $CurrentReplicas$, $CurrentMetricValue$ and $Desired$-$MetricValue$ parameters indicate the number of replicas after scaling, the currently running replicas, the latest collected metric value and an average target metric value, respectively.

On the other hand, CA checks periodically for unscheduled replicas, by default every 10 sec. It implements a vertical elasticity process that increases the number of nodes, when existing resources are exhausted and cannot host additional replicas. The relevant autoscaling thresholds (e.g., $DesiredMetricValue$) in both HPA and CA are fixed and configured according to context requirements.

A number of works adopt hybrid resource control strategies based on dynamic autoscaling thresholds, e.g., adjusted to the incoming workload. For example, Libra [99] and Copa [100] solutions propose hybrid autoscaling methods jointly employing vertical and horizontal scaling procedures, both based on adaptable thresholds. Libra [99] combines vertical and horizontal elasticity in consequent phases, while [100] applies the two strategies simultaneously (i.e., in a single phase).

A small number of proposals are relevant to our work and introduce edge cloud orchestration strategies based on dynamic shifting among alternative virtualized resources or service nodes. For example, paper [101] introduces a novel cloud orchestration approach, called *service shifting*. It suggests that a network service can be provided with alternative forwarding graphs, including a primary fully-fledged Virtualized Network Function (VNF) graph and a sub-optimal secondary graph (i.e., including only the essential network functions). Thus, the network service can be downgraded from its primary to its secondary graph, in resource shortage conditions, while revert back, whenever there is enough cloud resource availability. The authors conduct a performance evaluation and elaborate on the benefits of integrating their approach in 5G networks, without focusing on a corresponding optimization framework.

The work [102] extends the *service shifting* concept and proposes a *multi-flavored Virtual Network Functions (VNFs)* approach, suggesting that the network functions of VNF graphs can also be provided with alternative less-demanding versions. It mixes and matches different VNF flavors to particular requirements in contrast to single-flavor deployments, in order for service providers to flexibly match multi-flavored network service requests. The authors provide a relevant optimization model and performance evaluation to analyze feasibility and potential benefits, while considering time-efficiency of the model as a future work.

In contrast to the above solutions, we leverage the concept of *Virtualization Technology Shifting* to exploit the performance characteristics (e.g., trade-offs) of alternative virtualization flavors through dynamically shifting among them, with respect to a given resource availability and particular performance goals. We also tailor our solution to common edge cloud operations (e.g., elasticity). For example, we may trade the robust performance of containers for the quick deployment capabilities of unikernels, e.g., whenever there is an emergent need for new resources. Nevertheless, VTS and service shifting approaches could also be combined, e.g., consider particular VNF graphs with alternative lightweight virtualization options.

Furthermore, several other service or network orchestration solutions adopt lightweight virtualization solutions (e.g., unikernel or container-based) due to the resource-efficiency and rapid manipulation capabilities of the latter. These deployments target particular environments, e.g., Internet of Things (IoT), Content-Delivery Networks (CDN) or 5G networks, such as: (i) PiCasso [103], an IoT infrastructure utilizing containers; (ii) FADES [104], an IoT facility, based on MirageOS unikernels [33], offloading single-purpose functionalities from the core to the edge cloud; (iii) ECCO [105] delivering timely road context assessments to vehicles through MirageOS-based edge cloud functions; (iv) Kubeedge [106] and K3S [107] offering IoT-specific container orchestration; (v) papers [46] and [47] investigating CDN deployments that utilize ClickOS [34] unikernels as TCP proxies and content caches, respectively; and (vi) NGPaaS [108] and Superfluidity [109] projects which introduce 5G platforms using ClickOS unikernels.

These papers confirm the choice of important 5G initiatives (e.g, [81], [83]) to consider employing containers or unikernels in edge cloud deployments, as well as highlight that there is no single virtualization solution that matches the requirements of all applications or environments.

Moreover, chapter 2 introduces a novel elastic CDN infrastructure that utilizes content caches based on MirageOS unikernels and change-point analysis for content popularity prediction (i.e., [72]). In the same chapter, we have proposed a CDN platform that delivers content over 5G slices operating on top of the Fed4FIRE+ testbed infrastructure, while supporting heterogeneous unikernel-based technologies (i.e, [110]).

In chapter 3, the above works inspired us to conduct a comparative evaluation of alternative unikernel and container-based technologies, as well as implementations of application functions, to identify the particular performance trade-offs of each combination [97], from the view-point of basic edge cloud operations (i.e., resource manipulation, service operation and elasticity events). Along these lines, we argue that heterogeneity of virtual resources can be exploited from a novel edge cloud orchestration platform, as a means to improve its flexibility and adaptability to diverse network conditions and application requirements.

On the other hand, related papers introducing resource-allocation optimization models and mechanisms mostly focus on a single virtualization technology (e.g., regular virtual machines or containers) and data centers, while recent works re-visited the problem in the context of edge clouds.

In particular, these resource optimization approaches target (i) reducing CPU, RAM or space consumption, such as [111]; (ii) improving Quality of Service (e.g., in terms of service latency or response time), including [112], or (iii) achieving energy efficiency through reducing the number of active server nodes [113]. Other works improve resource efficiency through load balancing, achieving improved service performance, server or network resource utilization, as well as a scalable operation, as identified in survey paper [114].

Other related models or mechanisms focus on container-based data center deployments, including: (i) the resource allocation algorithm [115] minimizing the cost of application deployment with respect to specific Quality of Service requirements, while considering compute resource, network and energy consumption of each node; and (ii) the EnLoB algorithm [116] that minimizes the overall energy consumption and balances the load between active hosts.

Furthermore, recent resource-optimization proposals consider edge clouds. For example, [117] proposes an auction-based mechanism for resource allocation that guarantees the network latency in each task and maximizes the resource-allocation efficiency for both service and edge infrastructure providers. In [118], the authors experiment with two greedy algorithms that correspondingly aim at minimizing IoT data request delay and cost of service deployment, based on a relevant lightweight framework solution called FOGPLAN.

In comparison to these models, our cloud resource optimization framework exhibits the following novel characteristics: (i) it is the first one considering *Virtualization Technology Shifting*, our proposed edge cloud orchestration strategy that employs multiple virtualization solutions; (ii) it maps users to alternative builds of service nodes over distributed edge and core cloud infrastructures with heterogeneous hardware; and (iii) it implements particular performance goals for an increased number of users. Additionally, above models or mechanisms are commonly being evaluated through Monte-Carlo based simulations (e.g., adjusting their parameters by setting random values within specific ranges), e.g., in [115] or [118]. In our case, we configure the proposed model with realistic values extracted from our relevant extensive test-bed based evaluations, as documented in section 3.6 (i.e., paper [97]). However, we currently target a feasibility validation of our proposed cloud orchestration strategy, rather than time-efficiency of the model (e.g., like in [102]), which aspect deserves an independent study.

4.4 System Model

Here, we elaborate on our proposed system model, which is a functional description of a system implementing *Virtualization Technology Shifting*. Its main objective is to maintain service performance as close as possible to a particular performance goal, with a high cloud resource allocation efficiency.

As shown in Fig. 34, we assume an infrastructure provider operating multiple points of presence (PoPs) that are either core clouds, characterized by sufficient resource availability but usually located far from the majority of users, or edge clouds having limited resource availability, while being typically deployed close to users. Actually, each user participates in a cluster, illustrated as a pentagon, which is being assigned to a particular PoP, i.e., ideally to the closer edge cloud PoP, in networking distance terms. The PoPs may support diverse hardware, virtualization technologies or content server technologies, even at the same PoP, as indicated with distinct coloring at the bottom left of Fig. 34. Although multi-domain orchestration aspects are indeed important, here we assume either a single domain or an equivalent end-to-end resource control abstraction already in place, like in the cloud-network slicing system

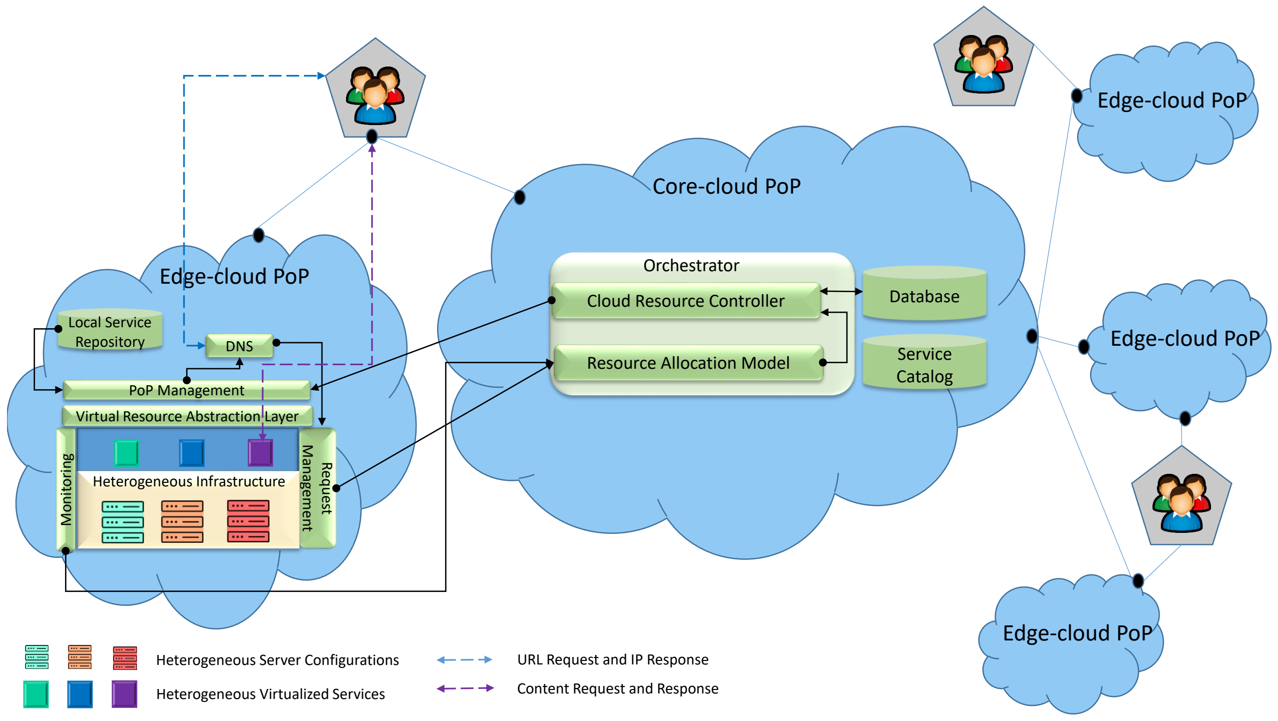

Edge-cloud PoP
Core-cloud PoP
Edge-cloud PoP
Edge-cloud PoP
Edge-cloud PoP
Local Service Repository
DNS
PoP Management
Virtual Resource Abstraction Layer
Monitoring
Heterogeneous Infrastructure
Request Management
Orchestrator
Cloud Resource Controller
Resource Allocation Model
Database
Service Catalog
Heterogeneous Server Configurations
Heterogeneous Virtualized Services
URL Request and IP Response
Content Request and Response

Figure 34: System Model

NECOS [95].

For simplicity, we assume an exemplary web-based service deployment over a given number of edge clouds and one core cloud, representing an operating content distribution network (CDN). In this context, different clusters of users reside at geographically dispersed areas and periodically download content. The content requests are directed to the most appropriate virtualized content server, in terms of satisfying a given performance goal (e.g, upper limit in content delivery time). Such allocation considers the available virtualization options and their performance capabilities that depend on the particular server characteristics, as well as the latest resource availability monitoring information.

We now briefly describe all involved system components of both Edge-cloud and Core-cloud PoPs, as shown in Fig. 34.

The Edge-cloud PoPs support the *Request Management*, *Monitoring* and *PoP Management* functions, as well as contain a *Virtual Resource Abstraction Layer*, a *DNS* and a *Local Service Repository*, all detailed below.

The *Request Management* function tracks all content requests over a fixed time interval (i.e., called status update period – SUP) and maintains historical information used from the function to identify the variations in client requests. For example, it may either (i) reactively employ a change-point analysis, like in our recent works [72], [57], or (ii) proactively utilize load prediction mechanisms (e.g., a matching *Seasonal Autoregressive Integrated Moving Average* model, such as in [97]). However, this paper is not focusing on this aspect, rather than assesses the benefits of adopting VTS, in terms of service performance and cloud resource allocation efficiency, under various client request demands and levels of cloud resource utilization. Similarly to *Request Management*, the *Monitoring* function measures periodically, i.e., for every SUP time-period, recent resource availability of compute and network resources in the corresponding PoP.

Furthermore, the *PoP Management* function is responsible for the control of virtual resources (e.g., manipulation or configuration) to match the service performance or cloud resource requirements. This process is realized through the *Virtual Resource Abstraction Layer* that employs virtualization-technology-specific APIs, supported for each one of the considered virtualization technology. The control of virtual resources may involve an interaction (i.e., downloading) with the *Local Service Repository*, which hosts alternative builds of virtualized services (i.e., in the form of container or unikernel images).

The *PoP Management* function is also responsible to confirm that services are up and running, i.e., it configures the local *DNS* with their IP addresses and a maximum number of clients to assign to each one of them. In the case of a service removal, the *DNS* notification precedes the removal, so no new clients are assigned and the old ones have time to complete their content retrieval. The services shutdown after all pending clients are finished. Consequently, the *DNS* is now able to balance the users to the appropriate virtualized services, i.e., in terms of quantities and types that satisfy the given performance goal. The *DNS* is also responsible to periodically communicate the number of content requests to the *Request Management* function. Actually, the SUP timer is maintained in the former, but triggers the operation of the latter with the communication of content requests' number, in a sequential fashion.

The Core-cloud PoP supports all main functions of Edge-cloud PoPs, acting as an edge cloud as well. However, we did not depict these functions in Fig. 34, for simplicity. The Core-cloud PoP also supports a centralized *Orchestrator*, a *Database* and a *Service Catalog*. The *Orchestrator* is comprised of two standalone functions: (i) the *Resource Allocation Model* and (ii) the *Cloud Resource Controller*.

The *Resource Allocation Model* initially verifies, based on the latest resource availability of every PoP, whether the estimated client requests can be fulfilled. The resource availability and client requests are received from the *Monitoring* and the *Request Management* functions of each PoP, respectively. The same component decides (i) whether to allocate new or release existing resources; (ii) the number and characteristics of required virtualized services (i.e., in terms of virtualization technology and physical server configuration to utilize) to match the resource-demands of the estimated client requests. More details on the optimization model can be found in Section 4.5.

The *Cloud Resource Controller*, in turn, is responsible to determine the quantity changes regarding the number of services and virtualization technology choices, based on the decision passed from the *Resource Allocation Model* and the current status of virtualized services. After the completion of the previous process, the *Cloud Resource Controller* is notifying the corresponding *PoP Management* functions about the modifications. Although here the *Cloud Resource Controller* employs the VTS strategy, it can ideally support alternative approaches, including horizontal and vertical elasticity, which can be selected on demand. A relevant initial investigation can be found in Section 4.8.

The *Database* is storing information related to the status of virtualized services (i.e., amount and virtualization option) in each PoP, based on the most recent decision of the *Cloud Resource Controller*. Finally, the *Service Catalog* enlists and hosts all available virtualized services. It provides them to the *Local Service Repositories*, in the case they are not cached locally.

In the section that follows, we elaborate on our resource allocation model, which is aligned to the above functional description.

4.5 Optimization model for resource allocation

The *Resource Allocation Model* introduced in this chapter implements the concept of *Virtualization Technology Shifting*. The model supports our argument that adopting alternative virtualization technologies enables resource allocation and service performance flexibility.

We formulate the model with the main objective to maintain a given performance goal and minimize resource consumption, while satisfying a maximum number of users' requests under given conditions, i.e., in terms of availability of resources and choices of virtualization technologies.

Symbol	Description
P	Set of points of presence (PoP)
C	Set of user clusters
R^c	Set of user requests in user cluster c
L	A set of all links between PoPs p and clusters c
S	Set of server configurations
V	Set of available virtualization options
$bc_{p,c}$	Bandwidth capacity of link (p, c)
$td_{r,p,c,s,v}$	Network throughput demand of request r over link (p, c), assigned to virtualization technology v that is hosted by server with configuration s
$RC_{r,s,v}$	Fraction of CPU resources from server with configuration s and virtualization technology v required from request r
$RM_{r,s,v}$	Fraction of Memory resources from server with configuration s and virtualization technology v required from request r
$RN_{r,p,c,s,v}$	Fraction of Bandwidth capacity of link (p, c), connecting cluster c with PoP p, required from request r assigned to server with configuration s that hosts virtualization technology v
$d_{p,v}$	Binary indicator variable reflecting whether PoP p supports virtualization technology v
$k_{p,s}$	Number of servers with specific configuration s which are deployed in PoP p
$T_{r,p,c,s,v}$	Average service performance of content request r over link (p, c), assigned to virtualization technology v that is hosted by server with configuration s
λ	Performance goal guarantee for average service performance

Table 9: Model Notations

The optimization model considers the topology illustrated in Fig. 34. We assume a directed graph $G = (N, L)$, where $N = P \cup C$. Table 9 provides the terminology, we employed in the paper (i.e., as a notation). Set P includes all points of presence p (i.e., regarding both edge and core clouds) and C all user clusters c. Each cluster c corresponds to a set R^c that includes all user content retrieval requests r. Moreover, set L contains the links between each PoP p with the user clusters c. We define the links as (p, c), whereas their respective bandwidth capacities are expressed as $bc_{p,c}$. For simplicity, we assume that all links are direct. Although intermediary hops and multiple paths with diverse network capabilities between the PoPs and user clusters impact on the overall end-to-end (E2E) delays, we consider this aspect as a future work.

Each PoP consists of node clusters with particular hardware configurations and such nodes can host diverse virtualization technologies. The set S represents all different server configurations s and the notation $k_{p,s}$ specifies the amount of physical server configuration types s that are allocated to PoP p. Moreover, V expresses all different supported virtualization technologies v. In order to determine whether a particular virtualization technology is supported from a PoP, we introduce the relevant indicator variable $d_{p,v}$, which definition follows,

$$d_{p,v} = \left\{ \begin{array}{lr} 1 & \text{, if virtualization technology } v \text{ is supported in PoP } p \\ 0 & \text{Otherwise} \end{array} \right\}.$$

At this point of investigation, we consider a particular performance metric, expressing the time passed from the initiation of a client request, until it receives the corresponding content, which we call Total Delivery Time (TDT). Although our model is general enough to express additional service performance metrics for 5G services (e.g., throughput, energy consumption, cost and latency), this aspect is considered as an open issue. We also assume that each virtualization option v has a relatively stable average TDT per request r. This can be maintained with a fixed number of clients and amount of physical resources assigned to each deployed virtual entity. Hence, we define the variable $T_{r,p,c,s,v}$, indicating the Total Delivery Time (TDT) of a particular virtualization option v, hosted by server with configuration s, serving a single request

r over link (p, c). Furthermore, λ is a specific threshold expressed as a number that reflects the average value of the chosen performance goal, i.e., targeted TDT.

Resource consumption of a content request r in a user cluster c is expressed as a fraction of the total resource amount of a server s. Along these lines, we introduce variables $RC_{r,s,v}$ and $RM_{r,s,v}$ indicating the percentage of CPU and Memory utilization requirements for a content request r assigned to a particular server s hosting a given virtualization technology v, respectively. We also define the network utilization demands of r in a similar way, i.e., as a variable $RN_{r,p,c,s,v}$. To express the latter as a percentage, it derives from the network throughput demand of content request r over link (p, c) assigned to a particular virtualization technology v and server configuration s (i.e., the $td_{r,p,c,s,v}$), divided by the total amount of bandwidth capacity $bc_{p,c}$ of the same link.

Finally, to indicate whether a specific user request r is allocated to PoP p and assigned to a server with configuration s hosting a virtualization technology v, we use the binary decision variable $x_{r,p,s,v} \in \{0,1\}$. Hence, our resource allocation objective function (3) along with the constraints are expressed as follows:

$$\min_{x_{r,p,s,v}} \sum_{c\in C}\sum_{r\in R^c}\sum_{v\in V}\sum_{p\in P}\sum_{s\in S}(RC_{r,s,v}+RM_{r,s,v}+RN_{r,p,c,s,v})x_{r,p,s,v} \quad (3)$$

s.t:

$$\sum_{c\in C}\sum_{r\in R^c}\sum_{v\in V} RC_{r,s,v}x_{r,p,s,v} \leq k_{p,s} \quad \forall p\in P, \quad \forall s \in S \tag{4}$$

$$\sum_{c\in C}\sum_{r\in R^c}\sum_{v\in V} RM_{r,s,v}x_{r,p,s,v} \leq k_{p,s} \quad \forall p\in P, \quad \forall s \in S \tag{5}$$

$$\sum_{r\in R^c}\sum_{v\in V} RN_{r,p,c,s,v}x_{r,p,s,v} \leq 1$$
$$\forall p \in P, \quad \forall c \in C, \quad \forall s \in S \tag{6}$$

$$\sum_{p\in P}\sum_{v\in V}\sum_{s\in S} x_{r,p,s,v}d_{p,v} = 1 \quad \forall r \in R^c, \quad \forall c \in C \tag{7}$$

$$\frac{\sum_{c\in C}\sum_{r\in R^c}\sum_{p\in P}\sum_{v\in V}\sum_{s\in S} T_{r,p,c,s,v}x_{r,p,s,v}}{\sum_{c\in C}\sum_{r\in R^c} x_{r,p,s,v}} \leq \lambda \tag{8}$$

$$x_{r,p,s,v} \in \{0,1\} \quad \forall r \in R^c, \quad \forall c \in C \tag{9}$$

Next, we discuss the constrains (4)-(9). The first three constraints, i.e., (4)-(6), ensure that the assignment of all content requests from all user clusters to particular virtualized servers cannot lead to consumed resources that exceed the available capacity of each point of presence and corresponding links. Given the constrains (4)-(5), we do not need a relevant constraint for server configurations s, since whenever hardware configurations are not supported from a PoP p, the particular $k_{p,s}$ variables take 0 values. Constraint (7) implies that all content requests r are being assigned to single virtualization options, which are available to the particularly chosen PoPs. Furthermore, constraint (8) ensures that average TDT performance of content requests r, assigned to particular virtualization technologies v, does not exceed a specific λ threshold. Finally, condition (9) expresses the binary domain for the decision variable $x_{r,p,s,v}$.

4.6 Methodology & Empirical Complexity Analysis

Here, we provide our methodological approach, including elaborating on (i) the goals of our simulation analysis; (ii) the considered instantiation of proposed system and model, i.e., its specific assumptions and parameters; and (iii) details on the simulation process. Finally, we provide an empirical analysis regarding the complexity of the optimization model.

Our simulation results target at supporting key findings in this chapter, such as on (i) whether virtualized resource heterogeneity can be exploited, i.e., through the *Virtualization Technology Shifting* strategy, to improve resource allocation efficiency and tackle resource shortage situations; and (ii) how well the proposed system and model can maximize the number of users enjoying a service meeting a particular performance goal, in terms of efficiently assigning users to appropriate virtualization technologies and PoPs, with respect to the levels of utilized network and cloud resources in heterogeneous hardware configurations.

We now detail the parameters of the considered system and model with respect to the assumed topology, physical server hardware, supported virtualization technologies, and model parameters, regarding the resource-efficiency and performance of alternative virtualization technologies and server configurations.

In the simulation results that follow next (i.e., in Section 4.7), we assume a topology with three PoPs, i.e., two edge clouds and one core. We assume that the core cloud serves multiple services providers across edge infrastructures, therefore, only a slice of the maximum bandwidth capacity is allocated in each service provider. Such network bandwidth capacity can be significantly reduced due to the bottlenecks across particular hops in the network path from the core cloud to the client. Thus, we assume that the core cloud can be accessed over a path with a lower capacity compared to the links towards to the edge clouds. Furthermore, one cluster of users resides in between the edge clouds, like the bottom right user cluster in Fig. 34.

In our simulations, we consider four alternative server configurations, inspired by real open test-bed deployments: (i) the Mpc node of our own SWN test-bed [56]; (ii) the Zotac nodes of w-iLab.2 test-bed [9]; and (iii) d710 as well as d430 of central Emulab test-bed [10]. In order to evaluate the impact of server hardware heterogeneity, a PoP may support more than one of above options.

Additionally, each PoP server is able to host alternative virtualization technologies.

[9]https://doc.ilabt.imec.be/ilabt/wilab/hardware.html
[10]https://wiki.emulab.net/wiki/UtahHardware

Here, we consider Docker containers [31] as well as the ClickOS [34] and RumpKernel unikernel flavors [35]. Each virtualized service consists of a web-server with an embedded 5MB content, with the former being resource-limited to 256MB of RAM and able to handle up-to 15 simultaneous clients, i.e., exhibiting a relatively stable resource-allocation efficiency and service performance.

Since our prior section 3.6 included relevant experimental results [97], i.e., evaluated the same assumed service with Docker, ClickOS and RumpKernel in the SWN test-bed, we exploit them to define realistic parameters in our resource allocation model. In Tables 10 and 11, we enlist the percentage of CPU and Memory utilization required for a content request assigned to a server with a given configuration and virtualization technology, respectively. These values reflect to the equivalent $RC_{r,s,v}$ and $RM_{r,s,v}$ model parameters and do not depend on the choice of p.

Regarding the Table 10, we assume that Zotac, d710 and d430, require 75%, 25%, and 10% less CPU resources compared to Mpc to serve the content, due to equivalent differences in CPU power. In other words, the CPU consumption values of Mpc have been extracted from the corresponding results in section 3.6 and the table is completed for the other servers based on the above analogies. We also assume that each content request requires a fixed amount of memory, i.e., the allocated memory to the virtualized service divided by the maximum number of requests it can handle. Consequently, each content request requires around 17MBs of memory, for all virtualization technologies. Table 11 values have been calculated by dividing this value with the total memory amount of each corresponding server build.

For simplicity, we assume that alternative server configurations or network connectivity do not impact on the achieved service performance of a particular virtualization technology for content delivery, in terms of throughput or Total Delivery Time. Consequently, we omit the relevant parameters from td, RN, and T model parameters. In the considered setting, our results revealed a higher dependence on the choice of virtualized service rather than hardware configuration, given the availability of resources. Alternatively, we would require additional time-consuming experiments with various server and network configurations, without making a significant difference in the main

Table 10: CPU resource requirements of a content request ($RC_{r,s,v}$)

s \ v	Docker	ClickOS	RumpKernel
Mpc	3.24%	1.27%	1.56%
Zotac	2.43%	0.95%	1.16%
d710	0.81%	0.31%	0.39%
d430	0.32%	0.12%	0.15%

Table 11: Memory resource requirements of a content request ($RM_{r,s,v}$)

v \ s	Mpc	Zotac	d710	d430
for all v, v ∈V	0.416%	0.416%	0.138%	0.026%

arguments in the paper. However, this aspect deserves a further investigation.

Along these lines, Table 12 enlists the network throughput demands for one content request in Mbps for each one of the available virtualized server options, i.e., $td_{r,v}$. The $RN_{r,p,c,v}$ values are being calculated based on the bandwidth configuration of the PoPs. Furthermore, Table 13 presents the $T_{r,v}$ reflecting the content delivery performance, in terms of Total Delivery Time (i.e., in sec), for one user request in conjunction to each one of the considered virtualized services.

As a bottom line, Docker achieves the best service performance, ClickOS the lower CPU consumption and RumpKernel is the most network-friendly option, among the considered virtualization options.

We now provide the basic details of our simulation process. We implemented the proposed model in Python and utilized the Gurobi solver. The latter can measure the execution time required to solve the model, based on the particular inputs. All our runs have a deterministic nature, so no statistical evaluation is needed. We carried out our simulations on a laptop with 6 CPU cores at 2.2 GHz and 16 GB of RAM, using

Table 12: Network throughput demands of a content request assigned to a particular virtualized service ($td_{r,v}$)

s \ v	Docker	ClickOS	RumpKernel
for all s, s ∈S	2.76	1.81	1.59

Table 13: Total Delivery Time (sec) with alternative virtualization options ($T_{r,v}$)

$s \setminus v$	Docker	ClickOS	RumpKernel
for all s, $s \in$S	2.7	3.89	4.18

the Linux-based Ubuntu 18.04 (LTS) operating system.

4.6.1 Empirical Complexity Analysis

At this point, we investigate the complexity of the model based on an empirical analysis. In our analysis, we (i) consider as input the predefined parameters enlisted in Tables 10, 11, 12, and 13; (ii) assume the topology configuration of Table 14 (e.g., nine edge clouds and one core); (iii) enable virtualization technology shifting; and (iv) consider a best-effort service. We note that 10 PoPs is a reasonably-sized edge cloud deployment, e.g., Amazon CloudFrond maintains 14 edge PoPs in UK[11].

Table 14: The setup of empirical complexity analysis

Parameter	Values
P	$p \in \{p_1,..,p_{10}\}$ (i.e, 10 clouds - 9 edge and one core)
S	$s \in \{Mpc, Zotac, d710, d430\}$
V	$\in \{Docker, ClickOS, RumpKernel\}$
C	$c \in \{c_1\}$
$bc_{p,c}$	10Gbps for all p, $p \in$P, and for all c, $c \in$C
$k_{p,s}$	10 for all p, $p \in$P, and for all s, $s \in$S

In our resource allocation model, the number of client requests is the dominant complexity factor. For the same reason, Fig. 35 illustrates the execution time of the solver to find the optimal solutions with respect to increasing user load. According to these results, a quadratic time complexity characterizes the model, in the worst case.

To assess the results of Fig. 35 both quantitatively and qualitatively, we consider an example use-case. In 2020, a well-known food delivery app, namely "Just Eat", received 294 millions of food orders in the UK[12]. This number is equivalent to an average of 140 orders per 15 sec, considering the default execution time period of the Kubernetes auto-scale component (i.e., HPA). According to Fig. 35, the model requires

[11]https://aws.amazon.com/cloudfront/features
[12]https://www.businessofapps.com/data/just-eat-statistics/

around 0.03 sec to find a solution with 250 client requests, which is an acceptable execution time, associated with a fixed time interval of 15 sec.

Consequently, given the abundant resources typically characterizing a core cloud that may host the model and above empirical analysis, it is safe to assume that our model exhibits an acceptable complexity for medium-sized deployments (e.g., of 10-20 PoPs and 2000-4000 client requests per 15 sec). In this case, there is no reason to trade the complexity level of the model for a potential sub-optimal solution of a heuristic, i.e., impacting on cloud resource allocation efficiency. Since this work targets validating the proposed *VTS* concept rather than time efficiency, introducing heuristics that are more suitable for larger-scale deployments is in our future plans.

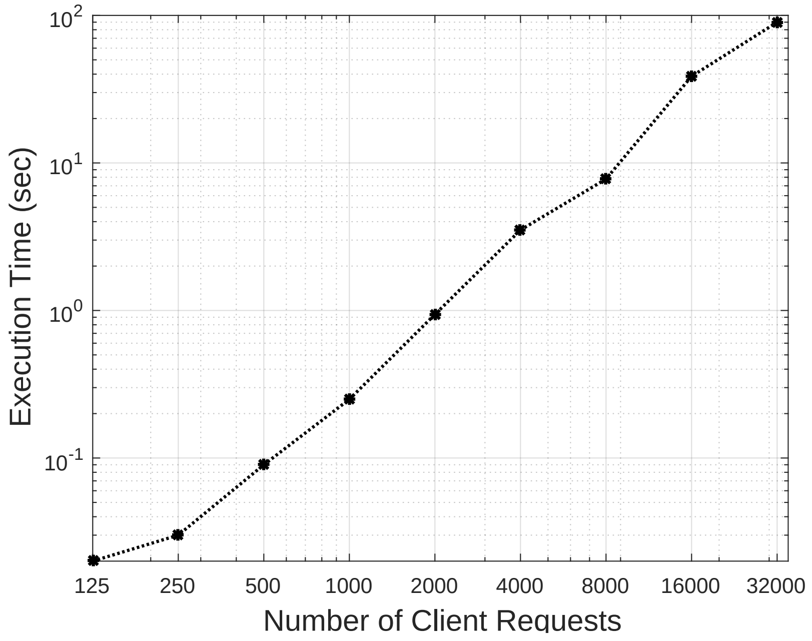


Figure 35: Execution Time of increasing request demands

In the section that follows, we provide and elaborate on our simulation results.

4.7 Evaluation Simulation Results

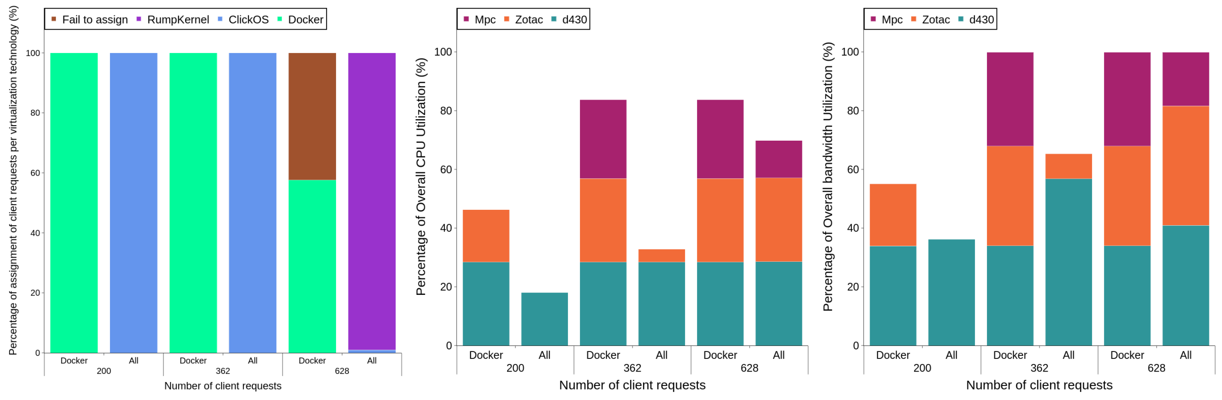
We grouped our simulation results into two scenarios. The first scenario, called *"Impact of Adopting VTS"*, motivates the fresh view-point of this chapter that an edge cloud orchestration system can significantly benefit from integrating VTS. The second scenario, entitled *"Assessing VTS"*, evaluates *Virtualization Technology Shifting*, as defined in the proposed system and optimization models, with various network conditions and server hardware configurations.

4.7.1 Scenario 1: Impact of Adopting VTS

In the first scenario, we provide simulation results in support of our main idea that VTS brings benefits, in terms of (i) improving server and network resource utilization; and (ii) increasing the number of users that can enjoy a service with a given resource availability, including meeting a particular service performance goal. In practical terms, we compare the outcome of applying VTS against a typical edge cloud orchestration system implementing horizontal elasticity. In the former case, VTS utilizes multiple virtualization technologies (i.e., marked as all), where in the latter case, only docker containers are utilized (i.e., marked as docker).

We consider two types of services: (i) a best-effort service that is able to serve users when particular physical resources are available, only; (ii) and a performance-sensitive service that delivers content, when a given service performance goal is satisfied. The considered performance goal of the second service is a Total Delivery Time (TDT) below 3.5 sec (i.e., $\lambda = 3.5$), i.e., being delay-sensitive. We employ a topology with a single edge cloud PoP serving all client requests via a 1Gbps link. The PoP hosts 6 servers with Mpc configuration, 3 with Zotac and one with d710. This allows us to study also the impact of server hardware heterogeneity. In the case of best-effort service, we relaxed the constraint of Eq. (8).

Fig. 36 and 37 correspond to the best-effort and delay-sensitive service, respectively, illustrating: (i) the virtualization options utilized per group of content requests, i.e., in percentages, and (ii) the types of physical servers utilized as well as the respective overall CPU and network utilization. We omitted the memory consumption results, due to their expectable behavior, but we comment on them, when needed. In all figures, we consider three numbers of requests. The first one is indicative (i.e., 200 requests) and the other two express the maximum feasible amount of client requests (i.e., without a significant degradation in the service performance, as defined in Table 13), when the VTS strategy is not enabled or enabled (i.e., docker or all), respectively.



(a) Assignment of client requests (b) Overall CPU utilization (c) Overall network utilization

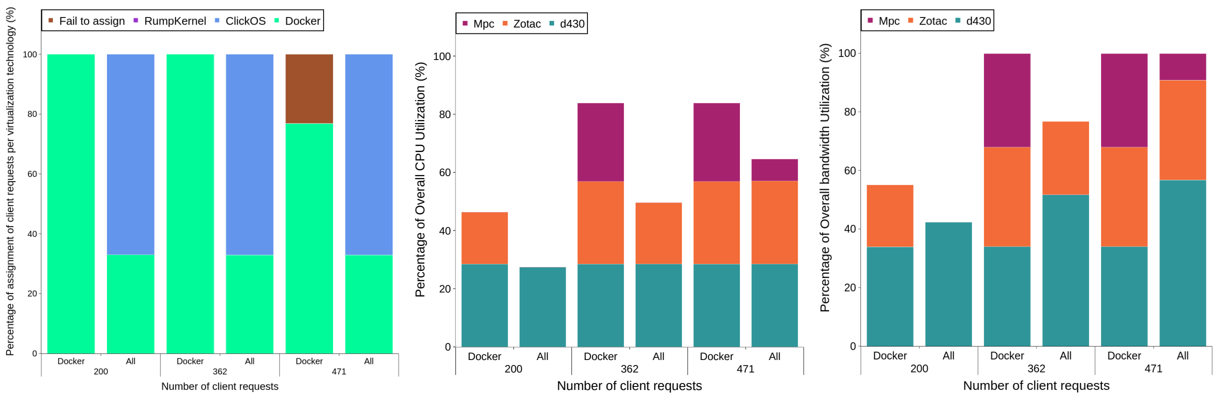
Figure 36: Impact of Virtualization Technology Shifting for a best-effort service

We start with the first part of our results that consider the best-effort service. As shown in Fig. 36, 200 client requests are not exhausting the available CPU and network resources. Nevertheless, VTS strategy assigns all 200 content requests to the ClickOS unikernel, which, compared to the traditional employment of docker, leads to an improved overall CPU and network resource consumption by 28.1% and 19%, i.e., Fig. 36(b) and 36(c), respectively.

In Fig. 36(a), we notice that docker option is able to accommodate a maximum number of 362 clients due to a bottleneck in network resources (i.e., 36(c)). However, a triggering of a VTS strategy from an intelligent orchestrator would lead to serving content up-to 628 clients (i.e., a 74% improvement), with the given resource availability. In the latter case, we notice a shift from ClickOS to RumpKernel option, which appears around the 450 client requests, because there is a need to switch from an overall resource-efficient option to one with the lower network throughput demands, due to the gradual appearance of the network bottleneck. For both 362 and 628 client requests, VTS strategy consumes significantly lower CPU and network resources, e.g., up-to 50.9% CPU and 34.4% network resources (i.e., Fig. 36(b) and 36(c)).

The two options (i.e., all or docker) consume equivalent memory resources for the 362 clients, while VTS strategy consumes 8% more memory for the 628 users, since it handles more content requests. Furthermore, VTS utilizes d430 only, both d430 and

Zotac, and all server configurations for 200, 362 and 450 clients, respectively. For the docker option, more servers with diverse hardware configurations may be used, i.e., both d430 and Zotac for 200 and all server configurations for 362 clients. Finally, there is a trade-off to consider regarding the application of VTS strategy, since it may affect the service performance. In this particular case, the Total Delivery Time (TDT) is being gradually increased from 2.7 to 4.2 sec. However, the proposed model allows the definition of particular service performance goals, in support of services with less relaxed performance requirements. A relevant investigation follows next.



(a) Assignment of client requests (b) Overall CPU utilization (c) Overall network utilization

Figure 37: Impact of Virtualization Technology Shifting for a service under a performance goal (i.e., $\lambda = 3.5$)

We now assume a service with a particular performance goal (i.e., with $\lambda = 3.5$). In Fig. 37, we notice an equivalent behavior for the docker option as the best-effort service simulations. This can be justified by the decent performance and resource-allocation capabilities of Docker, i.e., it nevertheless addresses the service performance goal with under-utilized resources. However, the VTS strategy, for both 362 and 471 client requests, assigns the 67% of clients to ClickOS and the 33% of them to Docker, i.e., to serve services that meet the performance goal, while improving CPU and network utilization up-to 34.2% and 23.1% (i.e., Fig. 37(b) and 37(c)), respectively. However, comparing Fig. 37(b) and 37(c) with the equivalent Fig. 36(b), 36(c) of the best-effort service reveals that mixed virtualization option requires 9.3% to 16.7% and 6.2% to 11.3% more CPU and network resources to achieve the targeted service performance with the equivalent client requests, respectively. We also observe in Fig. 37(a) that VTS strategy leads to 30% more clients enjoying the requested service quality (i.e., 471 compared to 362 of Docker).

The memory consumption for 362 clients is equivalent between Docker and VTS strategy, while the latter consumes 4% more memory in the case of 471 clients. Additionally, both docker and all virtualization options utilized the same server configurations with the best-effort service. Finally, now the TDT for VTS strategy takes a value which is equal to the targeted performance goal (i.e., 3.5 sec).

According to the above results, the incorporation of VTS increases significantly the load magnitude the system handles, before allocating additional physical resources. For example, for both considered best-effort and delay-sensitive services, a hybrid horizontal / vertical elasticity strategy would trigger vertical elasticity for 362 client requests, which is the limit of applying a horizontal elasticity process. However, the employment of VTS after the exhaustion of resources caused by horizontal elasticity allows the system to satisfy up-to 628 clients, without the addition of new resources. Consequently, it makes sense for horizontal elasticity, VTS, and vertical elasticity processes to be triggered in a sequential fashion. Section 4.8 gives a relevant example, where alternative edge cloud orchestration strategies can jointly be supported by an intelligent orchestrator.

As a bottom line, applying the VTS strategy in resource-shortage conditions leads to the accommodation of both best-effort and delay-sensitive services with a larger number of users (i.e., up-to 74%) and improved server and network resource allocation (i.e., up-to 50.9% and 34.4%, respectively), compared to a traditional deployment utilizing Docker containers only. This demonstrates the capabilities of VTS, which are stressed next under different conditions.

4.7.2 Scenario 2: Assessing VTS

Here, we evaluate the behavior of proposed VTS resource optimization framework (i.e., system and optimization model) under different conditions, in terms of service performance goal, available server configurations and network capacity. Our goal is to demonstrate its capability to organize available heterogeneous virtual resources and assign users to available PoPs and virtualized services, in a way that satisfies a particular service performance goal. We consider the three-PoP topology and user cluster described in section 4.6. We also set the bandwidth capacity as defined in Table 15. In practice, the edge cloud PoPs have a ten times higher bandwidth capacity compared to the core cloud, when considering two distinct levels of bandwidth capacity (i.e., moderate and high). We range the service performance goal according to $\lambda \in \{5,4,3\}$, while assuming homogeneous hardware resources among the 3 PoPs with alternative server configurations, i.e., $s \in \{Mpc, Zotac, d710, d430\}$, indicating different sets of resources. We rank the server types on how powerful they are, in terms of CPU and Memory resource capacity. All servers support RumpKernel, ClickOS and Docker virtualization technologies. For each server configuration, we assume both an indicative number of clients (i.e., 50) and the maximum number of clients the PoPs can support, depending on the availability of resources.

For all of the above diverse conditions, i.e., in terms of service performance goal, server configuration, network capacity and number of clients, we illustrate results on (i) the user assignment to virtualization technologies; (ii) the average CPU utilization per PoP; and (iii) the network throughput of each PoP.

Table 15: Bandwidth capacity ($bc_{p,c}$) of link (p, c) in Mbps, i.e., between Edge Cloud (EC) or Core Cloud (CC) PoPs and user cluster c

Link / Bandwidth	EC to c	CC to c
Moderate bandwidth case	100	10
High bandwidth case	500	50

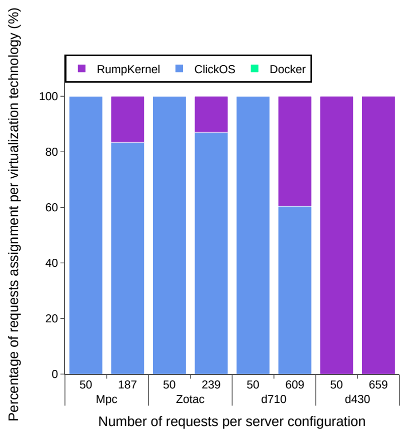
We first consider the case with high bandwidth availability and investigate the impact of different service performance goals on the behavior of proposed system. In Fig. 38, we observe that service performance goal threshold λ impacts significantly on the chosen types and quantities of utilized virtualization options. For the majority of server configurations, ClickOS and Docker are preferred for most client requests in the cases of $\lambda = \{5,4\}$ and 3, respectively. An exception is the most powerful d430 server with the conservative $\lambda = 5$, where the system assigns all user requests to the

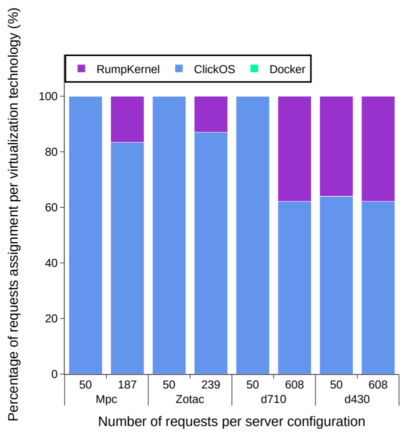
RumpKernel virtualization technology.

We also see that decreasing λ leads to reduced numbers of clients that can be accommodated in the cases of high-end servers and λ changing from 5 to 4 (i.e, slightly for d710 server and around 8% for d430), however there is a rapid reduction of feasible client requests for all server configurations and λ switching from 4 to the lowest considered value 3, i.e., around 50%, 51%, 48%, and 31.5% for Mpc, zotac, d710, and d430, respectively. This occurred due to the wide involvement of Docker, i.e., ranging from 75 to 76% of clients, that achieves robust service performance at the cost of more resources, i.e., reducing the number of clients enjoying a given service quality.

In Fig. 39, we see that overall CPU consumption per PoP decreases with the utilization of most powerful servers, in the case of 50 content requests. The 5 and 4 values of λ lead to the consumption of equivalent CPU resources and the satisfaction of a similar or the same number of clients, besides the run with d430 server configuration. In the latter case, the reduction in the number of clients is associated with the release of some CPU resources. Applying a more strict λ value of 3 leads to the utilization of an additional PoP for the 50 clients, which can be justified by the increased resource demands of the utilized Docker containers. Regarding the maximum achieved number of clients, there is clearly a bottleneck in the CPU consumption for all server configurations in the two edge clouds, besides the most powerful one (i.e., d430). Consequently, CPU availability may set an upper limit on the number of admitted clients. This also underlines the tendency of the system to select the most CPU-efficient ClickOS option, when such a bottleneck appears.

Furthermore, Fig. 40 indicates a bottleneck in the network connectivity of core cloud, for all cases that maximize the number of admitted clients. For the runs with d710 and d430 server configurations, the edge cloud links are fully utilized as well. In these cases, the system selects the most network-efficient RumpKernel option, whenever there is availability of resources. The d430 servers have enough CPU to serve many clients through RumpKernel servers, up to a limit that all links to the three clouds are saturated.


(a) λ=5


(b) λ=4

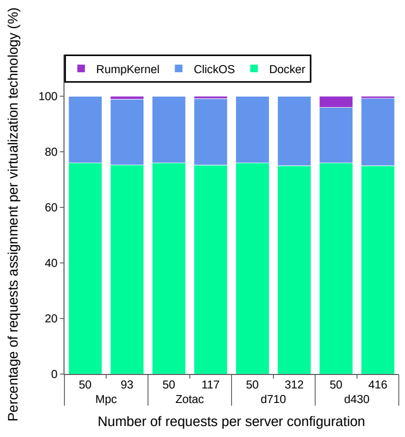
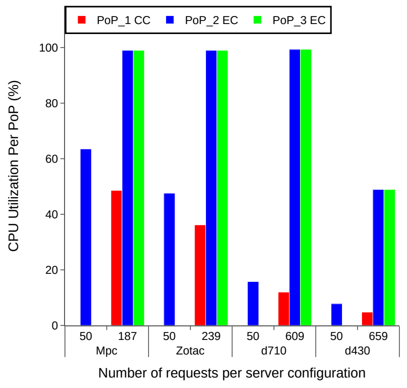
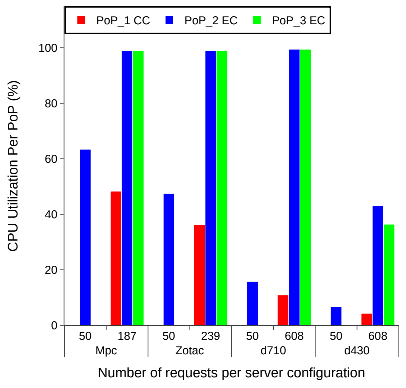

(c) λ=3

Figure 38: Assignment of client requests (high bandwidth capacity)


(a) λ=5


(b) λ=4

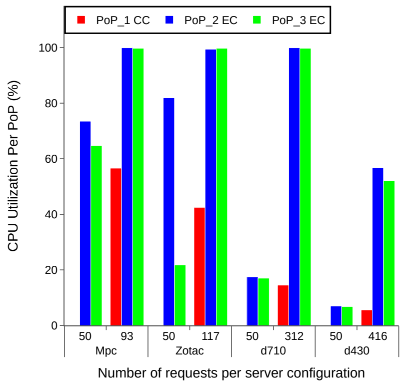
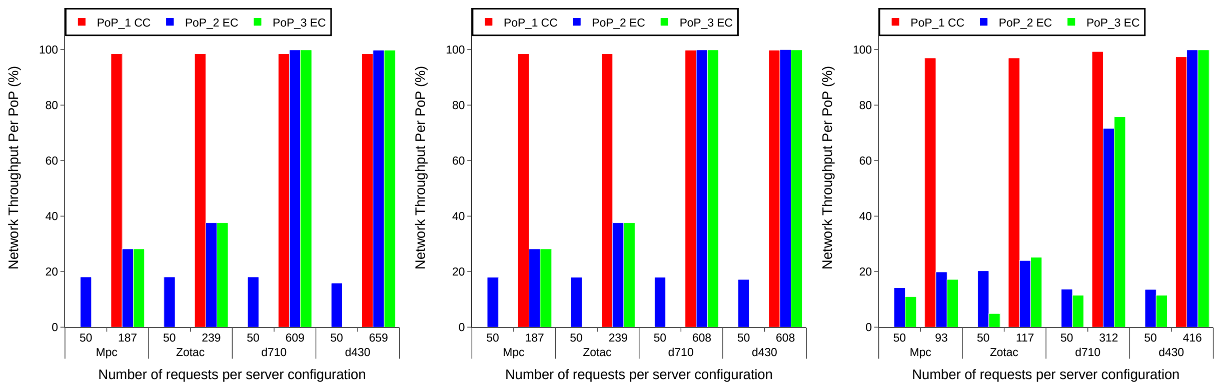

(c) λ=3

Figure 39: Overall CPU utilization per PoP (high bandwidth capacity)

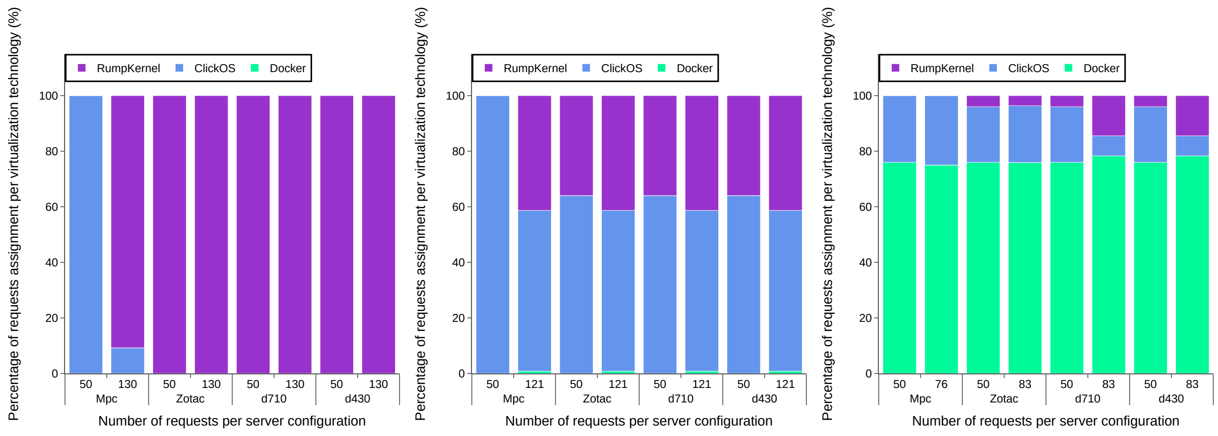
Finally, we now discuss the impact of a moderate PoPs' bandwidth capability on the system behavior. As we see in Fig. 42 and 43, in contrast to Fig. 39 and 40, the network bottleneck intensifies and CPU allocation is not having a critical role, besides the low-end Mpc server configuration. In the latter experimental run, CPU exhausts its resources, especially for $\lambda = 3$.

As we see in Fig. 41, service performance goal level leads for most clients to particular choices in terms of virtualization technology, i.e., RumpKernel, ClickOS and Docker for 5, 4, 3 values of λ, respectively. Switching λ from 5 to 4, leads to the



(a) λ=5 (b) λ=4 (c) λ=3

Figure 40: Overall network throughput per PoP (high bandwidth capacity)

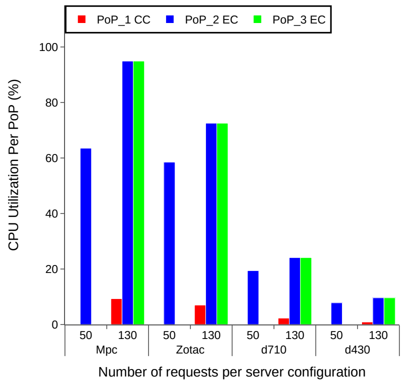


(a) λ=5 (b) λ=4 (c) λ=3

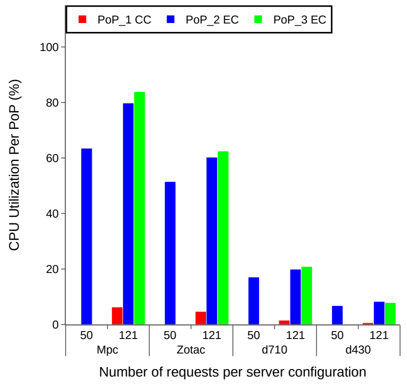
Figure 41: Assignment of client requests (moderate bandwidth capacity)

accommodation of 7% less users, and from 4 to 3, they are reduced by 32-38%.

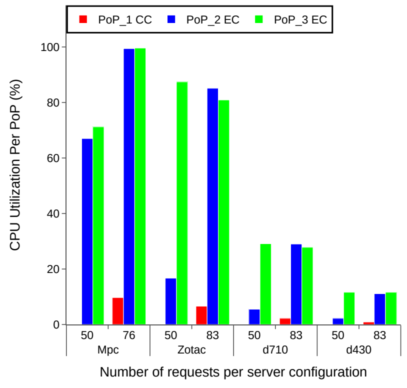
For all server configurations besides Mpc, the selection of each particular virtualization technology is mainly attributed to its service performance in contrast to its network efficiency. This is also validated by the almost zero impact of different server configurations on the maximum numbers of users that can be accommodated for alternative λ. Furthermore, the CPU resources of core clouds are minorly consumed due to their low link capacity (i.e., Fig. 42). In the case of 50 clients, only one edge cloud is required, besides the most challenging $\lambda = 3$, where the second edge cloud is utilized as well.



(a) λ=5

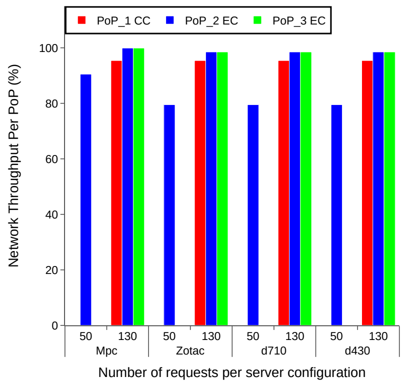


(b) λ=4

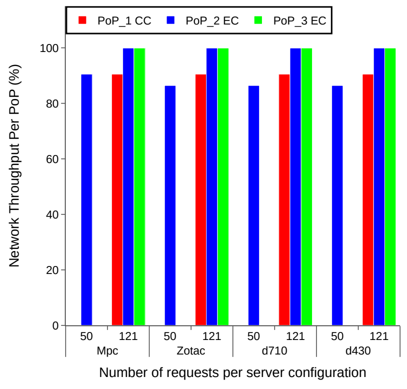


(c) λ=3

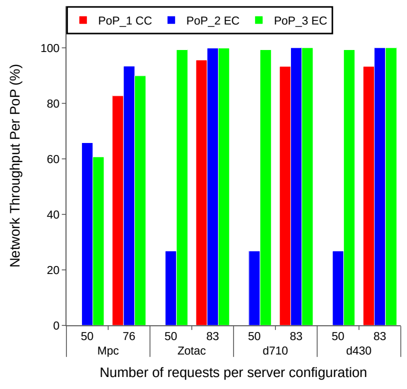
Figure 42: Overall CPU utilization per PoP (moderate bandwidth capacity)



(a) λ=5



(b) λ=4



(c) λ=3

Figure 43: Overall network throughput per PoP (moderate bandwidth capacity)

The above results confirmed the following aspects of our proposal: (i) it organizes heterogeneous virtual resources and client requests towards satisfying particular performance goals; (ii) it is able to tune the involved performance trade-offs towards maximizing the numbers of accommodated content requests and minimizing the utilization of server and network resources; and (iii) the potential performance bottlenecks play a critical role in the choice of virtualization technology and the balancing of users to multiple PoPs.

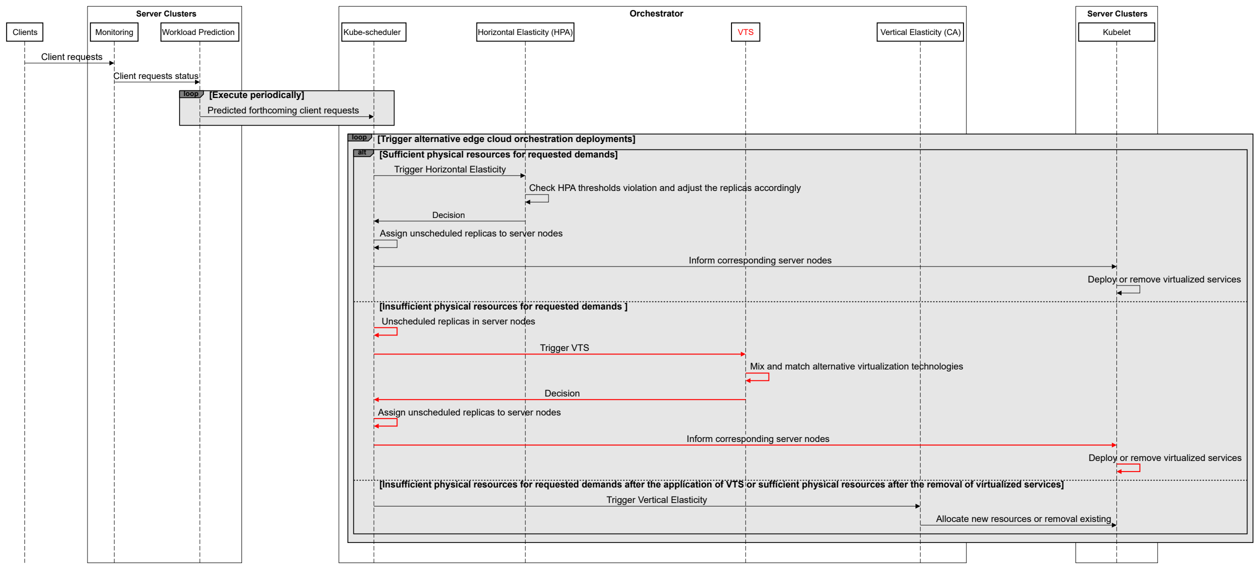
4.8 Example Integration of VTS

Here, we provide our initial considerations, being complemented with an example workflow, on the integration of VTS with a typical Kubernetes-based edge cloud orchestration environment. The workflow is inspired by the results' insights gained from our first experimentation scenario (i.e., Subsection 4.7.1), highlighting that incorporating VTS in a hybrid horizontal/vertical autoscaling process leads to a significantly increased user capacity of the system.

In this context, we assume a Kubernetes-based environment implementing horizontal and vertical elasticity in a sequential manner, i.e., allocates new virtual resources with the increase of user load up-to physical resource exhaustion, which is, in-turn, triggering the inclusion of additional physical resources. Specifically, it employs a centralized orchestrator component (i.e., *kube-scheduler*), which realizes the most appropriate cloud resource control strategy to a given predicted user load (i.e., based on *Monitoring* and *Workload Prediction* components). The selected strategy manipulates virtual and physical resources through distributed subsystems (i.e., *Kubelets*), hosted from each physical node cluster. The supported cloud resource control strategies are being handled by dedicated control entities, i.e, *HPA* and *CA* for horizontal and vertical elasticity, respectively.

In sequence diagram of Fig. 44, we portray all above components and processes. We also mark as red the extension of proposed system that incorporates the proposed VTS strategy. Specifically, we include a new centralized control entity (i.e., VTS) with its corresponding resource control processes, i.e., being sequentially executed in between horizontal and vertical elasticity actions.

Although the above example requires a deeper investigation and analysis, the considered components have analogies with the proposed system model, e.g., *Kube-scheduler* and *kubelet* match the functionalities of *PoP Management* and *Cloud Resource Controller*, respectively. However, we consider the implementation and experimentation of a cloud orchestration system integrating VTS, as an open issue to handle in a follow-up work.



Server Clusters
Clients
Monitoring
Workload Prediction
Orchestrator
Kube-scheduler
Horizontal Elasticity (HPA)
VTS
Vertical Elasticity (CA)
Server Clusters
Kubelet
Client requests
Client requests status
loop
[Execute periodically]
Predicted forthcoming client requests
loop
[Trigger alternative edge cloud orchestration deployments]
alt
[Sufficient physical resources for requested demands]
Trigger Horizontal Elasticity
Check HPA thresholds violation and adjust the replicas accordingly
Decision
Assign unscheduled replicas to server nodes
Inform corresponding server nodes
Deploy or remove virtualized services
[Insufficient physical resources for requested demands]
Unscheduled replicas in server nodes
Trigger VTS
Mix and match alternative virtualization technologies
Decision
Assign unscheduled replicas to server nodes
Inform corresponding server nodes
Deploy or remove virtualized services
[Insufficient physical resources for requested demands after the application of VTS or sufficient physical resources after the removal of virtualized services]
Trigger Vertical Elasticity
Allocate new resources or removal existing

Figure 44: Example workflow of a Kubernetes-based system integrating VTS (VTS extensions in red)

4.9 Conclusions

This chapter introduces *Virtualization Technology Shifting*, a new cloud orchestration strategy that exploits virtualization heterogeneity as a means to improve cloud resource allocation and service performance. Our approach is backed with a relevant system model and a bespoke resource allocation optimization model that considers the estimated performance trade-offs of a set of technologies (e.g., virtualization approaches and particular hardware) in a distributed edge and core cloud environment, as well as aims at a particular performance guarantee. We carried out extensive simulations using model parameters extracted from real experiments that: (i) validated the flexibility, resource utilization and performance impact of the VTS approach against a typical edge cloud orchestration strategy (i.e., horizontal elasticity); and (ii) assessed the proposed VTS resource optimization framework (i.e., system and optimization model) under different conditions, in terms of targeted performance goal, server configuration and network capacity. According to our results, integrating VTS leads to a 74% higher number of users enjoying a service with a given performance goal, as well as exhibits lower server and network resource utilization, i.e., up to 50.9% and 34.4%, respectively.

5 Conclusions and Further Work Discussion

5.1 Summary and Conclusions

This chapter concludes the dissertation, presenting and summarizing its contributions and describing relevant future research directions.

To begin with, in the thesis we proposed, designed, elaborated, and evaluated two edge cloud orchestration facilities, namely UNIC and 5G-CDN. The UNIC platform realizes flexible content and measurement distribution over heterogeneous virtual and physical resources, utilizes tiny Unikernel-based VMs (i.e., Unikernels). UNIC realizes intelligent and modular orchestration for both cloud and network aspects. We carry out proof-of-concept results to validate: (i) the efficiency and elasticity capabilities (i.e., content popularity detection mechanisms driven the VM placement) of our approach in terms of responsiveness and resource utilization; and (ii) the benefits regarding the improvement in communication connectivity of mobile users through employ an adaptable multi-homing solution with lightweight virtualization approaches. Furthermore, 5G-CDN is an experimentation facility that focuses on the next-generation Content Delivery Networks (CDNs), a paradigm for hosting and launching ME services. We designed and build on top of existing novel test-bed federations, such as the Fed4FIRE+, to enable large-scale, multi-domain experimentation, involving: (i) large-scale end-to-end network slicing over multiple infrastructure providers utilizing heterogeneous hardware and virtualization resources; (ii) dynamic resource discovery and allocation residing at both federated open-access and local test-beds; and (iii) experimentation with modular media service orchestration mechanisms, e.g., on content caching and service elasticity. We provided proof-of-concept results, validating the above aspects involving different test-beds, where heterogeneous physical and virtual resources coexist. Finally, our overall results showed that Unikernel virtualization flavors have rapid responsiveness, improve resource utilization as well as are suitable for resource constraint environments.

Additionally, we studied the performance dynamics of alternative edge cloud technologies, including different unikernel flavors, container builds, and implementations of exemplary web services. Our experiments investigated both service and infrastructure viewpoints, as well as all the basic edge cloud processes, including on service operation, cloud resource or service elasticity, dynamic resource allocation and removal, which revealed that each involved option is characterized by particular performance trade-offs benefits in different circumstances. In a nutshell: (i) unikernels represent an asset for edge clouds, bringing rapid manipulation capabilities, while seem to exhibit an expectable behavior in terms of server resource utilization, i.e., making prediction mechanisms less challenging, while containers provide a robust service operation performance; (ii) deployment frameworks utilizing containers at the core and unikernels at the edge cloud balances well the trade-offs between content delivery times and server resource utilization; and (iii) alternative application function options produce different outcomes, e.g., even a minimalistic web server can achieve performance advantages, in specific conditions. Finally, we provided key design guidelines insights for a conceptual edge cloud orchestration platform that gained from our extensive comparative evaluation based on a novel edge cloud experimentation environment.

Next, we introduced and elaborated a fresh cloud orchestration strategy, called Virtualization Technology Shifting (VTS). Our approach is backed by a relevant system model and a cloud resource optimization framework in order to assess the benefits as well as the feasibility of our solution. We carried out extensive simulations experiments: (i) validated the impact of our VTS approach against a typical edge cloud orchestration strategy (i.e., horizontal elasticity) in terms of flexibility, server and network resource utilization and assignment of users; and (ii) assessed the capabilities of the proposed system model, while enabling our cloud orchestration strategy under different conditions, in terms of a service performance goal, server configuration and network capacity. According to our results, such strategy augments the number of users that can be served up to 74% and improves the server and network resource utilization up to 50.9% and 34.4%, respectively.

To sum up, this thesis highlights, through leveraging the proposed edge cloud orchestration facilities, that the employment of lightweight virtualization technologies (i.e., containers and multiple unikernel flavors) can become a powerful asset of edge cloud deployments by improving resource allocation efficiency and tackling the dynamic changes with their fast deployment capabilities. However, there is no single best virtualization solution for the edge cloud. Thus, mobilizing different lightweight virtualization technologies in a joint scheme with edge cloud orchestration deployments, presents an attractive solution, since it enriches the available resource control options for edge clouds, i.e., mixing and matching those alternative technologies with challenging cloud resources optimization requirements. Therefore, understanding how and when to utilize the most appropriate virtualization technology for edge cloud orchestration deployments should be taken into account, according to resource constraints, particular requirements of network or application services, as well as the dynamic changes of user demands and network conditions. These technologies can become a vital capability of the next-generation edge cloud orchestration deployments.

5.2 Further Work Discussion

This thesis has presented novel platforms, mechanisms, experimental environments, comparative evaluations and proposed a new cloud orchestration strategy, investigating the benefits of a full-fledged adoption of lightweight virtualization technologies in 5G networks. However, there are still open research issues that can potentially enable further research works that stem from the research investigated in this dissertation. Concluding, we present future research activities as follows:

- *Design and implementation*, we target implementing a complete service-centric multi-domain orchestration platform, equipped with architecture and mechanisms in adherence to the defined design guidelines introduced in section 3.7 as well as incorporating our VTS approach;

- *Integration and evaluation*, we further plan to: (i) integrate our approach into a Kubernetes-based orchestration system and experiment with additional services with particular stringent performance requirements; (ii) evaluate important virtual network optimization mechanisms, such as eBPF and XDP [79]; (iii) exploit complementary unique edge cloud orchestration capabilities in the novel edge cloud infrastructure StarlingX; (iv) assessing NFV orchestration capabilities, such as those proposed in [119],[120]; (v) investigate more sophisticated slice embedding mechanisms (e.g., [121]) and investigate the scalability of multi-domain slice instantiation with near-optimal slice embeddings; and (vi) synchronize our approach with *service shifting* [101] in NVF orchestration framework;

- *Extensive Experimentation*, we also target: (i) experimenting with large-scale multiple PoPs deployments at geographically distributed open-access test-beds (Fed4FIRE+ or GENI [110]) involving our Virtualization technology shifting; (ii) investigating the service performance impact of alternative network or application services (e.g., video streaming and augmented reality) involving alternative hypervisors, heterogeneous server hardware and network connectivity; (iii) applying appropriate prediction and rapid detection mechanisms on the workload behavior (i.e., client requests), including employing novel prediction models or change-point analysis algorithms; and

- *Extension of our cloud resource optimization framework towards*: (i) integrating additional parameters that reflect E2E communication better, e.g., consider topologies with intermediary hops and multiple paths; (ii) incorporating additional service performance metrics (e.g., throughput or energy consumption), that are critical for particular 5G services; and (iii) implementing a sophisticated heuristic solution, focusing on both scheduling and scalability aspects.

As a bottom line, the novel edge cloud platforms, as well as the edge cloud orchestration approaches proposed in this dissertation, should be considered as promising enablers for 5G and beyond ecosystems. Our future goal is to keep integrating new features and mechanisms in order to achieve a holistic edge cloud platform solution.

References

[1] M. Series, "IMT Vision–Framework and overall objectives of the future development of IMT for 2020 and beyond," *Recommendation ITU*, vol. 2083, p. 0, Sep 2015.

[2] 5G PPP Architecture Working Group, white paper, "View on 5G Architecture," 2020, Available: https://5g-ppp.eu/wp-content/uploads/2020/02/5G-PPP-5G-Architecture-White-Paper_final.pdf, Accessed on: Sep 2021.

[3] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan 2017.

[4] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Dec 2014.

[5] S. Ejaz and Z. Iqbal, "Network function virtualization: Challenges and prospects for modernization," in *2018 International Conference on Engineering and Emerging Technologies (ICEET)*. IEEE, Feb 2018, pp. 1–5.

[6] Virtualisation, Network Functions, "An introduction, benefits, enablers, challenges & call for action," in *White Paper, SDN and OpenFlow World Congress*, Oct 2012, p. 73.

[7] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, May 2017.

[8] N. T. Le, M. A. Hossain, A. Islam, D.-y. Kim, Y.-J. Choi, and Y. M. Jang, "Survey of promising technologies for 5G networks," *Mobile information systems*, vol. 2016, Oct 2016.

[9] J. Matias, J. Garay, N. Toledo, J. Unzilla, and E. Jacob, "Toward an SDN-enabled NFV architecture," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 187–193, Apr 2015.

[10] X. Li, M. Samaka, H. A. Chan, D. Bhamare, L. Gupta, C. Guo, and R. Jain, "Network slicing for 5G: Challenges and opportunities," *IEEE Internet Computing*, vol. 21, no. 5, pp. 20–27, Sep 2017.

[11] D. Bhamare, M. Samaka, A. Erbad, R. Jain, and L. Gupta, "Exploring microservices for enhancing internet qos," *Trans. on Emerg. Telecommun. Technol.*, vol. 29, no. 11, p. e3445, Oct 2018.

[12] 5GPPP Technology Board and 5G-IA's Trials Working Groups, white paper, "Edge Computing for 5G Networks," 2021, Available: http://shorturl.at/hxFO9, Accessed on: June 2021.

[13] P. Heidari, Y. Lemieux, and A. Shami, "Qos assurance with light virtualization-a survey," in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec 2016, pp. 558–563.

[14] A. Madhavapeddy and D. J. Scott, "Unikernels: Rise of the virtual library operating system," *Queue*, vol. 11, no. 11, pp. 30–44, Dec 2013.

[15] A. Madhavapeddy, T. Leonard, Skjegstad *et al.*, "Jitsu: Just-in-time summoning of unikernels," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI) 15)*, May 2015, pp. 559–573.

[16] A. Medeiros *et al.*, "End-to-end elasticity control of cloud-network slices," *Internet Technology Letters*, vol. 2, no. 4, p. e106, Jul 2019.

[17] Wang, Xiaofei and Chen, Min and Taleb, Tarik and Ksentini, Adlen and Leung, Victor CM, "Cache in the air: Exploiting content caching and delivery techniques for 5G systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, Feb 2014.

[18] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack, "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," in *Proceedings IEEE INFOCOM Workshops*, Mar 2012, pp. 316–321.

[19] M. Zeni, D. Miorandi, and F. De Pellegrini, "YOUStatAnalyzer: a tool for analysing the dynamics of YouTube content popularity," in *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, Dec 2013, pp. 286–289.

[20] Ö. Alay, A. Lutu, M. Peón-Quirós *et al.*, "Experience: An open platform for experimentation with commercial mobile broadband networks," in *23rd Ann. Int. Conf. on Mobile Computing and Networking, ACM*, Oct. 2017, pp. 70–78.

[21] Sinha, Vivek and Doucet, Frederic and Siska, Chuck and others, "YAML: a tool for hardware design visualization and capture," in *Proc. of 13th Int. Symposium on System Synthesis (ISSS'00)*, Sep. 2000, pp. 9–14.

[22] Node-RED, [Online], Available: https://nodered.org, Accessed on: December 2020.

[23] T. Wauters, B. Vermeulen, W. Vandenberghe *et al.*, "Federation of Internet experimentation facilities: architecture and implementation," in *European Conf. on Networks and Communications (EuCNC)*, Jun. 2014, pp. 1–5.

[24] Pathan, Mukaddim and Buyya, Rajkumar and Vakali, Athena, "Content delivery networks: State of the art, insights, and imperatives," *Content Delivery Networks*, pp. 3–32, 2008.

[25] A.-M. K. Pathan, R. Buyya *et al.*, "A taxonomy and survey of content delivery networks," *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report*, vol. 4, p. 70, Feb 2007.

[26] Cisco, White Paper, "Cisco Annual Internet Report (2018 - 2023)," *https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.*

[27] Akamai Technologies, Inc., [Online], Available: www.akamai.com, Accessed on: October 2021.

[28] Azure Microsoft Interntet, Inc., [Online], Available: https://docs.microsoft.com/el-gr/azure/, Accessed on: October 2021.

[29] Amazon.com, Inc., [Online], Available: https://www.aws.amazon.com/, Accessed on: October 2021.

[30] N. Alliance, "5g white paper," *Next generation mobile networks, white paper*, vol. 1, Feb 2015.

[31] Docker, "The Docker Containerization Platform," [Online]. Available: https://www.docker.com/ Accessed on: July 2021.

[32] Canonical Linux Containers (LXC), [Online]. Available: https://linuxcontainers.org Accessed on: July 2021.

[33] A. Madhavapeddy *et al.*, "Turning Down the LAMP: Software Specialisation for the Cloud," *in Proc. 2nd USENIX Conf. HotCloud*, vol. 10, pp. 11–11, Jun 2010.

[34] J. Martins, M. Ahmed, C. Raiciu *et al.*, "Clickos and the art of network function virtualization," in *Proc. of 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, Apr 2014, pp. 459–473.

[35] J. Cormack, "The rump kernel: A tool for driver development and a toolkit for applications," in *Proc. of the Asian BSD conf.(AsianBSDCon)*, Mar 2015.

[36] A. Kivity, D. Laor, G. Costa *et al.*, "OSv|Optimizing the Operating System for Virtual Machines," in *proc. of the USENIX Annual Technical Conf. (USENIX ATC '14)*, Jun 2014, pp. 61–72.

[37] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, and F. Huici, "My vm is lighter (and safer) than your container," in *Proceedings of the 26th Symposium on Operating Systems Principles*, Oct 2017, pp. 218–233.

[38] F. Lopez-Pires and B. Baran, "Virtual machine placement literature review," *arXiv preprint arXiv:1506.01509*, Feb 2015.

[39] Aue, Alexander and Hörmann, Siegfried and Horváth, Lajos and Reimherr, Matthew, "Break detection in the covariance structure of multivariate time series models," *The Annals of Statistics*, vol. 37, no. 6B, pp. 4046–4087, Dec 2009.

[40] Hoga, Yannick, "Monitoring multivariate time series," *Journal of Multivariate Analysis*, vol. 155, pp. 105–121, Mar 2017.

[41] Tartakovsky, Alexander G and Polunchenko, Aleksey S and Sokolov, Grigory, "Efficient computer network anomaly detection by changepoint detection methods," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 1, pp. 4–11, Dec 2012.

[42] Chandola, Varun and Banerjee, Arindam and Kumar, Vipin, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, Jul 2009.

[43] Marnerides, Angelos K and Schaeffer-Filho, Alberto and Mauthe, Andreas, "Traffic anomaly diagnosis in internet backbone networks: A survey," *Computer Networks*, vol. 73, pp. 224–243, Nov 2014.

[44] Nevat, Ido and Divakaran, Dinil Mon and Nagarajan, Sai Ganesh and Zhang, Pengfei and Su, Le and Ko, Li Ling and Thing, Vrizlynn LL, "Anomaly detection and attribution in networks with temporally correlated traffic," *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 131–144, Dec 2017.

[45] Turner, Daniel and Levchenko, Kirill and Snoeren, Alex C and Savage, Stefan, "California fault lines: understanding the causes and impact of network failures," in *Proceedings of the ACM SIGCOMM 2010 Conference*, Aug 2010, pp. 315–326.

[46] G. Siracusano, R. Bifulco, M. Trevisan *et al.*, "Re-designing Dynamic Content Delivery in the Light of a Virtualized Infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2574–2585, Oct 2017.

[47] S. Kuenzer, A. Ivanov, F. Manco *et al.*, "Unikernels Everywhere: The Case for Elastic CDNs," *In Proc. of the 13th ACM SIGPLAN Notices*, vol. 52, no. 7, pp. 15–29, Apr 2017.

[48] CDNSim - a stream-level simulator for large content delivery networks, [Online], Available: https://github.com/cnplab/cdnsim, Accessed on: October 2021.

[49] P. Marques, C. Silva, V. Frascolla *et al.*, "Experiments overview of the eu-brazil futebol project," in *Proc. of the 26th European Conf. on Networks and Commun.*, Jun. 2017, pp. 1–2.

[50] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "Tosca: portable automated deployment and management of cloud applications," in *Advanced Web Services*. Springer, 2014, pp. 527–549.

[51] A. Gavras, S. Denazis, C. Tranoris *et al.*, "Requirements and design of 5g experimental environments for vertical industry innovations," in *Global Wireless Summit (GWS)*, Oct. 2017, pp. 165–169.

[52] I. Afolabi, A. Ksentini, M. Bagaa *et al.*, "Towards 5g network slicing over multiple-domains," *IEICE Transactions on Commun.*, vol. 100, no. 11, Nov. 2017.

[53] X. Foukas, M. K. Marina, and K. Kontovasilis, "Orion: Ran slicing for a flexible and cost-effective multi-service mobile network architecture," in *ACM Proc. of the 23rd Annual Int. Conf. on Mobile Computing and Networking*, Oct. 2017, pp. 127–140.

[54] S. Rizou, P. Athanasoulis, P. Andriani *et al.*, "A service platform architecture enabling programmable edge-to-cloud virtualization for the 5g media industry," in *IEEE Int. Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, Jun. 2018.

[55] N. Makris, C. Zarafetas, S. Kechagias *et al.*, "Enabling open access to lte network components; the nitos testbed paradigm," in *Proc. of the IEEE conf. on Network Softwarization (NetSoft)*, Apr. 2015, pp. 1–6.

[56] Softwarized Wireless Network, [Online], Available: http://emulab.swn.uom.gr/, Accessed on: December 2020.

[57] S. Skaperas, L. Mamatas, and A. Chorti, "Real-time video content popularity detection based on mean change point analysis," *IEEE Access*, vol. 7, pp. 142 246–142 260, 2019.

[58] L. Horváth, P. Kokoszka, and J. Steinebach, "Testing for changes in multivariate dependent observations with an application to temperature changes," *Journal of Multivariate Analysis*, vol. 68, no. 1, pp. 96–119, Jan 1999.

[59] P. Burridge and B. U. K. D. of Economics;, *A Very Simple, Positive Semi-Definite, Heteroskedasticity and Autocorrelation Consistent Covariance Matrix*. University of Birmingham, Department of Economics, 1991.

[60] L. Y. Vostrikova, "Detecting disorder in multidimensional random processes," in *Doklady akademii nauk*, vol. 259, no. 2, 1981, pp. 270–274.

[61] C. Inclan and G. C. Tiao, "Use of cumulative sums of squares for retrospective detection of changes of variance," *Journal of the American Statistical Association*, vol. 89, no. 427, pp. 913–923, Sep 1994.

[62] G. Appel, "Become your own technical analyst: How to identify significant market turning points using the moving average convergence-divergence indicator or macd," *The Journal of Wealth Management*, vol. 6, no. 1, pp. 27–36, Apr 2003.

[63] S. Fremdt, "Asymptotic distribution of the delay time in page's sequential procedure," *Journal of Statistical Planning and Inference*, vol. 145, pp. 74–91, Feb 2014.

[64] Ansible, [Online], Available: https://www.ansible.com, Accessed on: October 2021.

[65] Httperf HTTP load generator, [Online], Available: https://github.com/httperf/httperf, Accessed on: October 2021.

[66] Collectd - The system statistics collection daemon, [Online], Available: https://collectd.org/download.shtml, Accessed on: October 2021.

[67] InfluxData (InfluxDB) | Time Series Database Monitoring & Analytics, [Online], Available: https://www.influxdata.com/developers/, Accessed on: October 2021.

[68] Grafana - The open platform for analytics and monitoring, [Online], Available: https://grafana.com/, Accessed on: October 2021.

[69] T. Theodorou, G. Violettas, P. Valsamas, S. Petridou, and L. Mamatas, "A multi-protocol software-defined networking solution for the internet of things," *IEEE Communications Magazine*, vol. 57, no. 10, pp. 42–48, Oct 2019.

[70] M. Berman, J. S. Chase, L. Landweber *et al.*, "Geni: A federated testbed for innovative network experiments," *Computer Networks, Elsevier*, vol. 61, pp. 5–23, Mar. 2014.

[71] B. Nogales, I. Vidal, D. R. Lopez *et al.*, "Design and deployment of an open management and orchestration platform for multi-site nfv experimentation," vol. 57, no. 1, pp. 20–27, Jan 2019.

[72] P. Valsamas, S. Skaperas, and L. Mamatas, "Elastic content distribution based on unikernels and change-point analysis," in *Proc. 24th Eur. Wireless Conf.(EW)*, May 2018, pp. 1–7.

[73] S. Skaperas, L. Mamatas, and A. Chorti, "Early video content popularity detection with change point analysis," in *Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–7.

[74] P. Di Francesco, P. Lago, and I. Malavolta, "Migrating towards microservice architectures: An industrial survey," in *IEEE Int. Conf. on Software Architecture (ICSA)*, 2018, p. Apr.

[75] T. Taleb *et al.*, "White Paper on 6G Networking," Jun 2020 Available: https://www.6gchannel.com/, Accessed on: July 2021.

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

[77] R. Behravesh, E. Coronado, and R. Riggio, "Performance Evaluation on Virtualization Technologies for NFV Deployment in 5G Networks," in *IEEE Conf. on Network Softwarization (NetSoft)*, Jun 2019, pp. 24–29.

[78] I. Mavridis and H. Karatza, "Lightweight virtualization approaches for software-defined systems and cloud computing: An evaluation of unikernels and containers," in *the 6th International Conf. on Software Defined Systems (SDS)*, Jun 2019, pp. 171–178.

[79] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, "Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, Feb 2020.

[80] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88–145, Sep 2018.

[81] ETSI GR NFV-IFA 029, \Network Functions Virtualisation (NFV) Release 3; Architecture; Report on the Enhancements of the NFV architecture towards \Cloud-native" and \PaaS"", V3.3.1 (2019-11), [Online], Available: https://www.etsi.org/deliver/etsi_gr/NFV-IFA/001_099/029/03.03.01_60/gr_NFV-IFA029v030301p.pdf, Accessed on: July 2021.

[82] ETSI GR MEC 027, \Multi-access Edge Computing (MEC); Study on MEC support for alternative virtualization technologies", V2.1.1 (2019-11), [Online], Available: http://shorturl.at/fDF59, Accessed on: December 2020.

[83] 5GPPP Technology Board and 5G-IA's Trials Working Groups, white paper, "Edge Computing for 5G Networks," 2019, Available: https://bscw.5g-ppp.eu/pub/bscw.cgi/d397473/EdgeComputingFor5GNetworks.pdf, Accessed on: July 2021.

[84] M. Plauth, L. Feinbube, and A. Polze, "A performance evaluation of lightweight approaches to virtualization," *Cloud Computing*, vol. 2017, p. 14, Feb 2017.

[85] T. Kurek, "Unikernel Network Functions: A Journey Beyond the Containers," *IEEE Communications Magazine*, vol. 57, no. 12, pp. 15–19, 2019.

[86] I. Briggs, M. Day, Y. Guo, P. Marheine, and E. Eide, "A performance evaluation of unikernels," in *Technical Report*, 2014.

[87] T. Goethals, M. Sebrechts, A. Atrey, B. Volckaert, and F. De Turck, "Unikernels vs Containers: An In-Depth Benchmarking Study in the context of Microservice Applications," in *in Proc. of 8th International Symposium on Cloud and Service Computing (SC2)*, Nov 2018, pp. 1–8.

[88] B. Xavier, T. Ferreto, and L. Jersak, "Time provisioning evaluation of kvm, docker and unikernels in a cloud platform," in *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2016, pp. 277–280.

[89] O. A. Ezenwigbo, J. Ramirez, G. Karthick, G. Mapp, and R. Trestian, "Exploring the Provision of Reliable Network Storage in Highly Mobile Environments," in *Proc. of the 13th International Conf. on Communications (COMM)*, Jun 2020, pp. 255–260.

[90] J. B. Filipe, F. Meneses, A. Rehman, D. Corujo, and R. L. Aguiar, "A performance comparison of containers and unikernels for reliable 5g environments,"

in *15th International Conference on the Design of Reliable Communication Networks (DRCN)*, Mar 2019, pp. 99–106.

[91] V. Aggarwal and B. Thangaraju, “Performance Analysis of Virtualisation Technologies in NFV and Edge Deployments,” in *Proc. of the IEEE International Conf. on Electronics, Computing and Communication Technologies (CONECCT)*, July 2020, pp. 1–5.

[92] Langston Nashold and Rayan Krishnan, “Using LSTM and SARIMA Models to Forecast Cluster CPU Usage,” *CoRR*, 2020.

[93] L. M. Contreras *et al.*, “Toward cloud-ready transport networks,” *IEEE Communications Magazine*, vol. 50, no. 9, pp. 48–55, Sep 2012.

[94] L. M. Contreras, J. Baliosian, P. Martínez-Julia, and J. Serrat, “Computing at the Edge: But, what Edge?” in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, Apr 2020, pp. 1–9.

[95] S. Clayman, A. Neto, F. Verdi, S. Correa, S. Sampaio, I. Sakelariou, L. Mamatas, R. Pasquini, K. Cardoso, F. Tusa, C. Rothenberg, and J. Serrat, “The necos approach to end-to-end cloud-network slicing as a service,” *IEEE Communications Magazine*, vol. 59, no. 3, pp. 91–97, 2021.

[96] P. Valsamas, P. Papadimitriou, I. Sakellariou, S. Petridou, L. Mamatas, S. Clayman, F. Tusa, and A. Galis, “Multi-PoP network slice deployment: A feasibility study,” in *IEEE 8th International Conference on Cloud Networking (CloudNet)*, Nov 2019, pp. 1–6.

[97] P. Valsamas, L. Mamatas, and L. M. Contreras, “Exploiting Edge Cloud Heterogeneity for 5G Networks and Beyond,” *IEEE Transactions on Network and Service Management, Submitted*, Dec 2021.

[98] Kubernetes, [Online], Available: https://www.kubernetes.io, Accessed on: November 2021.

[99] D. Balla, C. Simon, and M. Maliosz, “Adaptive scaling of Kubernetes pods,” in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, Apr 2020, pp. 1–5.

[100] Z. Ding and Q. Huang, “COPA: A Combined Autoscaling Method for Kubernetes,” in *IEEE International Conference on Web Services (ICWS)*, Sep 2021, pp. 416–425.

[101] F. Malandrino, C. F. Chiasserini, and G. Landi, “Service shifting: A paradigm for service resilience in 5g,” *IEEE Communications Magazine*, vol. 57, no. 9, pp. 120–125, Sep 2019.

[102] F. Paganelli, P. Cappanera, A. Brogi, and R. Falco, “Profit-aware placement of multi-flavoured VNF chains,” in *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*, Nov 2021, pp. 48–55.

[103] A. Lertsinsrubtavee, A. Ali, C. Molina-Jimenez *et al.*, “Picasso: A lightweight edge computing platform,” in *IEEE 6th International Conf. on Cloud Networking (CloudNet)*, Sep 2017, pp. 1–7.

[104] V. Cozzolino, A. Y. Ding, and J. Ott, “Fades: Fine-grained edge offloading with unikernels,” in *Proc. of the Workshop on Hot Topics in Container Networking and Networked Systems*, Aug 2017, pp. 36–41.

[105] V. Cozzolino, J. Ott, A. Y. Ding, and R. Mortier, “Ecco: Edge-cloud chaining and orchestration framework for road context assessment,” in *IEEE/ACM Fifth International Conf. on Internet-of-Things Design and Implementation (IoTDI)*, Apr 2020, pp. 223–230.

[106] Kubeedge, [Online]. Available: https://kubeedge.io/en/ Accessed on: July 2021.

[107] K3S, [Online]. Available: https://k3s.io/ Accessed on: July 2021.

[108] Next Generation Platform as a Service H2020 project (NGPaas), Available: http://ngpaas.eu/.

[109] Superfluidity, H2020 Project, Available: http://superfluidity.eu/.

[110] P. Valsamas, I. Sakellariou, S. Petridou, and L. Mamatas, "A Multi-domain Experimentation Environment for 5G Media Verticals," in *IEEE INFOCOM Workshop on Computer and Networking Experimental Research using Testbeds (CNERT)*, Apr 2019, pp. 461–466.

[111] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized iot service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, no. 4, pp. 427–443, Dec 2017.

[112] A. Yousefpour, G. Ishigaki, and J. P. Jue, "Fog computing: Towards minimizing delay in the internet of things," in *Proc. IEEE 1st International Conf. on Edge Computing (EDGE)*, Jun 2017, pp. 17–24.

[113] A. Dalvandi, M. Gurusamy, and K. C. Chua, "Time-aware vm-placement and routing with bandwidth guarantees in green cloud data centers," in *Proc. IEEE 5th International Conf. on Cloud Computing Technology and Science(CloudCom)*, vol. 1, Dec 2013, pp. 212–217.

[114] A. S. Milani and N. J. Navimipour, "Load balancing mechanisms and techniques in the cloud environments: Systematic literature review and future trends," *Journal of Network and Computer Applications*, vol. 71, pp. 86–98, Aug 2016.

[115] X. Guan, X. Wan, B. Y. Choi, S. Song, and J. Zhu, "Application oriented dynamic resource allocation for data centers using docker containers," *IEEE Communications Letters*, vol. 21, no. 3, pp. 504–507, Dec 2016.

[116] K. Kaur, S. Garg, G. Kaddoum, F. Gagnon, and D. N. K. Jayakody, "Enlob: Energy and load balancing-driven container placement strategy for data centers," in *IEEE Globecom Workshops (GC Wkshps)*, Dec 2019, pp. 1–6.

[117] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *Proc. IEEE international conf. on edge computing (EDGE)*, Jun 2017, pp. 47–54.

[118] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, W. Xie, and J. P. Jue, "FOGPLAN: A lightweight QoS-aware dynamic fog service provisioning framework," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5080–5096, Jan 2019.

[119] Castanho, Matheus S and Dominicini, Cristina K and Villacça, Rodolfo S and Martinello, Magnos and Ribeiro, RN Moises, "Phantomsfc: A fully virtualized and agnostic service function chaining architecture," in *IEEE Symposium on Computers and Communications (ISCC)*, Jun 2018, pp. 354–359.

[120] Dominicini, Cristina K and Vassoler, Gilmar L and Meneses, Leonardo F and Villaca, Rodolfo S and Ribeiro, Moises RN and Martinello, Magnos, "Virtphy: Fully programmable nfv orchestration architecture for edge data centers," *in IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 817–830, Sep 2017.

[121] Dietrich, David and Rizk, Amr and Papadimitriou, Panagiotis, "Multi-provider virtual network embedding with limited information disclosure," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 188–201.

Appendices

A Funding

This work was partially supported by the open call scheme of the H2020 MONROE (grant agr. nº 644399) project, partially by EU's Horizon 2020 research and innovation program through the 4th open call scheme of the FED4FIRE+ "Unikernel-based CDNs for 5G Networks(Unic)" Project (grant agr. nº 732638) and partially by EU's Horizon 2020 Research and Innovation Program through the EU-BRA Horizon 2020 "Novel Enablers for Cloud Slicing(NECOS)" Project (grant agr. nº 777067).