Doctoral Dissertation
Doctoral Program in Electronics and Telecommunications Engineering (32.nd cycle)

Energy Management in Virtualized Networks

Senay Semu Tadesse

* * * * * *

Supervisors

Prof. Carla-Fabiana Chiasserini, Supervisor

Politecnico di Torino
February 11, 2020

. . . .

Senay Semu Tadesse

Turin, February 11, 2020

Summary

Efficient energy management is a key necessity for the future generation networks. In line with this, the 5G-Infrastructure Public-Private Partnership (PPP) has set the following requirements as Key Performance Indicators (KPIs) for energy management: (i) energy efficiency improvement by at least a factor of 3 and (ii) reduction of energy cost per bit by a factor of 10. Virtualization plays a key role in energy management, wherein efficient utilization of storage and computation resources and network infrastructure is made possible. In this thesis, we address the issue of improving energy efficiency by formulating optimal routing strategies in SDN based 5G backhaul networks. Moreover, we investigate different virtualization technologies in order to establish how well they perform in terms of different performance measures: resource usage and energy consumption. In addition, we analyze IoT traffic, which is characterized by long inactivity times and then quasi-synchronous transmissions, to model the Evolved Packet Core (EPC) and to effectively scale the EPC components to real-time resource requirements of the peculiar IoT traffic.

First, we consider optimization of energy expenditure in the backhaul network, one of the main areas where efficient resource usage in 5G wireless networks can contribute to a tremendous amount of energy saving. We contribute to this aspect by developing an Energy Management and Monitoring Application (EMMA) that runs on top of a Software Defined Network (SDN) controller. This application is able to control the state of links and routing nodes according to the traffic load, resulting in the activation of minimal number of links and nodes. EMMA tries to route incoming traffic requests through already active links. It activates new links and/or nodes, whenever the already available links and nodes can no longer meet the QoS requirements for all flows. On the contrary, EMMA turns off links and nodes whenever they become underutilized. Before turning off links and nodes, the application reroutes the flows passing through these links and nodes to a path that can support them. We compare our energy efficient routing strategy, EMMA, against the optimal routing strategy and No-Power Saving Strategy to have insight into the amount of energy saved with EMMA in place as a routing application.

Next, we study the performance of virtualization environments where Virtual Network Functions (VNFs) are deployed, in the context of Multi-access Edge Computing (MEC). To this end, we carry out a wide range of experiments on two representative virtualization technologies: light-weight containerization technology and traditional virtualization technology. We profile and study the resource usage and energy consumption, in this virtualization environments, of different applications with unique requirements for system resources. For our experiments, we chose VirtualBox and Docker from traditional virtualization technology and light-weight containerization technology, respectively. We study the performance of a number of virtualized synthetic and real-world applications in both of the above mentioned virtualization environments.

Finally, we consider Massive-Internet-of-Things (MIoT), one of the main cases in which 5G is used, to characterize the Cellular MME by analyzing its delay and control traffic overhead in serving requests from Massive IoT devices. We study the traffic arrival analytically and Evolved Packet Core (EPC) components through the profiling of real-world EPC implementation from OpenAirInterface, which reveal Mobility Management Entity (MME) is the bottleneck among the EPC components. Then, we use the results of the analytical and practical studies to model the MME as M/D/1-PS queue. We use the MME model to obtain a closed form expression of the delay at which bearer requests are served. In order to verify the analytical results, we perform several simulations on the 3GPP IoT traffic model and on real traffic traces. Finally, we exploit our model for proper scaling of the assigned EPC system resources to match the traffic requirements.

Acknowledgements

First of all, I would like to thank the Almighty God for giving me the knowledge and strength to study my PhD and finish it successfully.

I would like to express my sincere gratitude to my supervisor Prof. Carla Fabiana Chiasserini for her kind support and guidance throughout my PhD study. Her patience and encouragement combined with her acumen and immense knowledge helped me successfully complete my study.

I would like to thank Prof. Claudio Casetti for his contributions and insightful comments in this work. I would also like to express my thanks to Prof. Francesco Malandrino and Dr. Christian Vitale for their kind support during my research.

I would like to thank my beloved parents, Semu Tadesse and Tadelech Abera. I also thank my brothers Bahiru and Wasehun and my sisters Serkalem and Scince for their generous and kind support. I would also like to express my gratitude to my husband Eskadmas for his encouragement, support and invaluable advice.

I would like to dedicate this thesis to my family for their love, prayers, and encouragement!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Energy efficiency is considered one of the necessities of the future generation wireless network. In fact, the 5G-Infrastructure Public-Private Partnership (PPP) has set the following Key Performance Indicators for energy management: (i) to reduce the energy consumption, i.e., joule/bit, by a factor of 10 and (ii) to improve energy efficiency by at least a factor of 3 [1, 2]. The reduction in energy consumption is intended to be achieved by a combination of approaches implemented both in the radio network and core network of 5G [3]. These approaches include implementing 5G New Radio to reduce signaling, using efficient caching techniques to reduce the frequency of access to the internet and even to the backhaul, and deploying cell, node and link switch off techniques whereby network switches and links are switched off and turned back on as needed. Implementing these collective approaches will result in saving network and computational resources, which in turn leads to a reduction in the energy consumption of the network.

A huge shift in the networking paradigm is needed to achieve the above mentioned KPIs for energy management, with Network Function Virtualization (NFV), Software Defined Networking (SDN) and Multi-access Edge Computing (MEC) being at the forefront of this change. NFV decouples the network functions from the hardware, thereby creating Virtual Network Functions (VNFs). VNFs are (network) functions implemented in software and deployed in a server or in an isolated virtualization environment within a server. As opposed to (network) functions implemented in proprietary hardware, VNFs come with the ability to be deployed in a general purpose high capacity compute machine in parallel with other functions. SDN, on the other hand, is a new paradigm to networking which decouples the control and the data forwarding functions [4]. SDN makes the forwarding decision on a logically centralized control plane, while the network switches are used to merely forward packets based on the forwarding rules decided by the control plane. The other new paradigm, MEC, places frequently required and latency constrained computational functions and data close to the user [5], thereby reducing backhaul traffic and latency.

In this thesis, we first adopt SDN to develop energy aware traffic routing using the Open Network Operating System (ONOS) [6] for the SDN controller and using the Mininet data layer network [7]. Then, we study the performance of different real-world and synthetic applications in different virtualization environments in terms of energy and resource usage in the context of MEC scenarios. We finally use a real-world 4G software implementation for the purpose of characterizing the control traffic of Massive-IoT devices and developing a model that can be used to scale dynamically the resources assigned to EPC components in order to satisfy the traffic requirements.

In the rest of this chapter, we will present an overview of 5G, SDN, NFV, MEC and IoT.

1.1 An Overview of 5G Networks

The future generation cellular network, i.e., 5G, consists of radio network and core network. The radio network known as New Generation Radio Access Network (NG-RAN) consists of a set of base stations known as next generation Node-Bs (gNBs). The 5G Core network consists of the following main elements: User Plane Functions (UPF), Access and Mobility Management Function (AMF), Session Management Function (SMF), Policy Control Function (PCF) and User Data Management (UDM).

5G is architectured so that it is capable of supporting different requirements from different vertical industries [8]. These include fulfilling the high density requirements of massive Machine Type Communications (MTC), meeting the low latency and high reliability requirements of applications like autonomous driving, and keeping up with high bandwidth requirements.

5G's potential to support these diversified requirements of consumer, service and business applications relies heavily on the SDN paradigm and Network Function Virtualization (NFV) [4]. SDN decouples the control layer from the data layer, introducing more flexibility in implementation of network control functions such

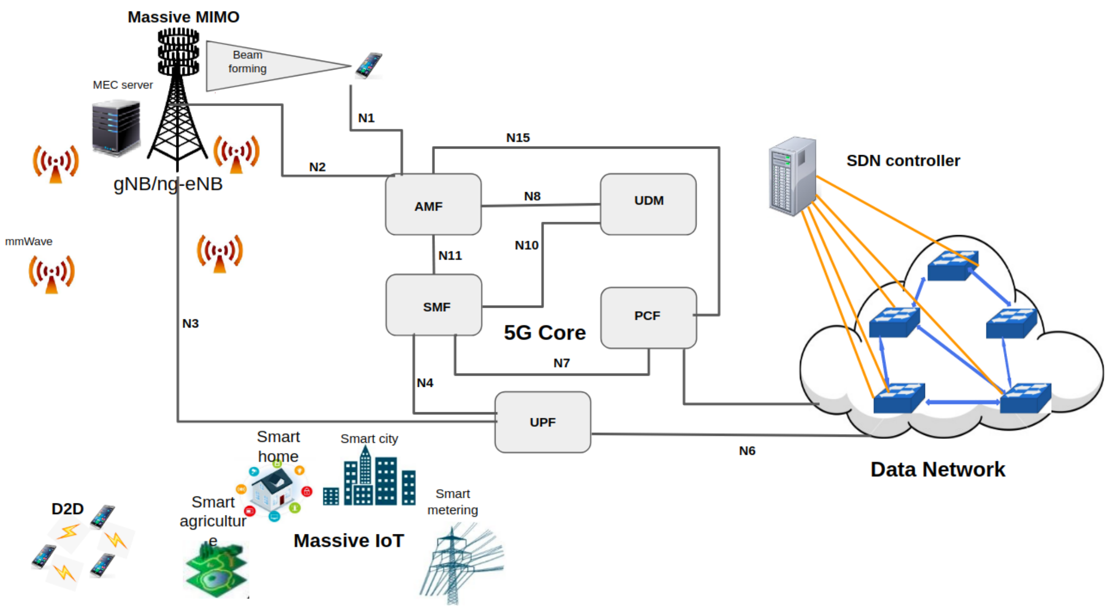

Figure 1.1: Schematic Diagram of Next Generation 5G Wireless Networks.

as routing protocols. Contrary to the traditional networking paradigm, which uses a distributed control plane embedded in each forwarding or routing device, SDN uses a centralized controller to make routing decisions, thereby leading to the use of simple forwarding devices which are directed by the controller. 5G leverages this virtue of SDN to easily manage the network and provide abstraction of the physical network to the applications running on the application layer of the SDN controller. On the other hand, NFV deploys network functions on general purpose machines, decoupling them from proprietary devices. Moreover, virtualization technologies are leveraged to deploy a number of network functions on high compute machines in the cloud to achieve more efficient use of resources [9]. The introduction of these paradigms into the mobile network decreases the development and implementation of new network applications, eases the management of network devices and decreases investment and running costs. In addition, 5G is enabled by Massive-MIMO, Device to Device Communication, millimeter wave, beam forming and small cells.

It is greatly anticipated that 5G will provide support to Enhanced Mobile Broadband (MBB) (supporting up to 10Gbps), Ultra-reliable and Low-latency Communications (ULC) (1ms) and Massive Machine Type Communications (MMTC) (10 thousand devices per km^2) [8]. Different network slices are created on top of the same physical network using SDN and NFV to support the requirements of MBB, ULC and MMTC applications. In this way, services can be automatically generated to serve specific requirements, maintained or terminated. This results in a significant reduction of subsequent operating expenditure.

5G introduces computing capability into the edge network with the new paradigm referred to as Multi-access Edge Computing. Network functions and services are deployed at the edge of the network on MEC servers located at the base stations. MEC paradigm is introduced into 5G with the purpose of reducing the latency in the network by hosting application near to where the user is located [5]. This is a critical requirement for latency constrained applications such as autonomous driving.

In this thesis, we contribute to the realization of one of the KPIs of 5G for energy management, i.e., reducing the energy cost (joule/bit) by a factor of 10, set by 5G-PPP. In this regard, we explore a variety of techniques that can be used to optimize the energy usage of 5G network. These techniques include optimization of routing applications, carrying out a range of experimental measurements in testbeds to understand and model virtualization environments in MEC scenarios and to understand and model the core network with respect to IoT traffic.

1.2 Software Defined Networking

Software Defined Networking is an emerging networking paradigm which is revolutionizing the way networks operate. SDN provides flexibility to the network enabling it to address varying requirements of different industries. SDN separates the control logic from the forwarding logic, thus making the switches forwarding protocols to be directly programmable from a centralized controller [10]. In SDN paradigm, the routing decision for each flow is made by a centralized SDN controller, which has the global view of the network.

The centralized routing approach of SDN abandons the traditional networking, where routing decisions are made in distributed manner. Instead, it adopts a routing approach which is based on the computation of the flow paths centrally by an SDN controller. Such approach highly improves the ease of network control and deployment of new network services. When a new flow enters into the network, the node encountering the flow checks if it has a forwarding rule in its forwarding table. Whenever a flow rule is not found in the forwarding table, the node forwards the flow directly to the SDN controller asking for a forwarding rule. Having received a request for a flow path, the SDN controller computes the end to end path based on the routing application in use. Then, the SDN controller forwards the packet to its next hop based on the path it just computed. Meanwhile, the forwarding rule is sent to every node which is on this path computed for the new flow.

Shifting from traditional networking into SDN enables providers for better and efficient management of their network. SDN is deemed vital, specially in managing the ever increasing role of virtualization in networks . Moreover, SDN enables efficient and reliable traffic routing, which is one of the main topics we consider in this thesis.

SDN controllers have northbound and southbound APIs. Northbound APIs are used to communicate with the the application layer and provide abstraction of the physical network to the applications. On the other hand, southbound APIs are used to interact with physical devices in the network: install flow rules in the devices and communicate control messages and statistics between the controller and the data layer.

SDN together with NFV is deemed to define future network and radically change the way networks operate. With SDN paradigm, switching devices are used to just forward data, leaving the control logic to be implemented in a centralized controller device. Taking the control protocol out of individual routing devices and implementing it at a different upper layer paves the way for the network to be easily managed. Moreover, SDN makes it easier to deploy new services. This will be possible by running applications on top of the SDN controller which will have a global view of the network. 5G will leverage this virtue of SDN to easily manage network devices, orchestrate applications deployed with NFV and to create network slices from the same physical network. With NFV, which goes hand in hand with SDN, it is possible to deploy, undepoly and migrate multiple applications across nodes in the network.

1.3 Energy Efficient Routing with SDN Paradigm

In SDN paradigm, applications run on top of the SDN controller. An SDN controller, which has a global view of the physical network, provides applications running on top of it, abstraction of the network entities, hiding the complexity therein. A routing application is one of such applications deployed on the application layer of the SDN controller; a routing application decides as to which next node in the network, packets from a given flow, should be forwarded. The decision is made centrally for all flows, and the forwarding rules are installed to the switches comprising the paths. We develop an energy efficient routing application to run on top of an SDN controller assuming SDN links and nodes as capable of being turned on and off as needed by the SDN controller southbound APIs.

We make use of SDN controllers to implement a routing protocol which finds shortest path for the flow in an already active part of the network. In implementing this protocol we try to minimize the number of active nodes and links, therefore minimizing the energy required to operate the network.

1.4 Network Function Virtualization

Network functions and services, which were used to be implemented in proprietary hardware devices, are massively being implemented as a software. Such softwarization of network functions provides flexibility to deploy these applications on general purpose hardware. These applications are not usually run directly on the machines, instead they are run in an isolated virtualization environment [9, 11]. Running a software in an isolated virtualization environment enables the hardware to run multiple instances of one application or different applications while keeping them isolated from each other. In addition, running applications in this way enables one to control the resource usage and the state (active or inactive) through NFV Orchestrator (NFVO).

In this regard, we study the performance of several synthetic and real-world applications, when run in different virtualization environments; we undertake this study in order to identify a better virtualization technology in terms of energy consumption. Moreover, we will model the power consumption as a function of the resource usage based on data obtained from the profiling of synthetic applications. These models can be a tool to estimate power consumption of real-world applications. The study of the performance of virtualized applications is deemed very important in MEC scenarios where such applications are started and ended as per the user request.

1.5 Massive Internet of Things

Provisioning Massive Internet of Things is one of the intended goals of 5G [12]. Due to the peculiarity of IoT traffic, with long inactivity time and quasi-synchronous transmissions, 3GPP has introduced a new protocol for connection establishment and data delivery from IoT devices. This new protocol enables the core network to establish a connection for IoT devices and deliver the data therein right away through the Mobility Management Entity (MME) which in 5G correspond to the

AMF. The AMF/MME receives all connection and session related information from the User Equipment (UE) (N1/N2) as shown in 1.1 but is responsible only for handling connection and mobility management tasks. All messages related to session management are forwarded over the N11 interface to the SMF. We focus our attention into the protocol introduced in 5G for IoT devices connectivity and model the MME based on the traffic arrival process from IoT devices and use this model to scale the resources of the MME according to the number of connection requests. In this work, my contribution mainly consisted in performing the experimental study, which was used to show some fundamental aspects of the system behavior and was essential to the development of the theoretical analysis.

1.6 Structure of the Thesis

In Chapter 2, we present the energy efficient routing strategy in SDN based backhaul network. In this chapter, we first present the formulation of the problem. Then, we present a heuristic algorithm that approximates the optimization problem. Thereafter, the chapter discusses the Energy Management and Monitoring Application (EMMA), developed in Java to run on the application layer of an SDN platform called Open Network Operating System (ONOS). This chapter ends with the analysis of simulation and optimization results. Chapter 3, deals with the performance measurements and modelling resource usage of applications deployed in virtualization environments. It starts by describing the methodology used and proceeds into describing numerical results and models. This chapter then presents a related work overview and finally concludes with a summary of the main observations. In Chapter 4, we first discuss the mathematical analysis of the model for the traffic arrival rate and delay characterization. Then we present simulation and emulation results obtained from real-world 4G implementation to back up our analytical model. In Chapter 5, we present concluding remarks.

Chapter 2

Energy Efficiency in 5G Backhaul Routing

The work in this chapter has been published in:

- Senay Semu Tadesse, Carla Fabiana Chiasserini, Claudio Ettore Casetti, Giada Landi, “Energy-efficient Traffic Allocation in SDN-based Backhaul Networks: Theory and Implementation”. In: IEEE Consumer Communications and Networking Conference (CCNC). 2017

2.1 Motivation

5G networks are expected to be highly energy efficient, with a 10 times lower energy consumption than today’s systems. This requires for an efficient use of the network infrastructure among other things. A part of the network infrastructure to act on to achieve more energy efficiency is the backhaul network. Controlling the allocation of traffic flows and the nodes operational state on the backhaul network can lead to a significant amount of energy saving. Controlling of the nodes operational states can be made easier to achieve with the adoption of Software Defined Networking (SDN); SDN also provides more flexible, manageable and resource efficient traffic allocation.

The 5G-Infrastructure-PPP (Public-Private Partnership), created to design and deliver architectures, technologies and standards for 5G communication, has set the following KPIs (Key Performance Indicators) for energy management [1, 2]: (i) energy efficiency improvement by at least a factor of 3 and (ii) reduction of energy cost per bit by a factor of 10. Of course, no single solution can achieve such ambitious goals. Instead, they should be achieved through orchestrated actions, involving a fully-unified, automated control and management plane, that oversees radio resources as well as computation and transport resources in the fronthaul and backhaul network. A key role is played by Software-Defined Networking and Network Functions Virtualization (NFV), tasked with the control and coordination of hundreds of nodes that need to be reconfigured on the fly in order to optimize utilization and QoS, in view of rapidly changing traffic flows. Among the coordinated actions that can be taken are the de-activation or decommissioning of scarcely used network portions, including links and switches, and the flexible re-routing of existing flows so as to jointly address energy saving and QoE requirements.

In this chapter, we address the latter action, by designing and evaluating an Energy Monitoring and Management Application (EMMA), that can minimize energy consumption of the backhaul network. We used Mininet [7] network emulator to test EMMA. Consistently with the pervasive use of SDN solutions expected in 5G networks, EMMA natively interacts with the Northbound interface of ONOS [6], a popular carrier-grade network operating system, which uses OpenFlow protocol as its southbound interface to control and program flow tables of OpenFlow switches that we used in our emulated Mininet network.

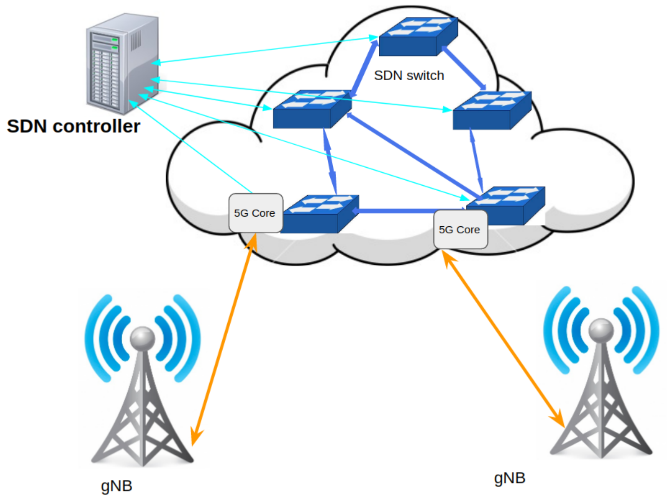

Figure 2.1: Schematic Diagram of SDN in 5G Wireless Networks.

The design of EMMA hinges upon heuristic algorithms for the dynamic routing of flows and the management of the resulting link and switch activity. These algorithms represent a heuristic solution to a non-linear integer problem that aims at minimizing the instantaneous power consumption of nodes and links. With the former action, the heuristic algorithm tries to consolidate flows in to minimum number of links and switches while with the later action, the heuristic algorithm turns of links and subsequently switches that are idle or turns on links and/or nodes whenever the active network can no longer support the QoS requirements of flows. Performance evaluation has been done by comparing the optimum obtained through the above optimization formulation, with practical results derived by implementing the algorithms in an SDN network emulation environment.

We summarize our main novel contributions as follows:

- *Analytical formulation:* We formulate energy efficient flow routing on the backhaul network as an optimization problem;
- *Heuristic Algorithm:* In light of the above optimization problem complexity, which impairs the solution in large-scale scenarios, we then propose a heuristic approach. Our scheme, named EMMA, aims to both turn off idle nodes and concentrate traffic on the smallest possible set of links, which in its turn increases the number of idle nodes;
- *Implementation and Results:* We implement EMMA on top of Open Network Operating System (ONOS), a carrier grade SDN controller. And derive experimental results by emulating the network through Mininet.

Our results show that EMMA provides excellent energy saving performance, which closely approaches the optimum. In larger network scenarios, the gain in energy consumption that EMMA provides with respect to the simple benchmark where all nodes are active, is extremely high, reaching almost 1 under medium-low traffic load.

The rest of the chapter is organized as follows. We begin by briefly discussing the project to which this work was a part of in Section 2.2. Then, in Section 2.3, we introduce the power model for OpenFlow switches that we adopt and formalize the problem under study. In Section 2.4, we will present the heuristic scheme for EMMA. In Section 2.5, we will introduce the SDN paradigm on which EMMA's implementation is based. EMMA's implementation on top of ONOS, as well as the required interactions between ONOS and the underlying network, are described in Section 2.6. Emulation results and the comparison between EMMA and the optimum solution are presented in Section 2.7. A detailed discussion of previous work and of our novel contribution with respect to that, is provided in Section 2.8. Finally, Section 2.9 draws some conclusions.

2.2 5G-Crosshaul Project

The work in this chapter is done as part of 5G-Crosshaul European Project [13]. 5G-Crosshaul project is planned to provide fronthaul (RAN) and backhaul (packet core) solutions for the ever increasing traffic volume and more diversified traffic requirements in a cost-efficient manner.

5G-Crosshaul project is intended to develop a 5G integrated backhaul and fronthaul transport network enabling a flexible and software-defined reconfiguration of all networking elements in a multi-tenant and service-oriented unified management environment. The 5G-Crosshaul transport network envisioned consists of high-capacity switches and heterogeneous transmission links (e.g., fiber or wireless optics, high-capacity copper, mmWave) interconnecting Remote Radio Heads, 5GPoAs (e.g., macro and small cells), cloud-processing units (mini data centers), and points-of-presence of the core networks of one or multiple service providers. This transport network will flexibly interconnect distributed 5G radio access and core network functions, hosted on in-network cloud nodes, through the implementation of:

- a control infrastructure using a unified, abstract network model for control plane integration (Crosshaul Control Infrastructure, XCI);
- a unified data plane encompassing innovative high-capacity transmission technologies and novel deterministic-latency switch architectures (Crosshaul Packet Forwarding Element, XFE).

2.3 System Model and Problem Formulation

Energy consumption of the backhaul network can be minimized by limiting the number of active links and nodes, i.e., by *(i)* turning off link drivers whenever possible, resulting in proportional (possibly non-linear) changes, and *(ii)* turning off those nodes whose links are inactive.

Both approaches can be studied by building a directed network graph whose vertices represent the network core switches, and edges correspond to links connecting the switches. Let us then consider that the network includes N core switches and L links and denote by $\mathcal{N}$ and $\mathcal{L}$ the set of switches and links, respectively. Let a link $(i, j) \in \mathcal{L}$, with $i, j \in \mathcal{N}$, have a capacity $C(i, j)$ bits/s. Let $\mathcal{F}(t)$ denote the set of flows at time t, with each flow, $f^{\alpha\omega} \in \mathcal{F}(t)$, characterized by the pair of end[1] core switches α and ω, and by QoS constraints that in our case correspond to the required data rate $R(f^{\alpha\omega})$.

Table 2.1: Model notations

N, L	No. of nodes and links	$\mathcal{N}$, $\mathcal{L}$	Set of nodes and links
$(i,j) \in \mathcal{L}$	Link from node i to node j	$C(i,j)$	Capacity of link (i,j)
$\mathcal{F}(t)$	Set of active traffic flows at time t	$f^{sd} \in \mathcal{F}(t)$	Active flow between source s and destination d
$R(f^{sd})$	Rate requirement for flow f^{sd}	$\boldsymbol{\pi}$	Generic path as an ordered sequence of links
P_{idle}	Power consumption of an idle node	$P(i,j,t)$	Power consumption associated with link (i,j) at t
$x_{ij}(t)$	Takes 1 if link (i,j) is on at t, 0 else	$y_i(t)$	Takes 1 if node i is on at t, 0 else
$z_{\boldsymbol{\pi},f^{sd}}(t)$	Takes 1 if f^{sd} is routed through path $\boldsymbol{\pi}$ at t, 0 else	$\tau_{ij}(t)$	Traffic flowing over link (i,j) at t

Let $x_{ij}(t)$ be a binary variable indicating whether link $(i,j) \in \mathcal{L}$ is "on" ($x_{ij}(t) = 1$) or "off" ($x_{ij}(t) = 0$), at time t. Likewise, $y_i(t)$ is a binary variable indicating whether node $i \in \mathcal{N}$ is active at time t ($y_i(t) = 1$) or not ($y_i(t) = 0$). Also, let a path $\boldsymbol{\pi}$ be an ordered sequence of links. We indicate by the binary variable $z_{\boldsymbol{\pi},f^{\alpha\omega}}(t)$ whether flow $f^{\alpha\omega} \in \mathcal{F}(t)$ is routed through path $\boldsymbol{\pi}$ at time t ($z_{\boldsymbol{\pi},f^{\alpha\omega}}(t) = 1$) or not ($z_{\boldsymbol{\pi},f^{\alpha\omega}}(t) = 0$).

We consider that the generic core switch i has zero power consumption when "off", and P_{idle} when "on" but idle. The power consumption associated with a link, (i,j), at time t linearly depends on the traffic that flows over the link. It follows that the total power consumption of an active core switch i is given by:

[1]Since we do not consider edge switches, the end core switches are those where the flow, respectively, starts and ends in the backhaul.

$$P(i,t) = P_{idle} + E_{sw} \sum_{j \in \mathcal{N}, j \neq i} \tau_{ij}(t) \tag{2.3.1}$$

where the second term on the right hand side of the above equation represents the power consumption at i due to traffic switching. In particular, E_{sw} is the energy consumption per bit due to traffic switching and $\tau_{ij}(t)$ is the total traffic (expressed in bit/s) flowing over link (i,j) at time t. We have:

$$\tau_{ij}(t) = \sum_{f^{\alpha\omega} \in \mathcal{F}(t)} \sum_{\boldsymbol{\pi}:(i,j)\in\boldsymbol{\pi}} R(f^{\alpha\omega}) z_{\boldsymbol{\pi},f^{\alpha\omega}}(t) \tag{2.3.2}$$

where $R(f^{\alpha\omega})$ is rate requirement for flow $f^{\alpha\omega}$ and $z_{\boldsymbol{\pi},f^{\alpha\omega}}(t)$ is a binary variable to determine whether flow $f^{\alpha\omega}$ is passes through path $\boldsymbol{\pi}$ at time t.

Note that in Eq. (2.3.1) we accounted only for outgoing links since we consider core switches, which cannot be source or destination of traffic flows. Furthermore, incoming traffic equals outgoing traffic and only one of them should be considered since the switch processes that traffic only once.

Below, we first present the power consumption model we adopt in order to determine realistic values for P_{idle} and E_{sw}, for OpenFlow switches. Then, we formalize the problem under study by using standard optimization.

2.3.1 Power Model

The power consumption of an IP router or an Ethernet switch that is "on" is the sum of the power consumed by its three major subsystems [14]: $P_{ctr} + P_{evn} + P_{data}$, where P_{ctr} accounts for the power needed to manage the switch and the routing functions, P_{evn} is the power consumption of the environmental units (such as fans), and P_{data} indicates the data plane power consumption. The latter can be decomposed into (i) a constant baseline component, and (ii) a traffic load dependent component. In other words, when a switch is powered on but it does not carry any data traffic, it consumes a constant baseline power. When a device is carrying traffic, it consumes additional load-dependent power for header processing, as well as for storing and forwarding the payload across the switch fabric. Combining the power model in [14] with that for OpenFlow switches in [15], we can write P_{idle} as the sum of P_{ctr}, P_{evn} and the base line component of P_{data}, while the load-dependent component of P_{data} is given by:

$$P(i,j,t) = (E_{lookup} + E_{rx} + E_{xfer} + E_{tx})\tau_{ij}(t)\,.$$

In the above expression,

- E_{lookup} is the energy consumed per bit in the *lookup* stage of a switch, which involves searching the Ternary Content-Addressable Memory (TCAM) for the received flow-key and retrieving the forwarding instructions;
- E_{rx} is the energy consumed per bit in the *reception* stage, which involves receiving a packet from the physical media, extracting important fields to build a flow-key and streaming the packet into the input memory system;
- E_{xfer} is the energy consumed per bit in the *xfer* stage, which involves reading a packet from the inbound memory, all of the logic required to initiate a transfer across the fabric, driving the fabric connections and crossbar, as well as writing the packet into the remote outbound memory;
- E_{tx} is the energy consumed per bit in the *transmission* stage, which involves reading a packet from the outbound memory and transmitting it on the physical media.

In the following, we set: $E_{rx} = E_{tx} = 0.2$ nJ/bit, $E_{xfer} = 0.21$ nJ/bit, $E_{lookup} = 0.034$ nJ/bit, and $P_{idle} = 90$ W [16].

2.3.2 Minimum-energy Flow Routing

The problem of energy-efficient flow allocation can be modeled similarly to what done in [17, 18]. However, we stress that, using the accurate power model introduced above, optimal flow routing becomes a non-linear integer problem (see our discussion at the end of this section). Furthermore, in order to achieve the minimum energy expenditure, we aim at minimizing the *instantaneous* power consumption of the network, i.e., the flow routing problem should be solved whenever a new flow starts or an existing flow ends. Typically, this is impractical in real-world networks but our goal here is to set the best performance that could be achieved. We report below the formulation of the optimization problem adapted to our scenario.

Objective. The goal is to minimize the instantaneous power consumption, which is the sum of the power consumption due to the nodes being "on" and to active link drivers:

$$\min \sum_{i\in\mathcal{N}} y_i(t)P_{idle} + \sum_{(i,j)\in\mathcal{L}} x_{ij}(t)P(i,j,t)\,, \tag{2.3.3}$$

where recall that $P(i,j,t)$ linearly depends on $\tau_{ij}(t)$, which in its turn depends on the binary variable $z_{\pi,f^{sd}}(t)$, as reported in Eq. (2.3.2).

Constraints.

- Flow conservation constraint, for any $j \in \mathcal{N}$:

$$\begin{aligned}\sum_{i\in\mathcal{N},i\neq j} \tau_{ij}(t)x_{ij}(t) - \sum_{k\in\mathcal{N},k\neq j} \tau_{jk}(t)x_{jk}(t)\\ = \sum_{f^{sd}\in\mathcal{F}(t):j=d} R(f^{sd}) - \sum_{f^{sd}\in\mathcal{F}(t):j=s} R(f^{sd})\,.\end{aligned} \tag{2.3.4}$$

- The total traffic flowing on a link must not exceed the link capacity:

$$\tau_{ij}(t) \leq C_{ij}\,, \quad \forall (i,j) \in \mathcal{L}\,. \tag{2.3.5}$$

- A link between two nodes, i and j, can be active only if i and j are both active:

$$\sum_{j\in\mathcal{N},j\neq i} [x_{ij}(t) + x_{ji}(t)] \leq My_i(t)\,, \quad \forall i \in \mathcal{N} \tag{2.3.6}$$

  where M is an arbitrary constant s.t. $M \geq 2(N-1)$.

The input parameters of the above problem are the set of nodes, links and traffic flows, along with their characteristics, while the decision variables are: $x_{ij}(t)$, $y_i(t)$, and $z_{\pi,f^{sd}}(t)$. Thus, the problem is an integer non-linear problem, due to the product of $x_{ij}(t)$'s and $z_{\pi,f^{sd}}(t)$'s in the objective function and in the flow-conservation constraints, which appears when $P(i,j,t)$ is expressed as a function of $\tau_{ij}(t)$, hence, of $z_{\pi,f^{sd}}(t)$ (see Eq. (2.3.2)). Also, it is akin to the bin packing problem[2], which is a combinatorial NP-hard problem. Thus, obtaining the optimal problem solution in large-scale scenarios is not viable. Below we propose a heuristic algorithm that has low computational complexity and whose performance results to be very close to the optimum.

[2]In the bin packing problem, objects of different volumes must be packed into a finite number of bins, each of a given capacity, in a way that the number of used bins is minimized.

2.4 EMMA: A Heuristic Approach

In order to design an efficient algorithm to solve the above problem, we first observe that an efficient heuristic for the solution of the bin packing problem is the First Fit algorithm [19, 20]. Given the set of items to be inserted into the bins, the First Fit algorithm processes an item at a time in arbitrary order and attempts to place the item in the first bin that can accommodate it. If no bin is found, it opens a new bin and puts the item in the new bin.

We use the First Fit algorithm and design a heuristic scheme, named Energy Monitoring and Management Application (EMMA), which: (i) monitors the network status, (ii) efficiently allocates traffic flows as they come, and (iii) re-routes the existing flows when necessary and possible. Flows are (re-)routed by EMMA with the aim to minimize the length of the flow path and the energy consumption of the overall network. In particular, upon the arrival of a new flow, EMMA first tries to fit the flow into the current "active network" while meeting the flow traffic requirements. It then turns on other links and/or nodes only if no suitable path is found. Every time a new link and/or node are added to the active network, EMMA looks for a better alternative path for flows that have been (re-)allocated long enough (more than half their expected duration) ago. If a more energy-efficient allocation is found, then a flow is diverted on the new path, provided that its traffic requirements are still met. Note that EMMA differs from the First Fit algorithm since it tries to find a better path for already allocated flows whenever any change in the active topology occurs.

Algorithm 1 New flow allocation

```
Require: Topology, new flow, network power state, traffic load
 1: for each new flow do
 2:     Compute all shortest paths across active topology
 3:     if suitable path is found then
 4:         % if more than one, select one at random
 5:         Allocate the flow
 6:     else
 7:         Compute shortest paths considering the whole network
 8:         if suitable path is found then
 9:             % if more than one, select one at random
10:             Turn on the selected links and nodes that are off
11:             Allocate the flow
12:             installation_time ← current_time
13:             Run Algorithm 2
14:             % It moves previous flows to a better path if any
```

More in detail, the EMMA scheme is composed of two algorithms. Algorithm 1 presents the sequence of actions to be taken whenever a new flow is activated in the network. Input to the algorithm is the network topology and the information on the new flow to be allocated, the power state of the network devices and the traffic crossing every link. Initially, the computation of the possible paths for the incoming flow is done considering the nodes and links that are currently active (namely, the active network) (line 2). Then if there exists a path that meets the flow traffic requirements, the flow can be successfully allocated (lines 3-5). Otherwise, the whole network should be considered and the search for a suitable path repeated (lines 6-7). If a path is found, the links and nodes that need to be added are activated (lines 8-11).

The active topology can approach to the whole topology in high traffic times. Moreover, a path might not be found in this active topology. Due to these possibilities, in the worst case, the flow allocation procedure in EMMA has a complexity of twice that of the shortest path computation complexity. The flow re-routing procedure increases the complexity.

Algorithm 2 states the steps followed during re-routing of existing traffic. This algorithm is run whenever there is a change in the active network topology, i.e., if nodes and/or links become active while finding a path for a new flow. Input to the algorithm are the network topology and the information on the current flows. The algorithm selects all flows that have started or re-allocated at least T_a time ago (line 1). For each flow satisfying the time hysteresis, it computes a path on the current topology starting with the flows with higher rate requirements (lines 2-4). If the cost of the computed path is less than that of the flow current path, it diverts the flow to the new path and updates the flow installation time (lines 5-7). If the process of moving flows to a different path results in some links and/ or nodes being idle, those links and/or nodes are turned off (lines 8-9).

Algorithm 2 Move flows to a better path

```
Require: Topology, information on current flows
1: $\mathcal{S} \leftarrow$ {flows s.t. installation_time $\geq T_a$}
2: Order flows in $\mathcal{S}$ with decreasing rate requirements
3: for each flow in $\mathcal{S}$ do
4:     Search a path on the active topology
5:     if suitable path exists $\wedge$ (new path cost $<$ old path cost) then
6:         Move flow from old to new path
7:         installation_time $\leftarrow$ current_time
8: if there are links and/or nodes no longer carrying traffic then
9:     Turn them off
```

In addition to (re-)allocating paths to flows, EMMA also keeps track of the power usage in the network by computing the power at the FLOW_ADDED and FLOW_REMOVED events sent from nodes. In case of FLOW_REMOVED event generated due to finishing of a flow, nodes and links which are idle will be switched off to save energy.

In the next section we will briefly present the paradigm of SDN, that we will eventually use as a platform to develop and deploy EMMA.

2.5 Software Defined Networking for 5G Mobile Networks

5G is being designed to support huge volume of traffic, a tremendous number of devices and various service categories. On top of these requirements, 5G is expected to improve the energy efficiency by 10 fold. These goals can only be achieved by employing a set of strategies and technologies: SDN being one of the technologies in the forefront. SDN together with NFV are redefining the network architecture to sustainably support this ever growing traffic volume and newly e merging service demands.

Software Defined Networking separates the control logic from data forwarding functions. Moreover, SDN creates a centralized controller which will be in charge of dictating the routing policies to the switches. The switches, on the other hand, take forwarding rules from the controller and forward traffic according to these rules. This approach will open up the network for more flexible management, maintenance and for new protocols and services to be deployed easily and promptly.

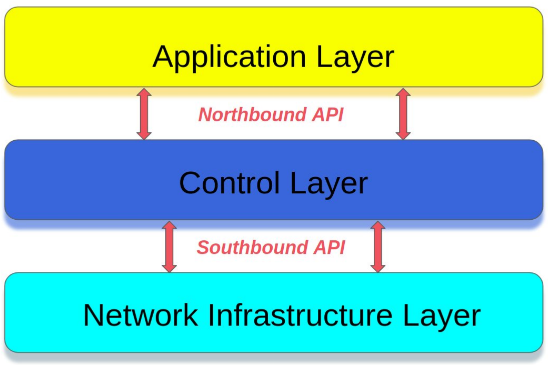

Figure 2.2: Software Defined Networking Architecture.

Figure 2.2 shows a high level architecture of SDN. The SDN controller lies between applications and network devices. The SDN controller has southbound interface to interact with the physical network infrastructure and northbound interface to interact with the applications.

Through its southbound interface, the controller dictates the devices in the physical network infrastructure the policies coming from the applications. The southbound interface is also used to provide the controller with the abstraction model of the physical network. The information provided to the controller include: available links, the functions provided by the device and interface states. Through its northbound interface, the controller learns from the application layer what services the network should provide, and the controller presents the state of the network capability to the applications.

The re-designing of the network architecture with SDN and Network Function Virtualization (NFV) makes the network easily programmable, capable of being shared between different operators by virtualizing the network resources. Moreover, in SDN the control logic is not deployed in a vendor-specific hardware but is made fitting to be deployed on any general purpose hardware. Consequently, the implementation and deployment of routing and other functionalities into the network infrastructure will be easier and faster. We leverage this behavior of the SDN controller in developing our energy-efficient routing application.

In SDN paradigm when a new flow enters into the network, the node encountering the flow checks if it has already a forwarding rule in its forwarding table. If a flow rule is not found, it forwards the flow directly to the SDN controller requesting a path for this flow. The SDN controller computes the path based on the routing application in use. Then, the SDN controller having found the path for this flow, forwards the packet to its next hop based on the path it just computed. Meanwhile, the forwarding rule is sent to every node which is on this path computed for the new flow. It is worth mentioning here that flow rules can also be installed proactively as we will discuss in Section 2.6.

Software Defined Networking makes it possible to manage the entire network through intelligent orchestration and provisioning systems, allowing resources to be allocated on demand. Our energy efficient application is developed based on the premise that the network is easily manageable with SDN to direct new/existing flows on a path the application decides. The routing application is designed in such a way that it uses a minimal number of links and switching nodes to carry the traffic. The links or nodes that are not carrying a traffic will be switched off by the controller to save energy. This is based on an assumption that the controller is able to change the operational state of the network devices.

In our work, we used Open Network Operating System (ONOS) SDN controller.

ONOS [6] is one of the main controllers for software defined networking. It is an open source controller; enables providers to build their network solutions with SDN and NFV paradigms. It is distributed control designed to address scalability. The ONOS controller manages network components, with software programs so as to provide end users and neighboring networks with resources.

2.6 EMMA Implementation in ONOS

We consider a system architecture including:

- a network of OpenFlow switches and links interconnecting them;
- an SDN controller;
- an application on top of the SDN controller that implements EMMA.

The system has to:

- keep track of the set of active network elements, and the set of current traffic flow routes;
- route traffic (i.e., check routing paths for traffic flows and push routes into the network);
- monitor and toggle the power state of the network elements.

In SDN, traffic forwarding rules are created by the controller and are installed into the switches. The flow rule installation could be either proactive, e.g., flows rules are installed into the switches prior to the actual flow arrival, or reactive, e.g., flows rules are created upon every new flow coming into the switches, as discussed in 2.5. In the latter case, the switches at the network edge will need to be always active, in order to receive the traffic that a host may generate. When an edge switch receives a packet for which it has no rule, it directly sends the packet to the controller. Then the controller gets a path from EMMA for such kind of packets and installs a flow rule into the switches. Afterwards, all packets of that kind will be forwarded based on the flow rule just installed.

In the implementation of our algorithms, we followed a reactive approach. The algorithms are tested using the Mininet SDN network emulation environment [7] as forwarding plane, and a controller whose functions are supported by the Open Network Operating System (ONOS) [6]. The algorithm is developed based on ONOS built-in Reactive Forwarding application, which uses the shortest path to route incoming traffic. However, we recall that the topology considered for the shortest path search is not always the whole topology. Indeed, EMMA first tries to find a path on the nodes or links which are already carrying traffic. In this way, EMMA concentrates all the traffic in the active infrastructure whenever possible. If no path is found using the active network, EMMA considers the whole network to route the traffic.

It is worth mentioning that EMMA implements the *PacketProcessor* interface of ONOS, running below it in the architecture, in order to process packets, i.e., we replicated the processing function at the *PacketProcessor* interface of ONOS within EMMA. In this way, EMMA can distinguish between traffic and control packets, as well as between unicast and multicast packets, thus further processing only unicast traffic packets.

Finally, we consider that all network nodes and links are initially "on" and that ONOS provides EMMA with the whole network topology. Specifically, EMMA gets the network topology via the *TopologyService* application of ONOS, to which EMMA has registered. EMMA can then use the network topology for path computation and flow provisioning. If there are idle core switches, EMMA will instruct ONOS to turn them off.

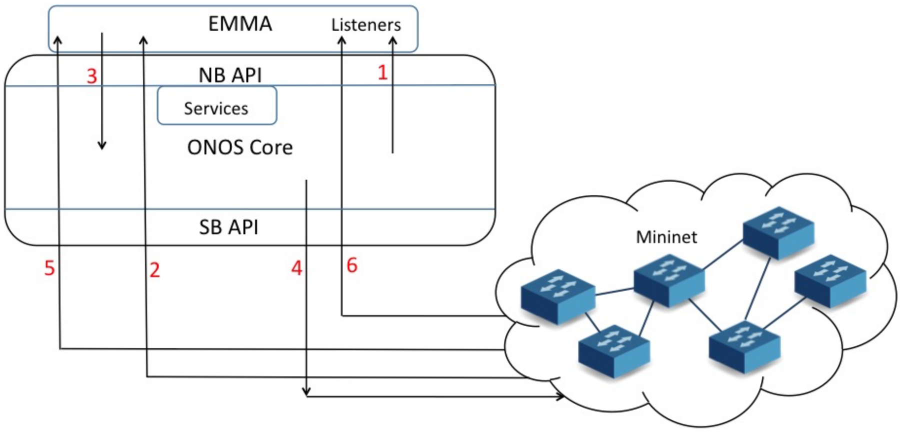

Figure 2.3: Interaction between EMMA and ONOS, and between ONOS and Mininet. The interactions are labeled with numbers corresponding to the sequence of events described in the text.

Figure 2.3 highlights the main components, chain of events and interactions in the implemented system. The description of the set of actions taken by EMMA and the list of events happening in the network are detailed below (the labels associated with the arrows in the figure correspond to the numbers of the listed events or actions).

1. Whenever a switch receives a packet for which it does not have a forwarding rule, it sends the packet of the flow directly to ONOS. EMMA intercepts the packet using the *requestPacket* method of the *PacketService* interface of ONOS.

2. EMMA processes the packet and, if it carries unicast traffic, it computes a flow path according to Algorithm 1. Then it creates a forwarding rule using a set of ONOS applications. Initially, it uses *TrafficSelector.builder*, to form the part of the rule specifying the pair of source and destination IP addresses to be matched when processing the packet at the switch. It then resorts to *TrafficTreatment.builder* in order to express a set of instructions, namely, forwarding the packet toward the intended output port and decrementing the packet TTL. Finally, it invokes *ForwardingObjective.Builder* to create the forwarding rule based on the *TrafficSelector* and *TrafficTreatment* outputs.

3. The ONOS application *FlowObjectiveService* is used to send the rule and install it in the switch.

4. A FLOW_ADDED event will be generated by the switch after the flow rule has been installed. Following the blueprint of the *FlowRuleListener* interface of ONOS, we built a *FlowListener* interface within EMMA. Such an interface allows EMMA to detect a FLOW_ADDED event, after which EMMA starts accounting for the power consumption due to the newly added flow.

5. A FLOW_REMOVED event will be generated by a switch whenever a flow finishes, or the application removes a flow from a switch when a better path is found. As for the FLOW_ADDED event, EMMA is able to detect a flow removal through the *FlowListener* interface we developed and, hence, to correctly compute the energy consumption of the active network. Note that when a FLOW_REMOVED event corresponds to a flow termination (i.e., it has not been triggered by an EMMA re-routing action), EMMA will check whether existing flows can be re-routed, according to Algorithm 2, thus performing a further step similar to step 2).

EMMA will be able to get events related to the topology and flow rules using the topologyListner interface and FlowListner which implements the FlowRuleListener interface. Events, when they are generated in the network will be sent to the

Manager residing in core. Then EMMA which is registered to topology and flow events listener will intercept the events.

We conclude this section by remarking that in our implementation, EMMA computes the active network power consumption based on the packet size and the flow rate, as explained in Section 2.3.1. Such values are obtained as the ratio of, respectively, the number of bytes to the number of packets, and the number of packets to the flow duration. The number of packets, number of bytes and flow duration are provided to EMMA by ONOS. Specifically, EMMA exploits the *getDevice* method of *DeviceService* in ONOS to get the list of available switches, and the *getFlowEntries* method of the *FlowRuleService* of ONOS to get information (i.e., number of bytes, number of packets and flow duration) about each flow handled by a given switch.

Table 2.2: Default settings

Parameter	Value
Flow arrival rate	0.1 flows/s
Average flow duration	20 s
Number of core switches	12
Number of edge switches	half the no. of core switches
Link Capacity	10 MB/s
Hysteresis(T_a)	10 s
P_{idle}	90 W [16]
$P(i,j,t)$	$0.644 \cdot \tau_{i,j}(t)$ nW [16]
Number of hosts per edge switch	10
Link prob. b/w switches	0.5
Packet size	1500 bytes
Experiment duration	500 s

2.7 Experimental Results

We evaluated the EMMA performance against the optimal solution, as well as versus the simple case where no power saving strategy is adopted and the whole network is always active (hereinafter referred to as No Power Saving). The performance of EMMA and of the No Power Saving schemes are obtained by emulation, in the system we implemented and that is described above. The solution of the optimization problem in Eq. (2.3.3) is instead obtained using the Gurobi solver, considering the same network as that emulated in our experiments with Mininet.

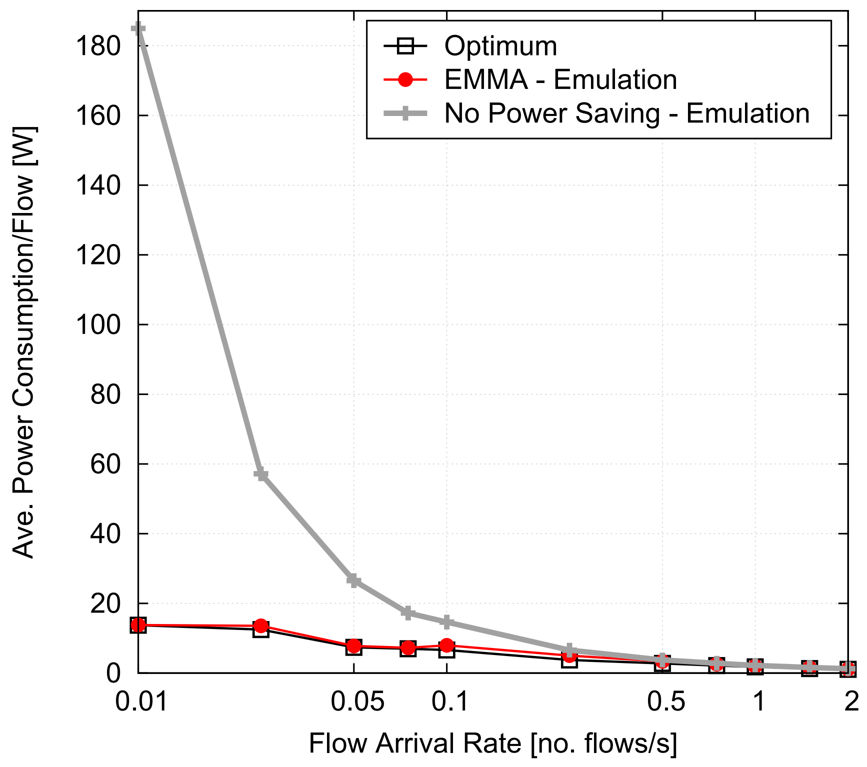

Figure 2.4: Comparing EMMA against the optimum and the No Power Saving scheme: Average power consumption per flow as a function of the flow arrival rate (no. core switches = 12).

We derived the results assuming a default number of core and edge switches equal to 12 and 6, respectively; 10 hosts are connected to each edge switch. Links between any two core switches are set with probability 0.5 and the link capacity is set to 10 Mbytes/s. TCP traffic flows are generated using the Iperf tool, using 1500-byte packets. For each traffic flow, source and destination are selected at random among all possible hosts. Note that this is a worst case assumption for EMMA, while it favours the No Power Saving strategy. The inter-arrival time of newly generated flows follows a negative exponential distribution with a default mean arrival rate of 0.1 flows/s. The traffic flow duration is also exponentially distributed with mean equal to 20 s. The complete list of default values that we adopted for the system parameters is reported in Table 2.2.

In the following figures, we show the average power consumption per flow, as the flow arrival rate and the number of network switches vary. The results have been obtained by averaging over 20 experiments. Note also that power consumption is computed based on traffic statistics and nodes operational states, consistently in all cases.

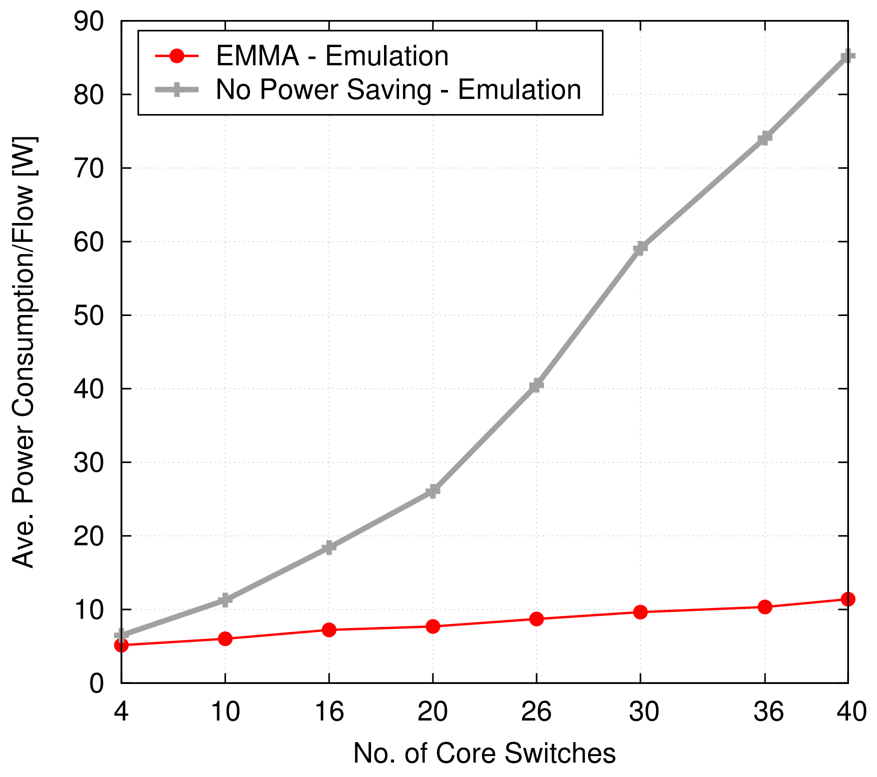

Figure 2.5: Average power consumption per flow vs. number of core switches: comparison between EMMA and No Power Saving (flow arrival rate = 0.1)

Figure 2.4 compares the performance of EMMA to the optimum as well as to that of the No Power Saving scheme, as the flow arrival rate varies and for the default number of core switches (namely, 12). Observe that EMMA matches the optimum very closely, for any value of flow arrival rate. The power saving it provides with respect to the case where all network switches are always on is very noticeable. Clearly, the power gain tends to shrink when many flows have to be allocated (high flow arrival rate), i.e., as an increasing number of switches and links have to be used.

The behavior of EMMA compared to the No Power Saving scheme, as the network size varies, is presented in Figure 2.5. Here we do not show the optimum performance, as we could not solve the optimization problem for a number of core switches significantly larger than the default value. All results are therefore derived by emulation, under a flow arrival rate equal to 0.1. The plot confirms the excellent performance of EMMA: it reduces the power consumption per flow by a factor ranging from 2 (for 10 core switches) to 8 (for 40 core switches). As noted before, a

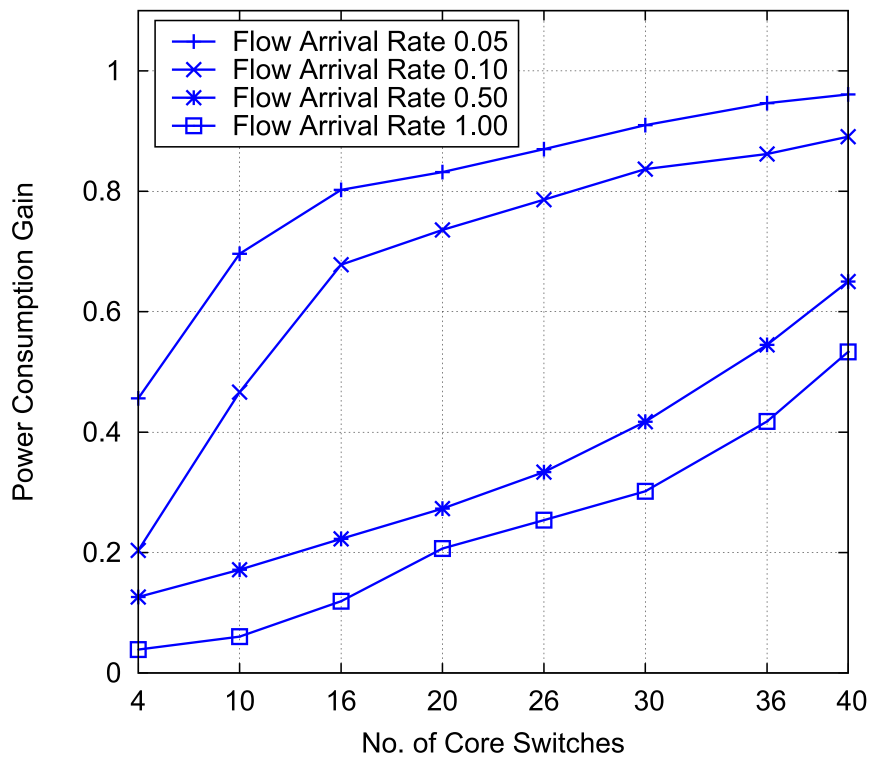

Figure 2.6: Gain in average power consumption per flow (derived by emulation) provided by EMMA with respect to No Power Saving, as the number of core switches and the flow arrival rate vary.

smaller improvement is obtained only when the network size is small compared to the flow arrival rate (e.g.,for 4-5 core switches).

Finally, Figure 2.6 depicts the gain that we can achieve with EMMA with respect to the No Power Saving strategy, as a function of the number of core switches and for a flow arrival rate equal to 0.05, 0.1, 0.5, 1. The gain is computed as the difference in power consumption between No Power Saving and EMMA, normalized to the power consumption of the former scheme. As expected, the gain that EMMA provides is higher for a lower value of flow arrival rate and a larger network size, since it is possible to aggregate more flows on the same links and there are more idle switches that can be turned off. Interestingly, the gain we obtain is always quite high, with peak values that approximate 1. [3]

[3]We would like to note that the energy savings obtained by turning of the nodes is significantly larger than the power saving obtained by turning of link drivers.

2.8 Related Work

Energy-efficient traffic routing in wired networks has been largely addressed in the literature. Here, we limit our discussion to the studies that are most relevant to ours. In particular, at the end of the section we highlight our major contributions with respect to those that are closest to our study.

One of the first works to investigate energy-efficient management of nodes and links in backbone networks, can be found in [17]. The optimization problem they present to obtain an energy-efficient traffic allocation is an integer linear problem ILP, thus a heuristic is proposed too. Their algorithm first turns off nodes with the smallest traffic load and re-routes traffic consequently, then it tries to de-activate links. An opposite approach with respect to [17] is adopted in [21, 22], where the least congested links are turned off first.

In [19], both Virtual Machine (VM) placement and traffic flow routing are optimized so as to turn off as many unneeded network elements as possible. In particular, the authors use traffic-aware VM grouping to partition VMs into a set of VM-groups so that the overall inter-group traffic volume is minimized while the overall intra-group traffic volume is maximized. An approach based on the greedy bin-packing algorithm is proposed to route the traffic, and to put as many network elements as possible into sleep mode. A similar approach is adopted in [23], which focuses on the case where a sudden surge in traffic occurs after an off-peak period, during which most of the nodes have been turned off. The work tries to minimize the number of nodes/links that have to be activated and to reduce service disruption by avoiding turning off links that are critical to guaranteeing network connectivity. In [24], an application called OptiLoop is designed considering the hybrid nature of nodes, i.e., nodes that are enabled with forwarding and computational capabilities and the fact that traffic changes across processing steps in 5G networks. The Application communicates with SDN controller and NFV Orechesterator (NFVO) to control the states of the network node, physical and virtual, and to route traffic. The authors have provided an optimization problem for traffic assignment and activation and deactivation of backhaul and fronthaul nodes to minimize energy consumption of the network whose integer relaxation is implemented in Java.

Relevant to our work are also the studies in [25, 18], which minimize the power consumption of a data center network. In particular, [25] considers a hierarchical topology and proposes a hierarchical energy optimization technique: all edge switches connecting to any source or destination server in the traffic matrix must be “on”. Also, the network is divided into several pod-level subnetworks and a core-level subnetwork, and traffic is reorganized accordingly. [18] instead assumes that each traffic flow can be split over different paths. Interestingly, they have used OpenFlow to collect the flow matrix and port counters, which are used as input to their routing scheme. The work in [26] extends [18] by introducing a monitoring module that collects statistics, such as switch state, link state, active topology and traffic utilization.

Finally, physical characteristics of the links are accounted for in [27, 28]. The former considers network routers connected by multiple physical cables forming one logical bundled link, and it aims at turning off the cables within such links. The problem [27] poses accounts for the bundle size, besides the network topology and the traffic matrix, and it consists in maximizing the spare network capacity by minimizing the sum of loads over all links. Instead, in [28] the focus is on optical links and the minimum-energy traffic allocation is solved taking into account their peculiarities.

We remark that [17, 19] are the closest works to ours, nevertheless the study we present significantly differs from them. In particular, we recall that our problem formulation resembles that in [17], but it accounts for the instantaneous power consumption and for a more realistic model of the nodes power consumption, which changes the nature of the optimization. As far as our heuristic is concerned, we leverage [19] but design an algorithm that, unlike [19], aims to find a better route for all existing flows whenever there is a change in the active topology. In addition, our focus is on the implementation of energy-efficient flow routing: our algorithms are implemented on ONOS, we define and implement the interactions that our application requires between ONOS and the network emulated through Mininet, and we derive emulation results by letting ONOS and Mininet interact.

2.9 Summary

We addressed energy-efficient flow allocation in the 5G backhaul where traffic forwarding rules are created by an SDN controller, which can also turn switches on or off. The adoption of SDN in our solution is also due to its ability to provide global view of the network infrastructure, therefore, enabling easier control of the network and flow (re-)allocation. We first formalized flow allocation by formulating an optimization problem whose complexity, however, results to be unbearable in large-scale scenarios. We therefore used a heuristic approach and developed an application, named EMMA. We implemented EMMA on top of ONOS and derived experimental results by emulating the network through Mininet. The comparison between EMMA and the optimal solution (obtained in a small-scale scenario) showed that the EMMA performance is very close to the optimum. Also, in larger scale scenarios, emulation results highlighted that EMMA can provide a dramatic energy improvement with respect to our benchmark where switches are always on.

Chapter 3

Energy Efficient Virtualization of 5G Edge Network

The work in this chapter has been published in:

- Senay Semu Tadesse, Carla Fabiana Chiasserini, and Francesco Malandrino, “Energy Consumption Measurements in Docker”. In: IEEE Computers, Software, and Applications Conference (COMPSAC). 2017.
- Senay Semu Tadesse Senay Semu Tadesse, Carla Fabiana Chiasserini, and Francesco Malandrino, “Assessing the Power Cost of Virtualization Through Real-world Workloads”. In: IEEE International Symposium on Local and Metropolitan Area Networks (IEEE LANMAN). 2018.
- Senay Semu Tadesse, Carla Fabiana Chiasserini, and Francesco Malandrino, “Characterizing the power cost of virtualization environments”. In: Transactions on Emerging Telecommunication Technologies (ETT) (2018).

3.1 Motivation

Network Function Virtualization (NFV) is a key enabler of next-generation mobile networks. NFV decouples the network functions from proprietary hardware and runs them as a software on computation capable network nodes. These functions, referred to as Virtualized Network Functions (VNFs), are often deployed on virtualization platforms. The virtualization platforms can take two main categories: traditional virtual machines and lighter-weight containers. Virtualization platforms are leveraged to efficiently utilize servers in the edge network.

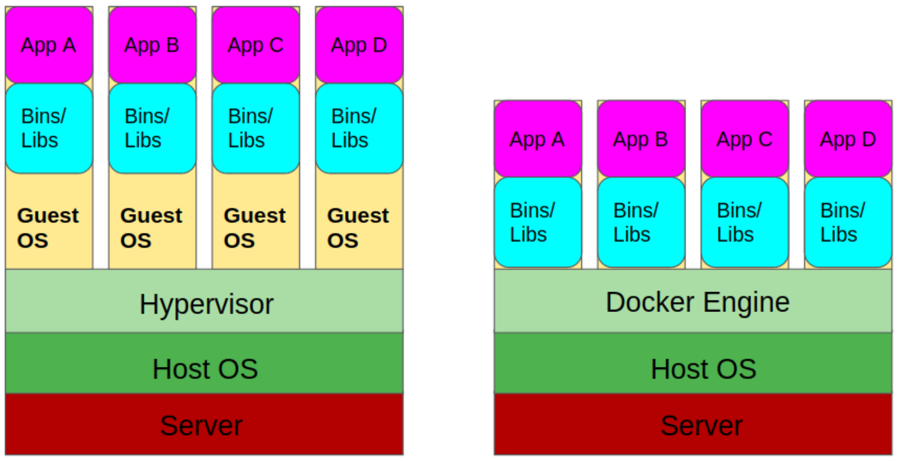

Figure 3.1: High-level architecture of virtual machine-based virtualization (left) and container-based virtualization (right).

One of the most notable differences between next-generation ("5G") mobile networks and present-day ones is the former's ability to serve user request within the network itself. Instead of traveling all the way to an Internet-based server, requests will be processed at computation-capable nodes belonging to the backhaul network itself. This paradigm is known as multi-access edge computing (MEC). Virtualization is a key enabling technology of MEC, as it allows general-purpose network hardware to process requests of any type. At the same time, it is associated with an overhead in terms of CPU usage and memory occupation; such overhead in turn translates into additional power consumption. Power consumption is an increasingly important key performance indicator (KPI) for all types of networks, as it affects their profitability as well as their environmental sustainability. In this context, it is important that the virtualization technique is chosen accounting for the associated power consumption. The traditional approach is virtual machine (VM)-based virtualization: as summarized in Figure 3.1(left), a hypervisor software emulates a whole virtual machine, running the guest operating system and applications. The main advantage of VM-based virtualization is the high level of separation between guests and host: guest machines can have different architecture (e.g., x86 and ARM) and different operating system (e.g., Windows or Linux) from their host; furthermore, the same host can run multiple guests with different architectures and operating systems at the same time. On the negative side, the tasks of emulating guest hardware and running the guest operating system translate into a significant CPU and memory overhead. More recently, container-based virtualization has emerged as a lighter-weight alternative to VMs. As shown in Figure 3.1(right), both the hardware and the operating system kernel are shared between host and guest applications. Isolation is obtained through operating system-level primitives such as namespaces and cgroups, controlling the parts of the execution environment (running processes, daemons...) and system resources visible to guest applications running within containers. The main advantage of container-based virtualization is its lower overhead compared to VMs. Main drawbacks include a lower level of isolation between host and guests (which can lead to security issues) and the impossibility of emulating a guest operating system different from the host one.

In this chapter, we undertake the twofold task of (i) measuring the power consumption associated with VM- and container-based virtualization under a variety of workloads, and (ii) modeling how such consumption depends upon the workload at hand. By fulfilling the first task, we can check to which extent the lower overhead touted by containers translates into lower power consumption. Perhaps more importantly, studying the link between workload, chosen virtualization technique and power consumption allows us to estimate the power consumption associated with a generic workload, beyond those we directly test.

The following are the main novel contributions of our work in this chapter:

- We perform a large set of real-world resource usage and energy consumption measurements, running both synthetic workloads and real-world applications;
- Thereafter, we use these measurements to model the relationship between the resource usage of the hosted application and the power consumption of both virtual machines and containers hosting it;
- We find that containers incur in substantially lower power consumption than virtual machines. Moreover the power consumption for containers increases more slowly than that of virtual machines with the application load.

The rest of this chapter is organized as follows. First we briefly discuss Multi-access Edge Computing and Network Function Virtual in Sections 3.2 and 3.3, respectively. Then, we describe our methodology in Section 3.4, including our testbed and the workloads we use. Then, in Sections 3.5 and 3.6 we summarize the results we obtain from synthetic and real-world workloads respectively. In 3.7, we present the related works and we conclude the chapter in Section 3.8.

3.2 Multi-access Edge Computing

Several services require ultra-low latency and real-time access to radio networks. It has become impossible to satisfy these requirements with the current network architecture where most of the processing is done in the central cloud data centers. Serving all requests of users in the cloud creates two main challenges. First, the delay will be very large to support some service's requirements, and second, it creates congestion in the core of the network.

Multi-access Edge Computing paradigm alleviates this problem by relocating the storage and compute resources near to the user, typically on the base stations. With MEC in place, in addition to the radio processing capability, the base stations are equipped with servers that may act as a cloud in the edge. The server integrated within the base station is a Commercial-of-the-Shelf(COTS) server, providing user applications with computing resources and storage. This will help realize many latency constrained applications.

MEC, in essence, is a highly distributed cloud which is placed in close proximity to the end users. Usually a service requested by a user is composed of multiple functionalities. The functionalities are served in the edge or in the cloud depending on the latency requirements. Moreover, MEC is very useful to ease the load on devices by offloading tasks to the (edge) cloud. Particularly, this is very important for mobile devices characterized by limited storage, power and computation capacity.

MEC servers can be made scalable with the use of virtualization technology. With virtualization, MEC is also made dynamic in its resource usage, i.e., resources can be scaled up or down according to the applications requirements. The dynamic resource usage eventually leads to a better resource management and energy efficiency. The following section briefly discusses Network Function Virtualization, also known as Network Softwarization.

3.3 Network Function Virtualization

Network Function Virtualization (NFV) is making use of virtualization technology to softwarize network functions; so they can be deployed on COTS hardware. Network Function Virtualization creates flexibility in the assignment of resources of the network and servers. NFV is a complementary technology to SDN; it facilitates the network resources to be managed flexibly by SDN controller. With NFV and

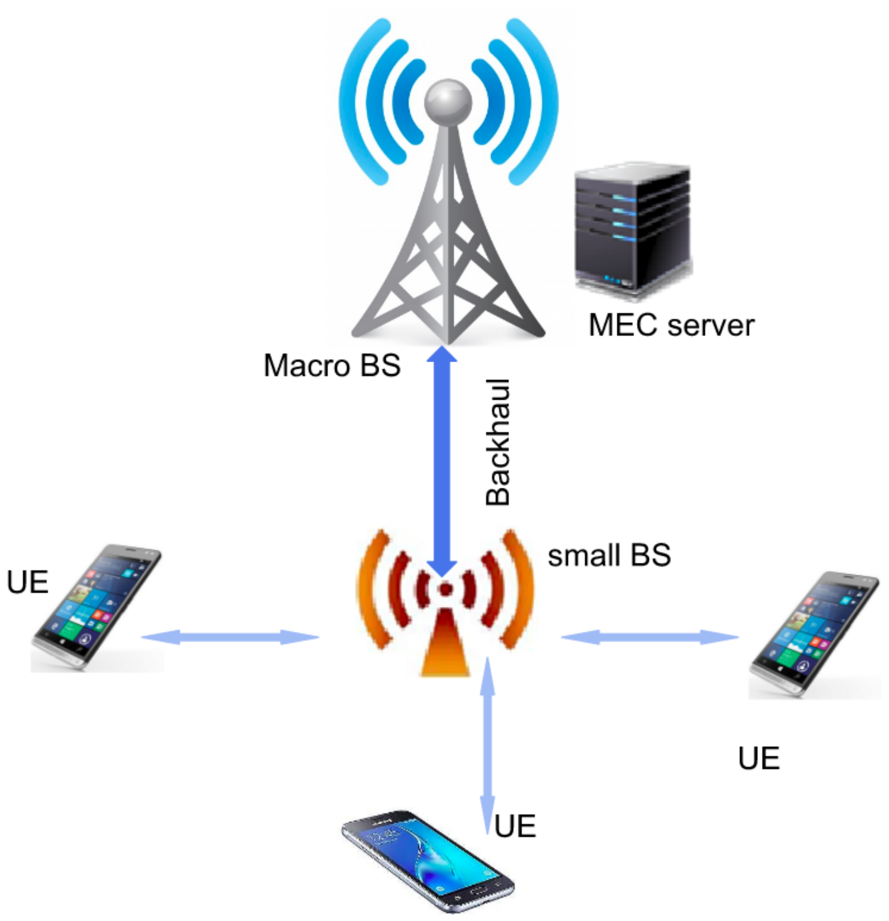

Figure 3.2: Multi-access edge computing

SDN, any network function can be softwarized and deployed in a virtual environment. These network functions can be chained to provide the same service provided by legacy networks. The ability to run the network functions on a virtualization environment with a general purpose hardware reduces operational and investment costs. It reduces deployment time of new services and makes it possible to consolidate various network functions to run in non-proprietary hardware. An example of such network functions, which can benefit from NFV, is the Evolved Packet Core (EPC). The EPC components can be softwarized, as we can see in Chapter 4, and be deployed in virtualization environments.

This flexibility opens up the network for better management. Choosing the node in the network used to deploy the the network functions would no more be constrained by the position of a specific hardware. Indeed, NFVs can be deployed any where in the network where there is a server with the needed computation capability. Moreover, NFVs can be moved from one server to the other to provide optimal service to the end users. In this way, VNFs can be placed near to the end user to reduce latency, or it can be put in the cloud, or may be relocated to stay near to users as the users navigate to different cell sites. Moreover, using NFV has the potential to make the network more energy efficient. This stems from the possibility of assigning resources dynamically according to the applications' requirements and service requests from users.

There are different virtualization technologies to deploy NFVs. In this work, we study the performance of different virtualization technologies in terms of the energy cost they incur, their resource utilization and latency.

3.4 Methodologies Used to Characterize Energy Consumption of Virtualization Technologies

In order to characterize virtualization environments, we measure the power consumption associated with virtualization through a small-scale testbed. Whenever possible, we use commodity hardware and open-source software, so as to make our results as easy to reproduce as possible. In this section, we describe the hardware (3.4.1) and software (3.4.2) we use in our testbed. We will also describe in detail the workloads we take into account, both synthetic and real (3.4.3).

3.4.1 Hardware

The host machine we use for both containers and virtual machines is a HP EliteBook 820 G3 laptop, equipped with a Intel Core i5-6200U processor (two cores, 2.3 GHz, 3 MByte cache) and 8 GByte of RAM. When testing client-server applications, we run the servers (within containers or virtual machines) on the laptop, and the clients on a separate computer, namely, a ThinkCenter M93p desktop. By doing so, we prevent client and server applications from interfering with one another, e.g., triggering thread preemption.

The laptop and desktop are connected through a portable Gigabit Ethernet switch, and that switch is not shared with other computers. This ensures that the connection between clients and servers is consistently fast, and never represents a bottleneck for the application running.

Finally, we measure the power consumed by the laptop through a RCE PM600 power meter, displaying the total power the laptop draws from the grid. By using a power meter instead of estimates from the operating system, we obtain very precise and reliable measurements. However, this also means that special care is needed to tell the power consumption due to the workload apart from other contributions, e.g., the power consumed by the host operating system. To account for this discrepancy:

- we measure the idle power consumption of the computer, before running any workload;
- we subtract such value from the power consumption measured with each workload.

Additionally, we remove the battery from the laptop to ensure that no power is drawn to charge it.

3.4.2 Software

There are two main decisions we need to make concerning the software to use in our testbed: the operating system to use, and which VM- and container-based virtualization solutions to adopt.

The natural choice for the operating system is Linux, due to its broad support for all virtualization technologies. Among Linux distributions we chose Ubuntu, which offers the best balance between up-to-date packages, documentation, and overall system stability; specifically, we use the 16.10 LTS version, with kernel version 4.8.0-46-generic. As both the i5 processor and this kernel support hyper-threading, four threads can be run at the same time.

VM-based virtualization is a mature technology, with several available options, both commercial and open-source. We choose VirtualBox, on the grounds that it is the most popular among open-source ones. Both commercial (most notably, VMWare) and open-source (namely, QEMU) alternatives are sometimes reported to be marginally faster than VirtualBox; however, VirtualBox is more universally available than both (VMWare requires a license, and QEMU which is branded a "processor emulator", is suited for narrower set of scenarios).

Choosing the reference container-based solution is the easiest. Docker is indeed the de facto standard, and arguably the primary reason why container-based virtualization attained its current level of popularity. It is an open-source application, and Linux support is its primary (though not exclusive) focus.

3.4.3 Workloads

We measure the power consumption under the following categories of workloads:

- *no workloads,* idle virtualization environments;
- *synthetic workloads,* consuming a predetermined amount of CPU resources, memory, storage, or network capacity;
- *real-world applications,* whose servers we run into VMs or containers.

By testing the virtualization environments with no workloads, we study how the idle VMs or containers behave as we increase the number of VMs or containers and their impact on the associated overhead. Through synthetic workloads, we are able to establish which among CPU usage, memory and storage consumption, and network traffic has the highest impact on power consumption. Real-world workloads, on the other hand, allow us to get a glimpse of how popular, present-day applications would fare in a virtualized environment, and the power consumption we can expect from them.

Synthetic workloads

Synthetic workloads aim at stressing a specific component of the system, e.g., CPU or memory, while using as little as possible of the other resources. As an example, a good CPU test will consume a predictable and configurable amount of CPU, while at the same time occupying a negligible amount of memory, storing no data on the hard disk, and generating no network traffic. Whenever possible, we use existing, off-the-shelf test applications as our workloads, as described next.

CPU test. We use the Sysbench application, simulating a high CPU load by testing prime numbers. Prime number testing is a CPU-intensive task, which requires little memory and no reading/writing from disk or network. This implies that the Sysbench process will almost never be put to sleep pending I/O operations, thus using (unless preempted by other processes) virtually 100% of a (logical) CPU core. Since Sysbench is a single-threaded application, it will not use multiple cores even if they are available.

Memory test. Our objective here is to read and write a significant amount of memory, while at the same time using the CPU as little as possible. To this end, we write a customized Java application that allocates and then copies a large array, without initializing or writing anything into it.

Network test. We use the iperf program to generate a predetermined quantity of TCP traffic from servers (running within VMs or containers) and clients, running on the host laptop. We keep the clients on the same machine as the servers in order not to use any networking hardware, and only measure the power consumption due to the processing at the hypervisor and kernel.

Disk test. We use the Flexible IO (FIO) application to transfer a 10-GByte file using a configurable block size. By doing so, we are able to adjust the number of disk operations needed for the transfer, where the number of disk operations is inversely related to the block size, and thus estimate their impact on the power consumption.

Real-world workloads

For real-world workloads, we select two types of application that are expected to dominate the traffic of next-generation networks – and, indeed, are regarded to as their very motivation: video streaming and on-line gaming.

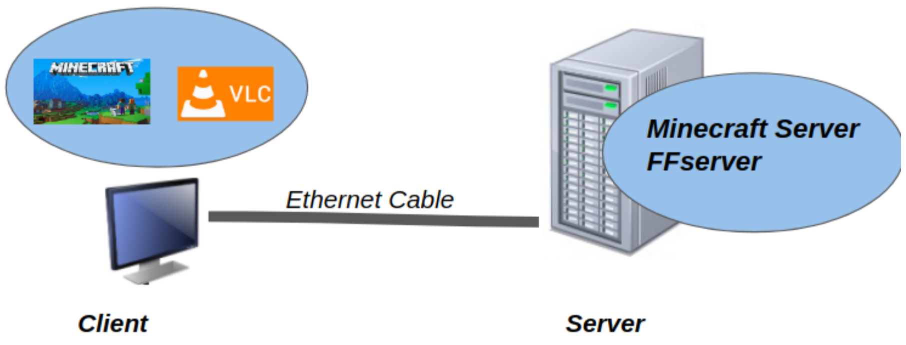

Figure 3.3: Testbed setup for real-world workloads).

Video steaming. We employ the FFServer open-source streaming server, and the VLC free client. A varying number of servers, each running in its own VM or container, stream a video to a varying number of clients, running on the desktop computer as stand alone applications. This setup, depicted in Figure 3.3, allows us to assess how both the number of servers and the number of clients per server affect the power consumption.

On-line gaming. We select the popular on-line game Minecraft, as it offers an open-source server. In selecting the client, we need to ensure that (i) it can run on a headless machine, i.e., without graphical interface, and (ii) we can reproduce multiple times the same input, i.e., the same game. To this end, we combine the Java-based Minecraft client with the xdtool keystroke emulator, both running on the desktop computer.

3.5 Results and Models: Synthetic Workloads

In this section we will discuss the performance of various synthetic workloads.

3.5.1 No Workload

Before we apply any workload, we measure the idle power consumption associated with VirtualBox and Docker. To this end, we create several VMs or containers, leave them idle, and measure the subsequent increase in CPU usage and power consumption.

The results are summarized in Figures 3.4 and 3.5 , and already point at a fundamental difference between VMs and containers. Not only Docker is associated with a lower CPU usage, and thus a lower power consumption, than VirtualBox, but its overhead remains constant as the number of containers increases. The overhead associated with VirtualBox, on the other hand, grows with the number of idle VMs we create: we can identify an offset, due to running the hypervisor, and a linear growth as the number of VMs (and thus the quantity of virtual hardware to emulate) increases. Intuitively, Figures 3.4 and 3.5 also suggests that there is a penalty associated with creating a VM and then leaving it unused, while containers only consume system resources if the application they host is active.

3.5.2 CPU- and Memory-intensive Workloads

We now move to the CPU-intensive workload and show how the number of Sysbench processes we start influences the power consumption.

Figure 3.6 shows that the amount of CPU usage grows linearly with the number of Sysbench processes we start. Recall that Sysbench is a single-threaded application, so if (for example) we launch two Sysbench processes then exactly two CPU cores

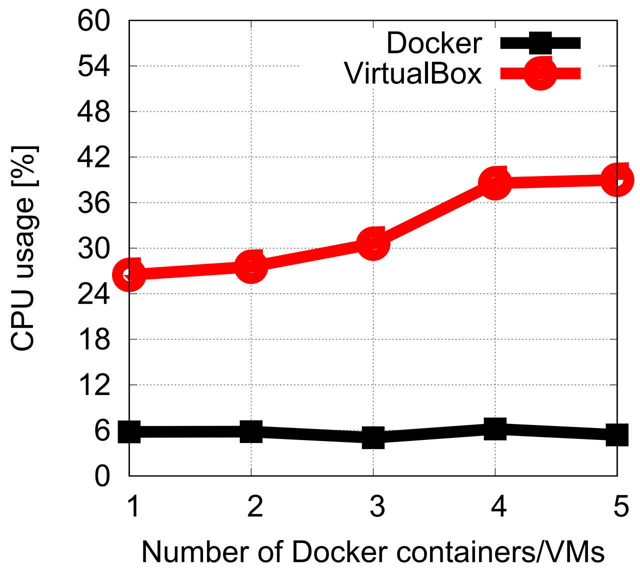

Figure 3.4: No workload: CPU usage as a function of the number of VMs/containers created, for VirtualBox (red lines) and Docker (black lines).

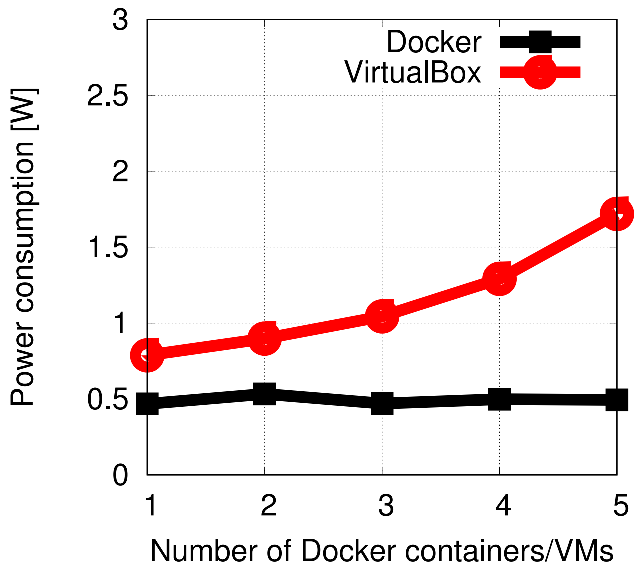

Figure 3.5: Power consumption as a function of the number of VMs/containers created, for VirtualBox (red lines) and Docker (black lines)

will be used. Figure 3.7, more interestingly, shows that the power consumption also grows linearly with the CPU usage, which suggests that CPU usage is the dominant component in this process. One might be surprised to see almost no difference between VirtualBox and Docker curves in Figures 3.6 and 3.7. We observed from Figures 3.4 and 3.5 that Docker has a lower overhead than VirtualBox, so we could expect a lower CPU usage in Figure 3.6 and thus a lower power consumption in Figure 3.7.

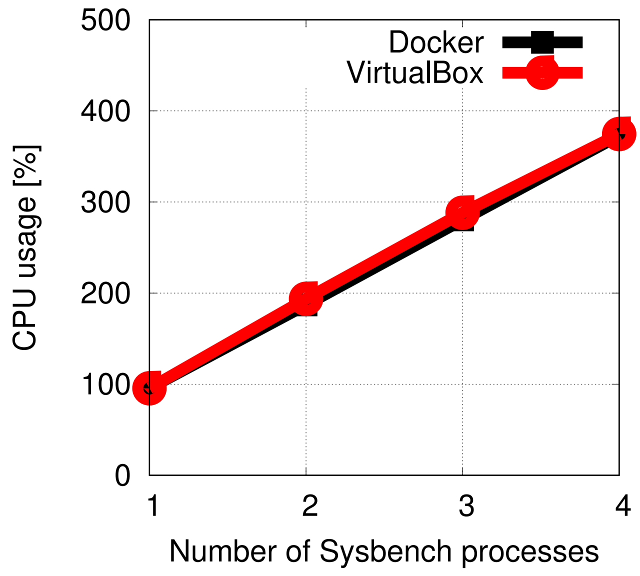

Figure 3.6: CPU-intensive workload: CPU usage as a function of the number of Sysbench processes, for VirtualBox (red lines) and Docker (black lines).)

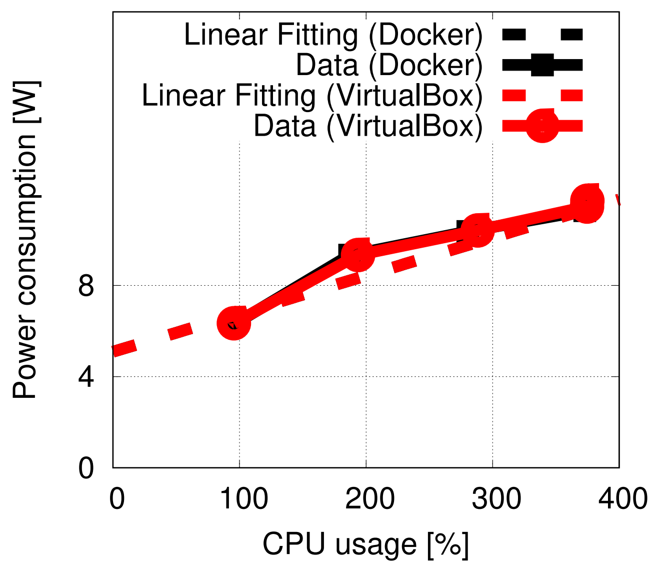

Figure 3.7: CPU-intensive workload: power consumption as a function of the number of Sysbench processes, for VirtualBox (red lines) and Docker (black lines))

The explanation to this apparent paradox is available in Figure 3.7, showing the quantity of work made by Sysbench, i.e., how many numbers it can check: using Docker instead of VirtualBox means that Sysbench is able to do more work in the

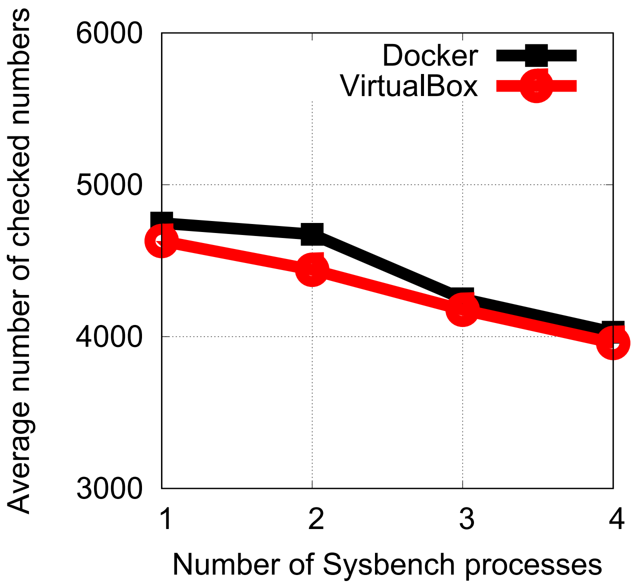

Figure 3.8: CPU-intensive workload: work performed as a function of the number of Sysbench processes, for VirtualBox (red lines) and Docker (black lines))

same time. In summary, the Sysbench test uses the same amount of CPU when VirtualBox and Docker are used; however, in the VirtualBox case, a higher fraction of that CPU is consumed by virtualization overhead, which results in less CPU available for actual work.

Figure 3.9 depicts the power consumption resulting from our Java-based memory test. We can clearly see that the quantity of memory we copy has almost no impact on the total power consumption. Indeed, memory operations require very little intervention from the CPU; Figure 3.9 also confirms our intuition that CPU usage is the main contribution to the total power consumption.

Linear fitting

We use the data summarized in Figure 3.7 to extrapolate a linear model linking CPU usage with power consumption. The fitted relationship for VirtualBox is the following:

$$P_{[W]} = 0.0175c_{[cores]} + 4.9091 \tag{3.5.1}$$

For Docker, we have the following:

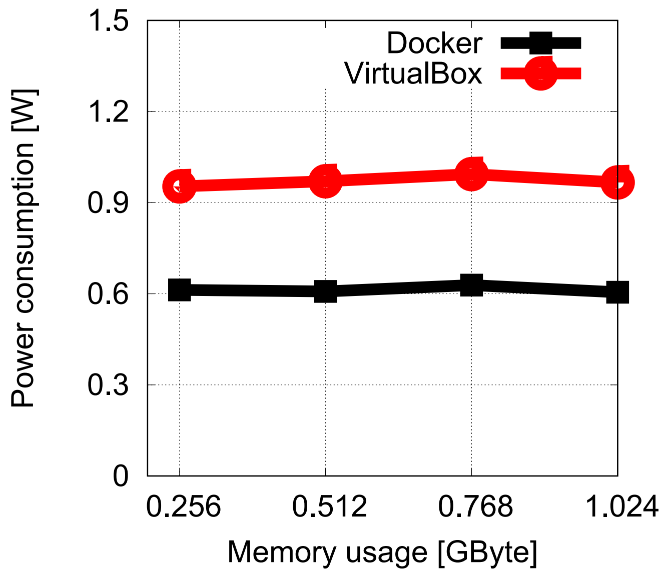

Figure 3.9: Memory-intensive workload: Power consumption as a function of the quantity of memory copied, for VirtualBox (red lines) and Docker (black lines).

$$P_{[W]} = 0.0166c[cores] + 5.0444 \tag{3.5.2}$$

In Equations (3.5.1) and (3.5.2), P is the power consumption (in watt), and c represents the CPU usage in cores. The average fitting error is very low, as little as 0.18% for VirtualBox and 0.04% for Docker. This confirms our intuition that CPU usage is the main driver of power consumption, and the link between them is as simple as a linear relationship.

3.5.3 Network-intensive Workloads

We now use iperf to create a network-intensive workload. Specifically, we use a single server (running in a container or VM) and a single client (running directly on the laptop), and change the offered data rate between 0.01 and 0.9 GBit/s.

Figure 3.10 depicts the power consumption as a function of the offered traffic, and highlights several very important facts. First, VirtualBox exhibits a higher power consumption than Docker, consistently with what we observed earlier. Even more importantly, such power consumption grows much faster for VirtualBox than for Docker, suggesting that containers have not only a better efficiency than virtual machines, but also a better scalability.

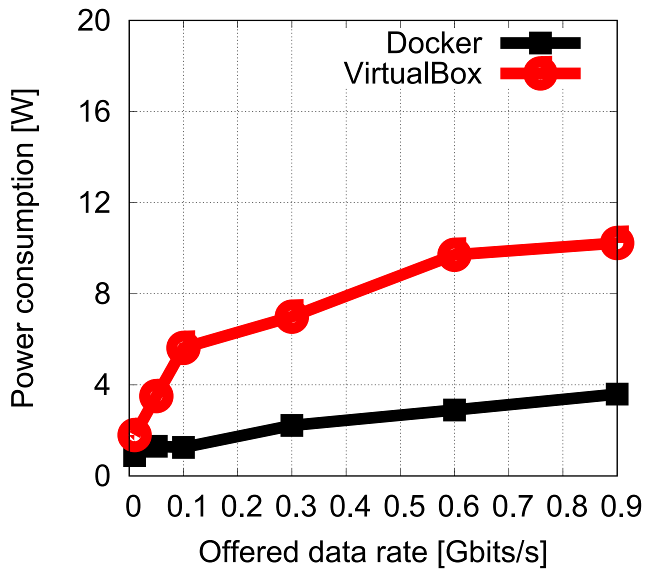

Figure 3.10: Network-intensive workload with one client and one server: power consumption as a function of the offered traffic, for VirtualBox (red lines) and Docker (black lines).

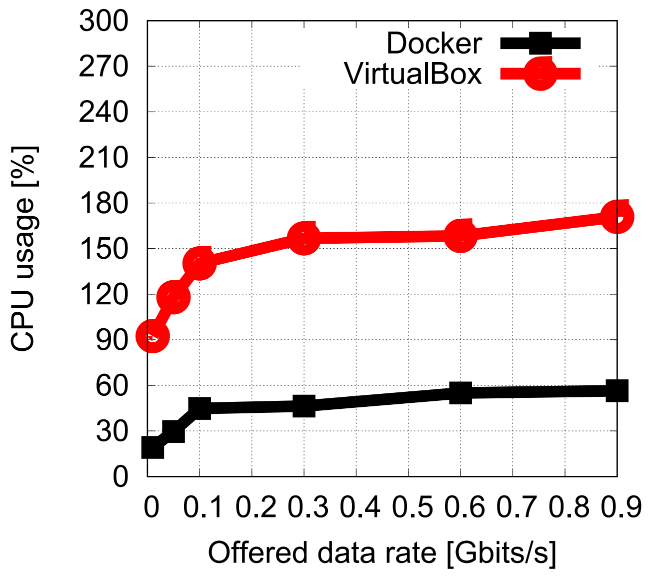

Figure 3.11: Network-intensive workload with one client and one server: CPU usage (b) as a function of the offered traffic, for VirtualBox (red lines) and Docker (black lines).

The reason for this difference lies in the operations needed to transfer packets between the iperf client and server. In Docker, all that is needed is that packets traverse the networking stack of the host operating system's kernel. In VirtualBox, packets need to traverse the host kernel, the hypervisor, and then the guest kernel. As shown in Figure 3.11, all these operations take a substantial toll on the CPU usage, which in turn translates into the higher power consumption we observed in Figure 3.10.

3.5.4 Disk-intensive Workloads

As described earlier, our disk-intensive workload consists in copying a 10-GByte file with different block sizes; the smaller the block size, the larger the number of I/O operations that are required. Figure 3.12 summarizes the power consumption resulting from such workload for different block sizes. We can observe that containers always exhibit a low power consumption, consistently with previous tests. More importantly, the power consumption of Docker is almost constant with respect to the block size, while the one of VirtualBox exhibits a steep increase as the block size drops below 32 kByte. The reason is similar to the one discussed for the network-intensive workload: a disk operation in VirtualBox requires the hypervisor to simulate the behavior of the hardware, which implies substantial CPU overhead and thus higher power consumption.

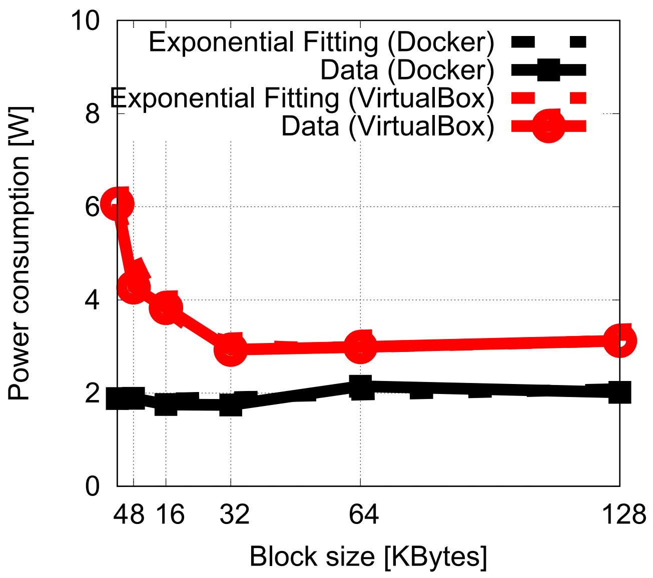

Figure 3.12: Disk-intensive workload: power consumption as a function of the block size, for VirtualBox (red lines) and Docker (black lines).

Figure 3.13 shows time elapsed while performing the disk transfer. We can observe that, for both Docker and VMs, smaller block sizes translate into higher transfer times. However, transfer times with Docker are consistently shorter than

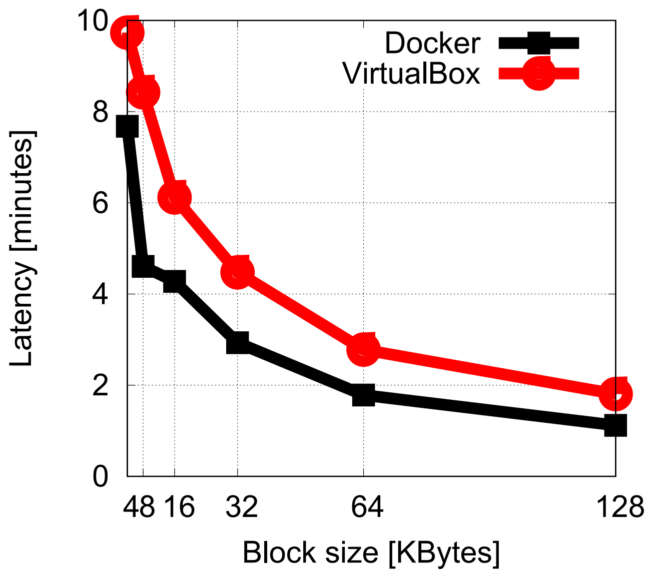

Figure 3.13: Disk-intensive workload: transfer time as a function of the block size, for VirtualBox (red lines) and Docker (black lines).

transfer times with VirtualBox: the higher overhead incurred by virtual machine translates not only in higher power consumption but also in longer transfer times.

Exponential Fitting

We use the data summarized in Figure 3.12 to extrapolate an exponential model linking block size with power consumption. The fitted relationship for VirtualBox is the following:

$$P_{[W]} = 4.81 * exp(-0.1014 * b) + 2.848 * exp(0.00072 * b) kByte \tag{3.5.3}$$

For Docker, we have the following:

$$P_{[W]} = 1.857 * exp(0.0009083 * b) kByte \tag{3.5.4}$$

In Equations (3.5.3) and (3.5.4), P is the power consumption (in watt), and b represents the block size kilobytes. The average fitting error is 11.4% for VirtualBox and 13.9% for Docker. Such figures are higher than those for the linear model described in 3.5; this suggests a fairly complex relationship between block size and power consumption.

3.6 Results: Real-world Workloads

We now move to the real-word workloads described in Subsection 3.4.3, i.e., the FFServer streaming application and the Minecraft game. As depicted in Figure 3.3, we run the servers on our laptop, each in its own VM or container, and the clients on a separate desktop computer.

3.6.1 Video Streaming: FFServer

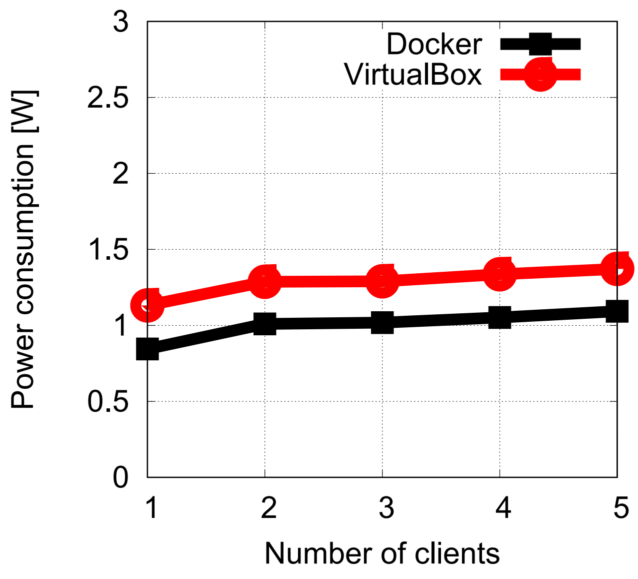

Figure 3.14: FFServer, single-server setup: power consumption as a function of the number of clients, for VirtualBox (red lines) and Docker (black lines).

In our first test, we deploy one FFServer server on our laptop (running within a VM or container) and use it to stream the same video to a varying number of VLC clients. It is important to highlight that we are modeling on demand streaming (à la YouTube) as opposed to real-time streaming (à la AceStream): each client plays an independent stream, and all traffic is unicast.

None the less, as we can see from Figure 3.14, the number of clients only has a small impact on the resources consumed by the server; indeed, neither the CPU usage

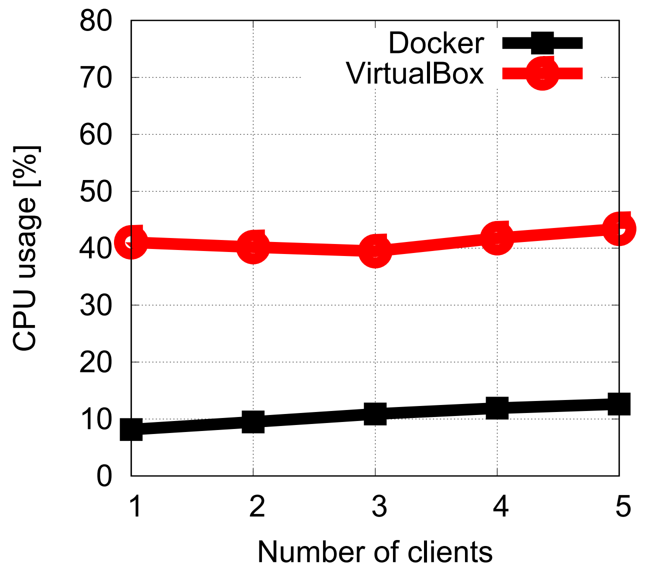

Figure 3.15: FFServer, single-server setup: CPU usage as a function of the number of clients, for VirtualBox (red lines) and Docker (black lines).

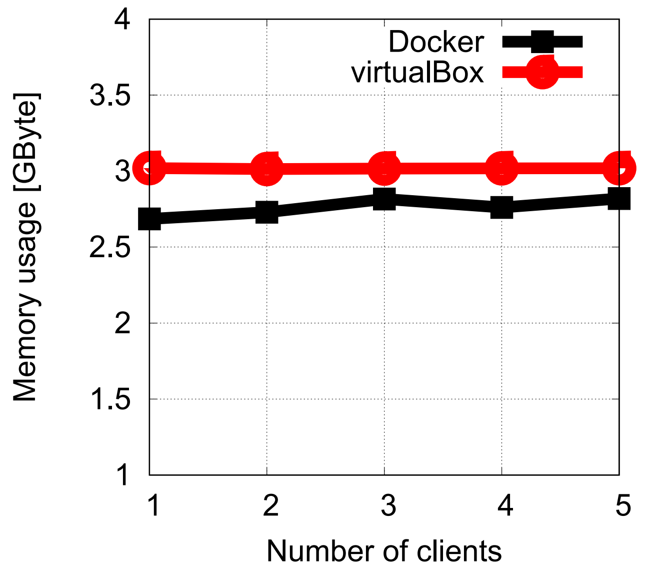

Figure 3.16: FFServer, single-server setup: memory usage as a function of the number of clients, for VirtualBox (red lines) and Docker (black lines).

(Figure 3.15) nor the memory consumption (Figure 3.16)) substantially increases with the number of clients. As a consequence, the power consumption (Figure 3.14) is essentially always the same. This effect is due in large part to FFServer's own scalability. Additionally, the native coding of the video (namely, H.264) was also supported by the client, and therefore no transcoding which is a CPU-intensive operation, was necessary in our case.

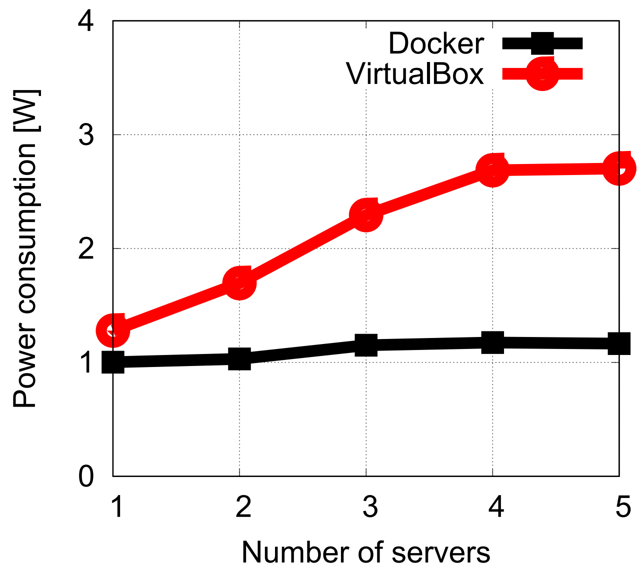

Figure 3.17: FFServer, multiple-server setup: power consumption as a function of the number of clients, for VirtualBox (red lines) and Docker (black lines).

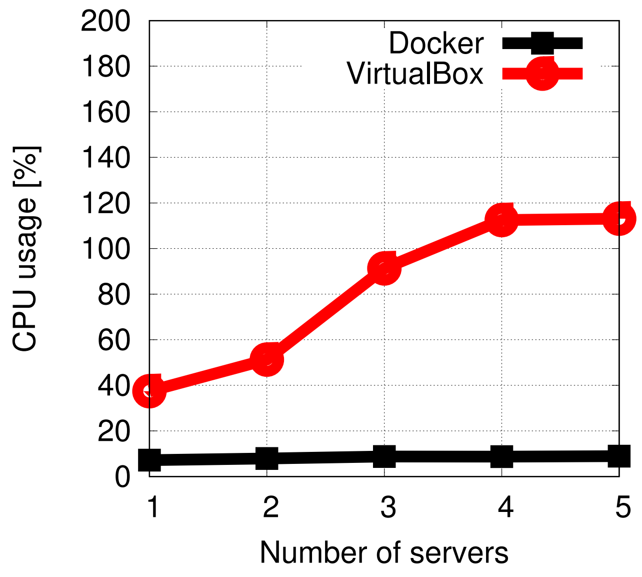

Figure 3.18: FFServer, multiple-server setup: CPU usage as a function of the number of clients, for VirtualBox (red lines) and Docker (black lines).

We now study the effect of having multiple servers running in parallel on the laptop, each in its VM or container, and each serving one client. This is relevant, for example, in MEC scenarios, where virtual (network) functions belonging to different services are often run separately, even if they perform the same task. The corresponding resource consumption is shown in Figures 3.17, 3.18 and 3.19.

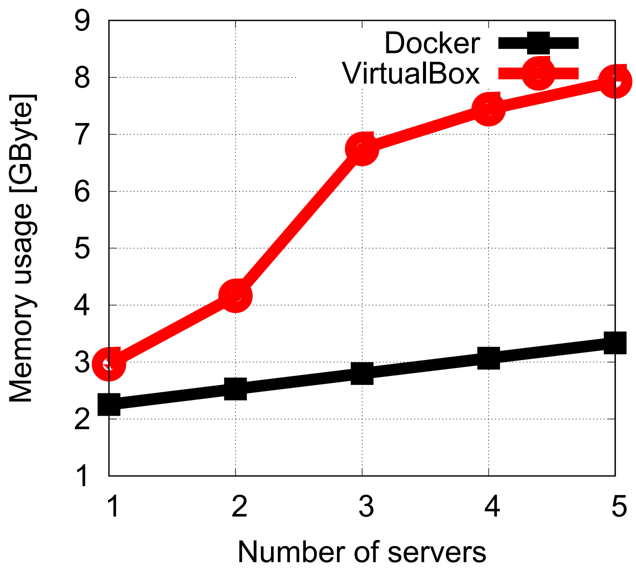

Figure 3.19: FFServer, multiple-server setup: memory usage as a function of the number of clients, for VirtualBox (red lines) and Docker (black lines).

Once more, we observe differences between Docker and VirtualBox that are not only quantitative, but qualitative. With Docker, five servers serving five clients consume approximately the same amount of CPU (Figure 3.18) and power (Figure 3.17) of one server serving five clients; only the memory usage shown in Figure 3.19 grows with the number of servers. Using VirtualBox, on the other hand, means that CPU usage and power consumption grow linearly with the number of servers: this behavior stems from the need to emulate separate virtual hardware and run separate operating systems for each server instance.

We can conclude that the overhead incurred by container-based solutions like Docker is not only lower than that of VMs, but also grows more slowly with the number of containers being run. Such a scalability justifies the interest that container-based virtualization has attracted as a key technology of MEC, and indeed, one of its enablers.

3.6.2 Gaming: Minecraft

We now move to the Minecraft game. To create the same world for all tests we have set the level-seed option in the *server.properties* file on the Minecraft server to a constant. We have chosen a seed that enables the game to last more than five minutes before any player dies. This helps us to collect enough samples of CPU,

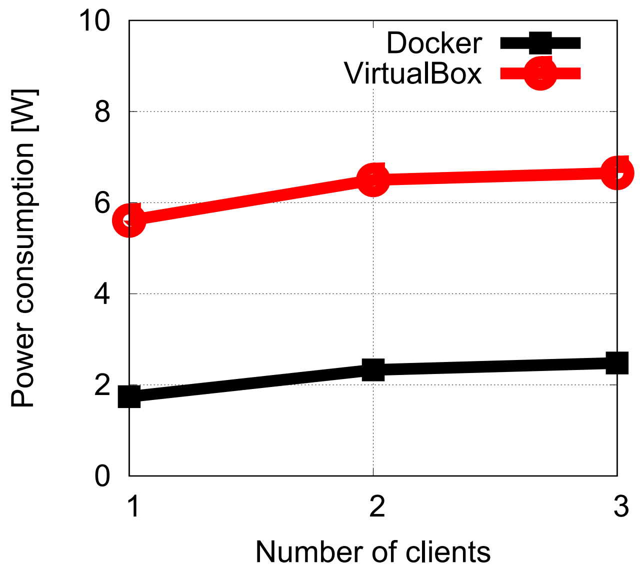

Figure 3.20: Minecraft, single-server setup: power consumption as a function of the number of clients, for VirtualBox (red lines) and Docker (black lines).

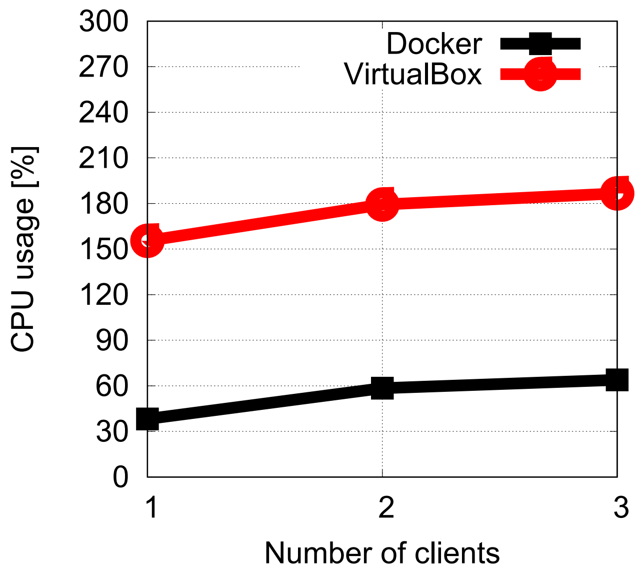

Figure 3.21: Minecraft, single-server setup: CPU usage as a function of the number of clients, for VirtualBox (red lines) and Docker (black lines).

memory and power consumption of the Minecraft server.

Figures 3.20, 3.21 and 3.22 refer to the setup with one server and a varying number of clients, similar to single-server experiment of the video streaming. We can observe a steep increase in the utilization of resources as the number of clients

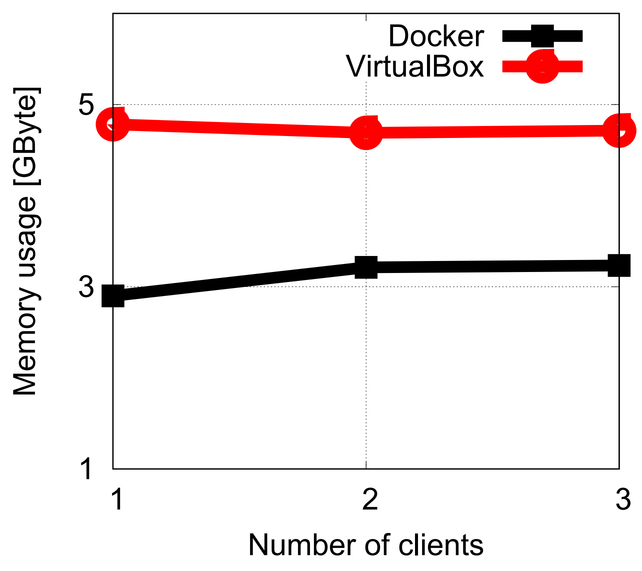

Figure 3.22: Minecraft, single-server setup: memory usage as a function of the number of clients, for VirtualBox (red lines) and Docker (black lines).

increases, for both Docker and VirtualBox; this is due to the different nature of the application being run. Indeed, unlike video servers like FFServer, game servers like Minecraft have to:

- keep a detailed description of the world users are set in, and update it according to the user's actions;
- compute view of the world to serve to the user as it moves;
- transfer the updated view to the user's client.

The two tasks above imply a higher consumption of (respectively) memory and CPU compared to the FFServer case, and therefore a higher power consumption.

As far as the difference between VirtualBox and Docker is concerned, Docker still exhibits a substantially lower power consumption, (Figure 3.20), CPU usage (3.21), and memory usage (Figure 3.22). It is also interesting to notice, from Figure 3.21, how the CPU overhead due to virtualization is higher than the load from the game server itself, intensive as it is.

Figures 3.23, 3.24 and 3.25 summarizes the resource consumption in the multi-server setup, when each user is served by its own server. Similar to multiple-server video streaming tests , we can see a higher resource consumption than in the

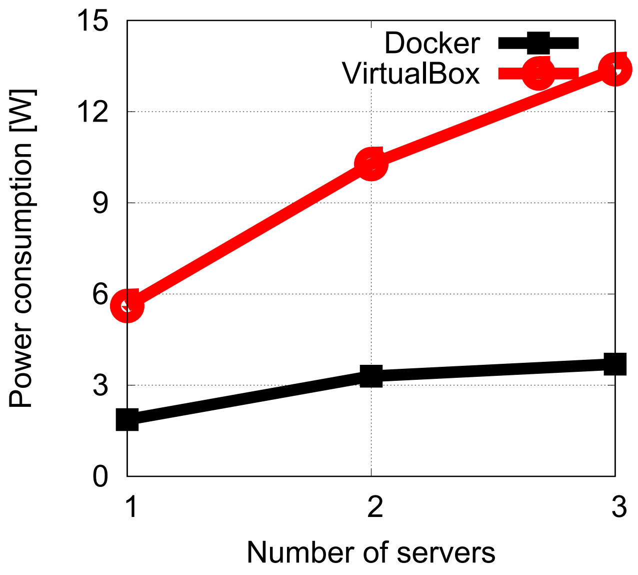

Figure 3.23: Minecraft, multiple-server setup: power consumption as a function of the number of clients, for VirtualBox (red lines) and Docker (black lines).

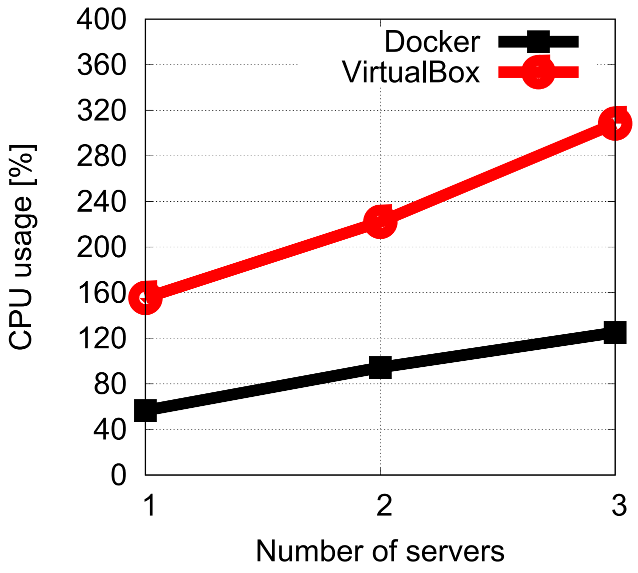

Figure 3.24: Minecraft, multiple-server setup: CPU usage as a function of the number of clients, for VirtualBox (red lines) and Docker (black lines).

single-server setup; furthermore, the qualitative behavior is now the same for both VirtualBox and Docker. The latter has, however, a much lower overhead than the former, suggesting that the scalability advantage of container-based virtualization is present under all types of workloads.

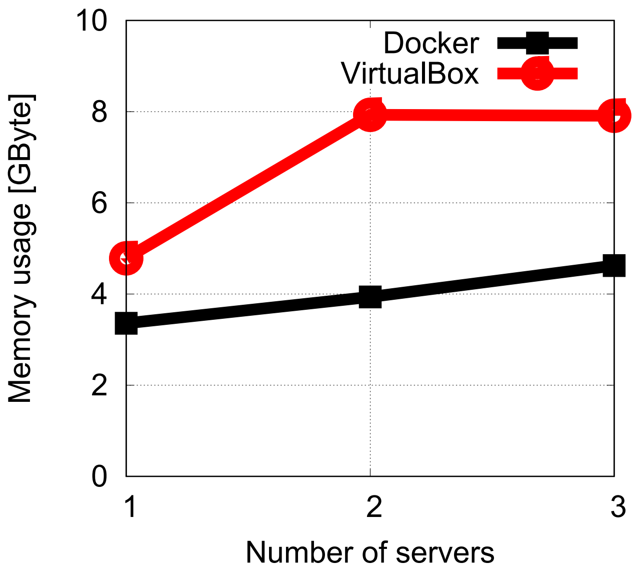

Figure 3.25: Minecraft, multiple-server setup: memory usage as a function of the number of clients, for VirtualBox (red lines) and Docker (black lines).

3.7 Related Work

Measuring the power consumption of virtualization environments and applications running on them is the subject of several existing works. Kansal et al.[29] have developed Joulemeter, a tool to measure the energy consumption of a virtual machine and break it down as the sum of the individual power consumptions of CPU, memory and disk. Krishnan et al. [30] have modeled the power consumption of virtual machines as a linear function of the number of CPU instructions and the number of Last Level Cache (LLC) memory misses.

Another VM power modelling technique is VMeter by Ata E et al. [31] , which is based on online monitoring of CPU, cache, disk and RAM. The model predicts instantaneous power consumption of an individual VM hosted on a physical node in addition to the full system power consumption. Yet another tool to measure power consumption of virtualized applications are presented by Maxime et al. [32]. A fine-grained monitoring middleware is presented in [32], which automatically learns an application-agnostic power model, which can be used to estimate the power consumption of applications. BITWATTS instances use high-throughput communication channels to spread the power consumption across the VM levels and between machines.

Dhiman et al. [33] present a system for online power prediction in virtualized environments based on Gaussian mixture models that use architectural metrics of the physical and virtual machines (VM) collected dynamically by our system to predict both the physical machine and per VM level power consumption. Bertran [34] proposed a system based on CPU and memory power models, relying on Performance Monitoring Counters (PMCs), to perform energy accounting in virtualized systems. Morabito [35] has presented an empirical investigation of different virtualization technologies (including containers) from the viewpoint of power consumption.

Kritwara et al. [36] investigated power consumption and performance issues concerning memory and disk I/O in Xen and KVM vitualization environments. Avino et al. [37] addressed the suitability of Docker in MEC scenarios by quantifying the CPU consumed by Docker when running two different containerized services: multi-player gaming and video streaming. They carried out tests by varying the number of clients and servers for both services. For the gaming service, the overhead logged by Docker increased only with the number of servers; conversely, for the video streaming case, the overhead is not affected by the number of either clients or servers. Fan et al. [19] proposed the Energy driven AvataR migration (EARN) scheme to reduce the total on-grid energy consumption of GCN by considering the energy consumption of Avatar migrations.

3.8 Summary

In this chapter, we set out to assess the additional power consumption due to VM- and container-based virtualization. To this end, we performed an extensive set of real-world measurements, using VirtualBox and Docker as reference technologies and a wide set of workloads, both synthetic and real-world.

Through our measurements, we found that CPU usage is the main driver of the global power consumption, and the extra power consumption of container-based virtualization is not only lower than that of VM-based virtualization, but also grows more slowly as the workload increases. This suggests that container-based virtualization is an attractive technology for MEC scenarios, when large numbers of virtualized applications run on the same physical hardware.

Chapter 4

5G Core Network with IoT Support: Characterization of Control Traffic and Delay

The work in this chapter has been published in:

- Christian Vitale, Carla Fabiana Chiasserini, Francesco Malandrino, and Senay Semu Tadesse, "Characterizing Delay and Control Traffic of Cellular MME with IoT Support". In: IEEE/ACM MobiHoc. 2019.
- Christian Vitale, Carla Fabiana Chiasserini, Francesco Malandrino, and Senay Semu Tadesse, "Characterizing Delay and Control Traffic of Cellular MME with IoT Support", IEEE Transactions on Mobile Computing (outcome of 1st editorial round: Major revision).

4.1 Motivation

The support of massive Internet-of-things (MIoT) represents one of the main use cases of 5G networks, and, due to its peculiar patterns, IoT traffic is given special treatment in the cellular core network. In particular, the bearer instantiation procedure is simplified, and the Mobility Management Entity (MME) serves both control and data traffic. It follows that the performance of the MME is especially critical, and properly scaling its computational capability can determine the ability of the whole network to tackle IoT traffic effectively. This demands for a careful characterization of the MME performance as the IoT traffic load changes, considering virtualized networks and the need for an efficient allocation of computing resources to handle the peculiarity of MIoT traffic.

Massive Internet-of-things (MIoT) is an umbrella term for a fairly diverse set of applications, including smart factory, cloud robotics, automotive leveraging smart city sensors, and surveillance/security; as such, it represents one of the main motivations behind 5G [38]. For all these applications, service latency is a critical constraint, even more so than sheer network throughput. Also, IoT applications are characterized by a very high density of devices, up to 10,000 devices/km^2 [38, p. 6], and peculiar traffic patterns: devices may be inactive for a long time, and then multiple devices may transmit data in a (almost) synchronized manner.

Such traffic patterns are a poor match for the default procedures followed by the Evolved Packet Core (EPC) entities of the cellular network, and such a mismatch may jeopardize the application latency requirements. Indeed, before a terminal has to transmit data packets toward the cellular infrastructure, typically the following operations are required: authentication, identity verification, and bearer establishment. If the terminal remains silent longer than a timeout, the bearer is released and the whole procedure has to be performed again. Thus, for MIoT traffic, bearer instantiation (including bearer establishment and release) is one of the most critical tasks: using the default procedures would result in an exceedingly high latency and control overhead, compared to the data traffic generated by an MIoT device.

To cope with that, 3GPP has introduced a new standard [39], called Control Plane Cellular IoT Evolved Packet System (CIoT) optimization, which is already available in off-the-shelf products [40]. Such a standard (i) simplifies the procedures, roughly halving the associated overhead, (ii) uses the EPC Mobility Management Entity (MME) to forward user-plane traffic, and (iii) limits the involvement of MIoT sources in bearer establishment procedures, hence reducing the power consumption. Importantly, since under the CIoT optimization the MME is in charge of both control- and user-plane processing, it bears the brunt of MIoT traffic, thus becoming the pivotal component of the EPC. It follows that the MME performance and the associated delay determine the ability of the network as a whole to support MIoT traffic.

Ensuring that the MME has sufficient computational capability to efficiently process the traffic load generated by MIoT sources becomes even more sensitive in the context of network *softwarization*. Such a paradigm refers to a global trend towards replacing special-purpose network equipment, including EPC entities [41], with virtualized network functions (VNFs) running on general-purpose hardware. In virtual EPC (vEPC) scenarios, the number of MME instances and their computation capability can be *scaled* to adapt to the variations in the current and expected MIoT traffic they must process. In particular, in the case of the MME, effective scaling requires:

- characterizing the relation between the number of MIoT sources and the arrival rate of bearer requests at the MME;
- modeling the impact of the MME capacity on the delay introduced by the bearer establishment procedure.

In this chapter, we study both the above aspects with reference to the case where a network operating according to the CIoT optimization serves MIoT traffic. Specifically, our main contributions are as follows:

- We begin by characterizing the time between consecutive bearer instantiation requests coming from MIoT sources, finding that it is well described by an exponential distribution;
- Leveraging this key observation, we build an M/D/1-PS queuing model of the MME and study how the bearer instantiation time depends on (a) the traffic load, i.e., the arrival rate of bearer requests, and (b) the computational capability assigned to the MME itself. Importantly, we obtain *closed-form* expressions as a result of our analysis;
- By running and profiling the components of a real-world EPC implementation, we make some fundamental observations on the system that we then exploit to develop our analytical model;
- We validate our analysis through large-scale simulations using both a synthetic traffic model based on the 3GPP standard, and a real-world scenario including topology and mobility information from the city of Monte Carlo, Monaco. Furthermore, we show that the results we obtain represent a powerful tool to drive real-time scaling decisions in softwarized cellular networks.

In this work, my contribution mainly consisted in performing the experimental study, which was used to show some fundamental aspects of the system behavior and was essential to the development of the theoretical analysis, which is included in the following for sake of completeness.

The remainder of the chapter is organized as follows. After introducing our system model in Section 4.2, we describe our main results and provide a roadmap of our analytical derivations in Section 4.3. We present our analysis and a closed-form expression for the characterization of the bearer request arrival process in Section 4.4. Then, in Section 4.5, we run some experiments and make some handful observations to develop our analytical model of the EPC, and we derive the EPC delay performance. Through detailed simulations using both synthetic and real-world traffic, in Section 4.6 we show how our analysis can be used to effectively tune the computational capability of a vEPC. Finally, we review related work in Section 4.7 and conclude the chapter in Section 4.8.

4.2 System Model and Preliminaries

Here, we describe how the EPC serves MIoT traffic when implementing the CIoT bearer instantiation procedure [39]. After introducing the EPC architecture (Subsection 4.2.1), we present in detail the CIoT bearer instantiation procedure and its relevance to the NB-IoT standard (Subsection 4.2.2). Finally, we introduce the model we adopt for the IoT traffic (Subsection 4.2.3).

4.2.1 Evolved Packet Core Network

IoT cellular traffic has to traverse the EPC network, which includes four main components, as depicted in Figure 4.1:

- the Serving Gateway (S-GW) mainly routes data traffic and acts as anchor point when User Equipments (UEs) move from one eNB to another;
- the Packet Data Network (PDN) Gateway (P-GW) acts as ingress and egress point of the mobile access network; it is also the responsible for policy enforcement;
- the Mobility Management Entity (MME) is the termination point of UE control channels. The MME authenticates and tracks registered UEs and, most importantly, it handles bearer activation, i.e., it is the MME that creates a data path between the UEs and the P-GW. When CIoT optimization is in place, the data path between the UE and the P-GW includes the MME itself, since the MME is also responsible for relaying the traffic of the MIoT sources to the correct S-GW (see Figure 4.1);

- the Home Subscriber Server (HSS) is a central database where UE-related information is stored. The HSS assists the MME in UE authentication.

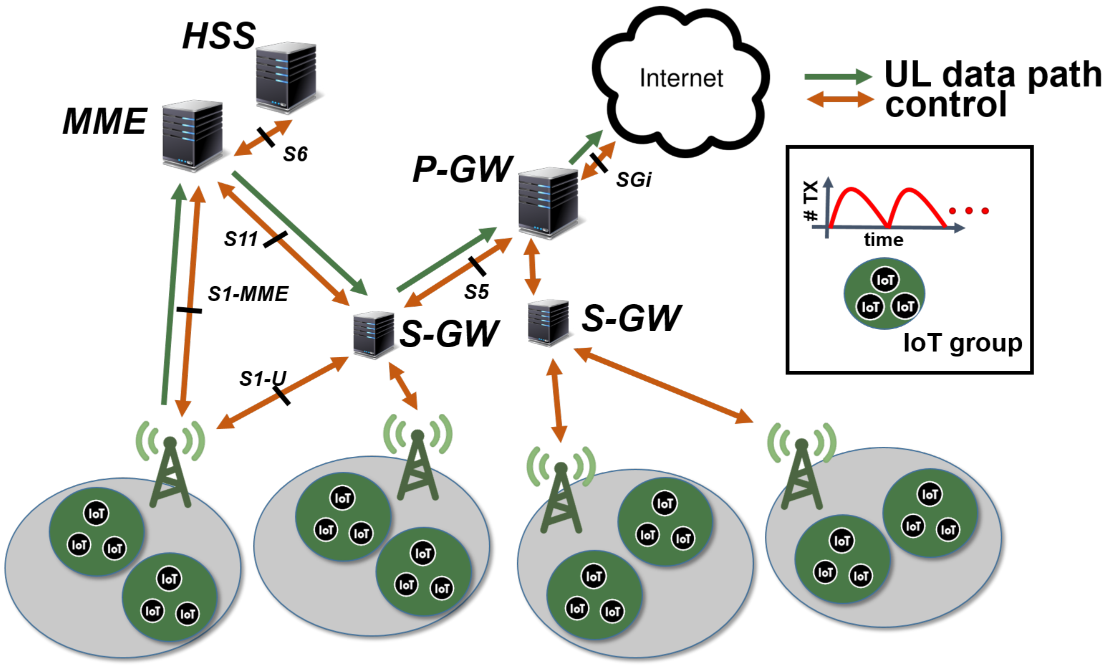

Figure 4.1: EPC architecture.

Note that the MME is connected to the S-GWs for bearer establishment and, under the CIoT optimization, it also performs packet decryption/forwarding, while the P-GW handles the data traffic to/from several S-GWs. Importantly, in the case of a vEPC, the MME, P-GW, and S-GW typically run on different (virtual) machines whose number and capability can be adjusted as needed.

4.2.2 CIoT Bearer Instantiation Procedure

Exactly as any other cellular transmitter, an MIoT source sends or receives data traffic only if a logical connection with the corresponding P-GW is in place, i.e., if the MME has completed the bearer instantiation procedure. However, unlike the ordinary procedure, the CIoT optimization foresees that bearers are released immediately after packet transmission/reception, unless an MIoT source explicitly signals the presence of imminent traffic. As a consequence, an established bearer lasts for quite a short time and no handover procedure is typically required for MIoT traffic at the MME level. In the following, we therefore focus only on the performance of the MME when handling bearer instantiation procedures.

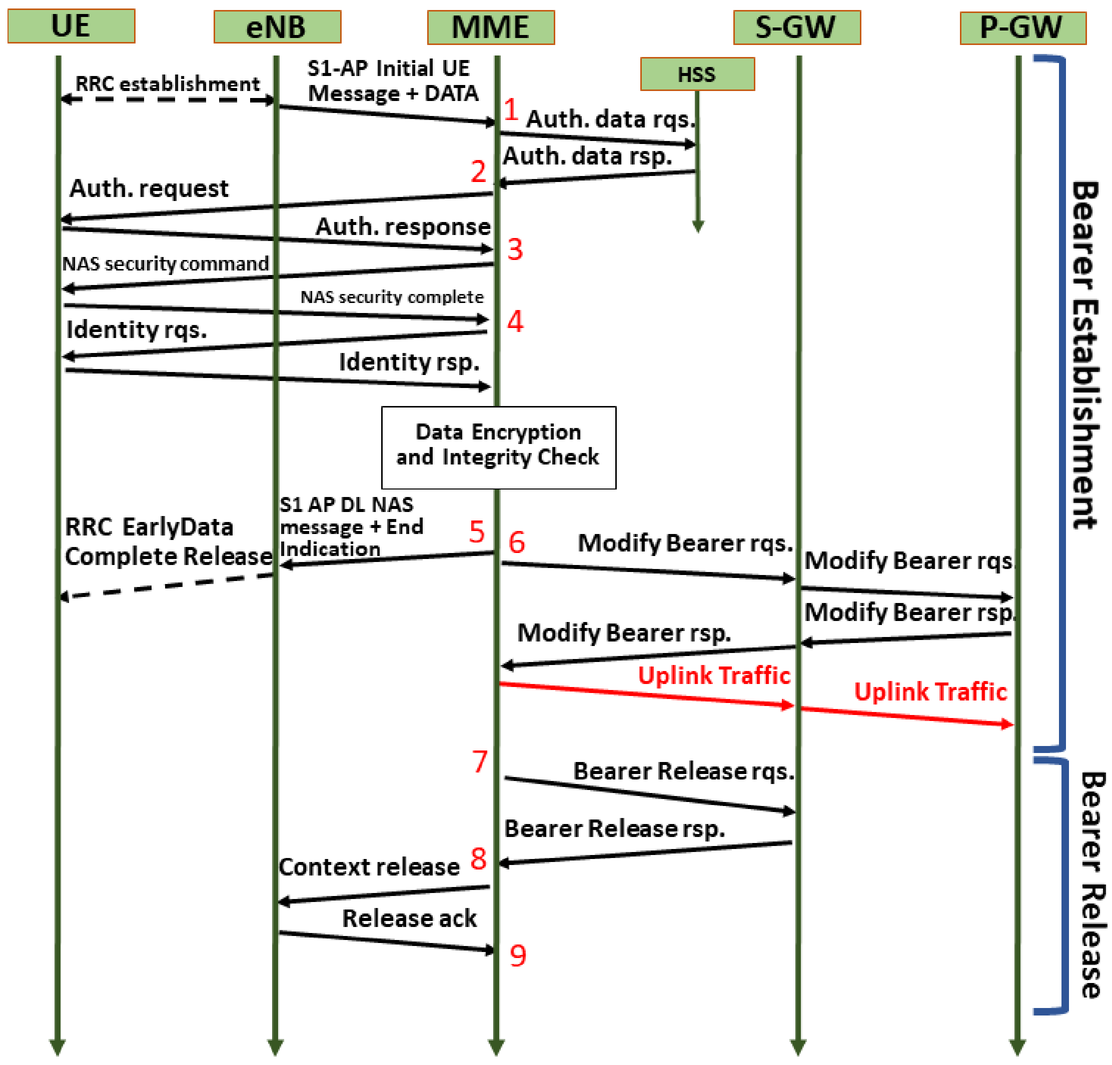

Figure 4.2: CIoT bearer instantiation procedure and uplink data transmission.

As depicted in Figure 4.2, each time an MIoT source has to transmit a packet, five operations are performed: (i) authentication, (ii) identity verification, (iii) bearer establishment, (iv) forwarding (after data decryption and integrity check) of the data packets piggybacked by the MIoT source in the Radio Resource Control (RRC) Early Data Request message, and (v) bearer release. Specifically, hereinafter *bearer establishment* will refer to the set of operations comprised between step 1 and step 6 (included) in Figure 4.2. We remark that such a procedure is a crucial part of data forwarding latency, and it cannot be overlooked in the MIoT data delay computation. Indeed, the time needed to complete a bearer establishment also accounts to the delay incurred by the data transfer within the EPC.

Finally, it is worth remarking that CIoT well pairs up with the Narrowband IoT (NB-IoT) standard, both being specifically designed to support massive IoT traffic, taking, respectively, the core network side and the radio access perspective. Indeed, NB-IoT is a IoT system built from existing LTE functionalities, which aims to support over 50,000 low-data rate stationary devices within a cell-site sector [42]. Beside defining an energy-efficient and robust physical layer for enhanced indoor coverage, NB-IoT also effectively addresses cell search, synchronization, and random access for initial link establishment. Specifically, according to NB-IoT, the RRC establishment on the top-right of Figure 4.2 summarizes the following steps [43]: (a) the UE transmits a random access preamble; (b) the eNB replies with a random access response including a timing advance command and which uplink resources are assigned the UE to perform (c); (c) the UE transmits its identity; (d) the eNB transmits a message to resolve any contention due to multiple UEs accessing the channel (in step (a)) using the same preamble.

4.2.3 IoT Traffic Model

As mentioned above, after data transmission/reception, the MIoT source's bearer is released and a new bearer has to be established if later on the MIoT source has some more traffic to send/receive. Intuitively, depending on the IoT traffic pattern, the time between subsequent data packets may vary significantly, and so does the rate of bearer instantiation requests of an MIoT source. In order to characterize the arrival process of bearer requests, in the following we consider the traffic model described by the 3GPP standard [44] for MTC.

In [44], MIoT sources are organized in groups. Reflecting real-world operation conditions, [44] envisions quasi-synchronous packet transmissions within a group. This represents, for example, a group of sensors monitoring a geographical area, programmed to raise an alarm when a specific event occurs. After the occurrence of the event of interest, e.g., a gas leak, the closest sensors to the event raise the alarm. Sensors neighbouring the area where the event occurred react to the event subsequently, with a delay due to the propagation of the phenomenon. Such an effect triggers alarms from all the sensors belonging to the group, with a peak of alarms (hence, of traffic) roughly at the center of a period and an aggregate traffic distribution over time that follows a Beta(3,4). In [44], the events, and the related group transmissions, occur in subsequent periods of duration T, each of which with an aggregate traffic distribution over time following a Beta(3,4).

Notice that this model is quite general, and setting the group size to 1 allows us to represent MIoT sources behaving independently from each other. Furthermore, as we explain later in Section 4.4, the model can be easily adapted to include data aggregators (a.k.a. gateways) that, as often envisioned in sensor network applications, collect and forward the data packets generated within groups of MIoT devices.

The traffic model specified by the 3GPP standard represents the aggregate behavior of a set of MIoT sources. However, we are interested in characterizing the latency of the data transfers by individual sources, each of which requires a bearer instantiation. In order to address this issue, we leverage the data generation model for an individual IoT device presented in [45], which results in an aggregate group traffic that still fits the Beta(3,4) distribution of the 3GPP standard.

In [45], each MIoT source is modeled as a Markov chain including two states, named *regular operation* and *alarm*, and hereinafter denoted with R and A, respectively. The period T of the IoT traffic pattern is divided into an arbitrary number N of slots, each of duration δ. In state A, the MIoT source sends packets according to a Poisson process with mean λ_A=1 packet/slot, i.e., it transmits at least one data packet with probability $(1 - e^{-1})$. In state R, instead, the MIoT source transmits packets following a Poisson process with mean λ_R=0.0005 $\cdot\, \delta$ packet/slot. Thus, in state R an MIoT source transmits on average 0.0005 packet/s, representing, e.g., keep-alive or synchronization messages.

In each slot n within a period $(n = 1, ..., N)$, the MIoT source may move from one state to the other. When in A, the source moves to R in the next time slot with probability 1. When in R, the source moves to A in time slot n with probability mass function (pmf)[1] $f_b(n)$, which depends on the considered slot in the period. As shown in [45], $f_b(n)$ can be obtained from the sampling of the Beta(3,4) shape[2], as follows:

$$f_b(n) = \text{Beta}\left(\frac{n\delta}{T}\right)\frac{\delta}{T} = 60\left(\frac{n\delta}{T}\right)^2\left(1-\frac{n\delta}{T}\right)^3\frac{\delta}{T}\,. \tag{4.2.1}$$

In summary, given the above IoT traffic model, on average an MIoT source visits state A once every T seconds (with T of the order of seconds [44]), and therein it transmits a packet with probability $(1 - e^{-1})$. Instead, when an MIoT source sojourns in state R, it transmits a packet every 2,000 seconds; for this reason, the global number of transmissions in state R is roughly three orders of magnitude lower

[1]The pmf of a discrete random variable x at n will be denoted by $f_x(n)$. The evaluation at n of the pmf of x conditioned to the random variable y, when $y = m$, will be denoted by $f_x(n|y{=}m)$. Also, we will indicate with $\mathbb{P}(X)$ the probability of a specific event X. The probability density function (pdf) and the cumulative density function (CDF) of a continuous random variable x will be denoted by $f_x(y)$ and $F_x(y)$, respectively.

[2]Note that the Beta(3,4) distribution is only defined in $[0, T]$.

than the number of packets transmitted in state A. Therefore, in the following, we neglect occurrences of a packet transmission in state R.

4.3 Roadmap of the Analysis and Main Results

To evaluate the delay performance of the MME when the CIoT optimization is supported, we proceed as follows.

- We first prove that the arrival process of the bearer instantiation requests at the MME follows a Poisson distribution (Section 4.4). To this end, we need to characterize the distribution of the time interval between subsequent bearer instantiations. Recall that, under the CIoT optimization, every time an MIoT source has a new packet to transmit, the MME has to establish a new bearer and forward the packet to the right S-GW. Thus, the time interval between subsequent bearer instantiations by the MME corresponds to the time interval between packet transmissions by any of the MIoT sources served by the MME. To compute the distribution of the inter-arrival time between bearer instantiation requests, we first derive the distribution of the time interval between a packet transmission by any source in the system and the subsequent visit to state A by any, potentially different, MIoT source. For $\delta \to 0$, we prove that such a distribution does not depend on the time of the last transmission in the system and turns out to be exponential. Furthermore, the result holds also for the time interval between subsequent packet transmissions, i.e., the inter-arrival time of bearer requests at the MME.
- Then, through experimental measurements, we reveal three important findings: (i) the MME is the main bottleneck in the control plane of the EPC, thus dominating the latency introduced by the EPC; (ii) each bearer establishment procedure takes a fixed and *deterministic* amount of processing at the EPC entities; (iii) it is fair to assume that the MME uses a Processor Sharing (PS) serving policy.
- Using our analytical and experimental results, we model the MME as an M/D/1-PS queue and obtain the distribution of the latency of bearer instantiation and of data forwarding at the MME (Section 4.5.2). We also characterize the control overhead at the MME, again under CIoT optimization. We remark that, as shown in Section 4.6, all our findings are demonstrated through an experimental testbed and validated through simulations using real-world data traces.

The notations we use in the following analysis are summarized in Table 4.1; we also highlight that we often use the term “packet transmission” interchangeably with “bearer request”.

Table 4.1: Table of notations

Symbol	Variable
δ	slot duration
T	time between events monitored by MIoT groups
N	number of slots in a period T
Q	number of MIoT sources served by the EPC
$f_b(n)$ $(f_b(n))$	probability of transition from R to A in slot n (time t)
β	time between bearer requests at the EPC
s	slot (time) of the last bearer request
s_q	slot (time) of the last bearer request by source q
α	time between the last bearer request and the next transition to A by any source
α_q	time between the last bearer request and the next transition to A by source q
ω_q	offset of the time reference of source q with respect to source 0
$E(z,\lambda_\alpha)$	Erlang CDF with shape z and rate λ_α
λ_x	rate of the exponential random variable x
O_X	number of CPU operations per bearer procedure for EPC entity X
C_X	capacity, in CPU operations/s, of entity X
d	time between a bearer request and its completion,
v	delay due to the MME of the bearer establishment procedure
K	constant delay due to all EPC entities, other than the MME, in bearer establishment

4.4 IoT Control Traffic Characterization

To derive $F_\beta(\cdot)$, i.e., the cumulative distribution function (CDF) of the time interval between subsequent bearer instantiation requests at the MME, we consider Q MIoT sources served by the same MME, generating traffic according to the 3GPP model described in Section 4.2.3.

As the first step, we fix to k the time slot at which the last transmission in the system occurred and we compute $f_\alpha(m|s{=}k)$, i.e., the probability density function (pdf) of the time interval between k and the slot in which the first device, among the Q MIoT sources, moves to state A.

It is easy to see that $f_\alpha(m|s{=}k)$ can be written as the minimum over the time intervals between k and the first visit to A of the Q MIoT sources, i.e.,

$$f_\alpha(m|s{=}k) = f_{\min(\alpha_q)}(m|s{=}k)\,. \tag{4.4.1}$$

In the above expression, α_q is the time interval between k and the transition to state A of the MIoT source q, and $f_{\min(\alpha_q)}(\cdot)$ is the pdf of the minimum over the α_q's.

Using (4.4.1) and considering the minimum among random variables, the CDF, $F_\alpha(m|s{=}k)$, can be obtained as:

$$F_\alpha(m|s{=}k) = 1-\prod_{q=1}^{Q}\Big(1-F_{\alpha_q}(m|s{=}k)\Big)\,. \tag{4.4.2}$$

From (4.2.1), we observe that an MIoT source moves from state R to state A with a probability that depends neither on the past, nor on the activity of other MIoT sources (not even those belonging to the same IoT group). It follows that the IoT traffic model in [44] is *memory-less*, which greatly simplifies the subsequent derivations. As a first consequence, in the proposition below we prove that (4.4.2) can be computed as if all MIoT sources transmitted a packet in k, i.e., denoting with s_q the slot of the last packet transmission by q, $F_{\alpha_q}(m|s{=}k) = F_{\alpha_q}(m|s_q{=}k)$, $\forall q$.

Proposition 1. *Denote with s the variable representing the slot in which the last packet transmission in the system by any of the MIoT sources occurred, and with α_q the time interval between slot s and the subsequent transition of the q-th MIoT source to state A. Since the IoT traffic is memory-less, α_q can be computed as if the last packet transmission in the system was by q. Denoted with s_q the slot of the last packet transmission by q, for a sufficiently small δ, we get:*

$$F_\alpha(m|s{=}k) = 1-\prod_{q=1}^{Q}\Big(1-F_{\alpha_q}(m|s_q{=}k)\Big)\,. \tag{4.4.3}$$

Proof. The probability $f_{\alpha_q}(m|s{=}k)$ that source q visits state A, m slots after k can be computed as the multiplication of the probabilities of the following events:

- no transition into state A for $m{-}1$ slots;

- a transition into state A, exactly m slots after k.

If instead the last packet in the system was transmitted by q itself, $f_{\alpha_q}(m|s{=}k)$ can be computed as the sum of the probabilities of the following events:

- a transition from A to R (with probability 1) in slot $k+1$;
- no transition into state A for $m{-}2$ slots;
- a transition into state A, exactly m slots after k.

The difference in the two cases is only the transition from R to A in slot $k{+}1$. Such a difference can be neglected if δ is small, since in this case the number of slots between a packet transmission in the system and the subsequent transition to A by any MIoT source is very large. Then, passing to the CDF, we get:

$$\begin{aligned} F_\alpha(m|s{=}k) &= 1{-}\prod_{q=1}^{Q}\left(1{-}F_{\alpha_q}(m|s{=}k)\right) \\ &= 1{-}\prod_{q=1}^{Q}\left(1{-}F_{\alpha_q}(m|s_q{=}k)\right). \end{aligned}$$

□

The above proposition tells us that $F_\alpha(m|s{=}k)$ can be derived by analyzing the dynamics of the individual MIoT sources separately, i.e., through the CDF of the time interval between the last transmission by q and the subsequent visit to state A of q itself, which is significantly easier to compute than using $F_{\alpha_q}(m|s{=}k)$.

Next, we rewrite $F_{\alpha_q}(m|s_q{=}k)$ accounting for the time reference of source q. To this end, we recall that each source belongs to a specific group and it is quasi-synchronized only with the IoT sources belonging to that group, while different groups may exhibit a temporal offset with respect to each other [3]. By taking as global reference the time of source 0, we denote with $\omega_q \in \{0, ..., N{-}1\}$ the time offset between source 0 and the q-th source ($q = 1, \ldots, Q-1$). Then (4.4.3) can be rewritten as:

$$F_\alpha(m|s{=}k) = 1{-}\prod_{q=1}^{Q}\left(1{-}F_{\alpha_q}(m|s_q(k,\omega_q))\right), \tag{4.4.4}$$

[3] Sources belonging to the same group have zero offset relatively to each other.

where $s_q(k,\omega_q)$=mod$(k+\omega_q,N)$ and

$$\begin{aligned} F_{\alpha_q}(m|s_q(k,\omega_q)) &= \sum_{x=1}^{m} f_b(\text{mod}(s_q(k,\omega_q)+x,N)) \cdot \\ & \prod_{y=s_q(k,\omega_q)+1}^{s_q(k,\omega_q)+x-1} [1-f_b(\text{mod}(y,N))] , \end{aligned} \tag{4.4.5}$$

with $f_b(n)$ being the transition probability from R to A given in (4.2.1). In (4.4.5), $F_{\alpha_q}(m|s_q(k,\omega_q))$ has been derived considering the probability that the following sequence of events takes place: no transition into state A for $m-1$ slots, and a transition into state A, exactly m slots after $s_q(k,\omega_q)$.

We now switch to continuous time and evaluate the system dynamics when the slot duration δ tends to 0. Let us denote with t the reference time of MIoT source 0, with $s_q(t,\omega_q)$ the time instant of MIoT source q in its period corresponding to t, i.e., $s_q(t,\omega_q)$=mod$(t+\omega_q,T)$, and with τ representing the interval from the last transmission in the system to the time of the first transition from R to A by any of the MIoT sources.

First, for $\delta \to 0$, we rewrite (4.4.4) and (4.4.5), respectively, as,

$$F_\alpha(\tau|s{=}t) = 1-\prod_{q=1}^{Q} \left(1-F_{\alpha_q}(\tau|s_q(t,\omega_q))\right) \tag{4.4.6}$$

and

$$\begin{aligned} F_{\alpha_q}(\tau|s_q(t,\omega_q)) &= \int_0^\tau f_b\left(\text{mod}(s_q(t,\omega_q)+x,T)\right) \cdot \\ & \prod_{y=s_q(t,\omega_q)}^{s_q(t,\omega_q)+x} (1-f_b(\text{mod}(y,T))\, dx, \end{aligned} \tag{4.4.7}$$

where $f_b(x)$ can be obtained directly from (4.2.1) as,

$$f_b(x) = \frac{60\left(\frac{x}{T}\right)^2\left(1-\frac{x}{T}\right)^3}{T} \tag{4.4.8}$$

Looking at (4.4.6), one can see that, when the number of MIoT sources in the system grows, the time interval between a packet transmission and the subsequent visit to state A by any MIoT source decreases dramatically, since the minimum over a large number of positive random variables should be considered. Consequently, it is enough to provide an expression for $F_{\alpha_q}(\tau|s_q(t,\omega_q))$ that is accurate for small values of τ; given that, we can assume: $s_q(t,\omega_q)+\tau < T, \forall\, s_q(t,\omega_q) \in [0,...,T]$. Then a good approximation of $F_{\alpha_q}(\tau|s_q(t,\omega_q))$ for Q large, hence τ small, is given by:

$$F_{\alpha_q}(\tau|s_q(t,\omega_q)) = \int_0^\tau f_b(s_q(t,\omega_q)+x) \prod_{y=s_q(t,\omega_q)}^{s_q(t,\omega_q)+x} (1-f_b(y))\, dx\,. \tag{4.4.9}$$

Interestingly, the product form in (4.4.9) is the Volterra's product integral. Using such an integral expression in (4.4.9), we obtain:

$$F_{\alpha_q}(\tau|s_q(t,\omega_q)) = \int_0^\tau f_b\left(s_q(t,\omega_q)+x\right) e^{\int_{s_q(t,\omega_q)}^{s_q(t,\omega_q)+x} -f_b(y)dy} dx. \tag{4.4.10}$$

Replacing (4.4.8) in (4.4.10) and solving both integrals, we get:

$$\begin{aligned} F_{\alpha_q}(\tau|s_q(t,\omega_q)) &= 1 - e^{-\frac{\tau}{T}\left(60\left(\frac{s_q(t,\omega_q)}{T}\right)^2\left(1-\frac{s_q(t,\omega_q)}{T}\right)^3\right)} \\ &\quad \cdot e^{o(\tau^2)} \\ &\overset{(a)}{\approx} 1 - e^{-f_b(s_q(t,\omega_q))\tau}\,, \end{aligned} \tag{4.4.11}$$

where (a) holds for τ small. As a result, for Q large, $F_{\alpha_q}(\tau|s_q(t,\omega_q))$ follows an exponential distribution with rate parameter $f_b(s_q(t,\omega_q))$. Substituting (4.4.11) in (4.4.6), we obtain:

$$\begin{aligned} F_\alpha(\tau|s{=}t) &= 1-\exp\left(-\sum_{q=1}^{Q} f_b(s_q(t,\omega_q))\tau\right) \\ &= 1-e^{-\lambda_{\alpha|t}\tau}\,, \end{aligned} \tag{4.4.12}$$

which states that, when Q grows large, $F_\alpha(\tau|s{=}t)$ follows an exponential distribution with rate $\lambda_{\alpha|t} = \sum_{q=1}^{Q} f_b(s_q(t,\omega_q))$.

Interestingly, under the above conditions, we can write:

$$\begin{aligned} \lambda_{\alpha|t} &\approx Q\int_0^T \mathbb{P}(s_q(t,\omega_q)) f_b(s_q(t,\omega_q)) d\omega_q \\ &= Q\int_0^T \frac{f_b(s_q(t,\omega_q))}{T} d\omega_q \\ &= \frac{Q}{T} \\ &\triangleq \lambda_\alpha\,, \end{aligned} \tag{4.4.13}$$

where we considered that:

- for high values of Q, also the number of IoT groups served by the MME grows large, hence the offsets ω_q can be assumed to be random variables uniformly distributed in $[0, T]$;
- such an observation holds also for $s_q(t, \omega_q)$, $\forall t$, since, by definition, $s_q(t, \omega_q) = \text{mod}(t+\omega_q, T)$;
- exploiting the law of large numbers, $\lambda_{\alpha|t}$ can be approximated accurately by its average value. From (4.4.13), it follows that $F_\alpha(\tau|s{=}t)$ not only follows an exponential distribution, but such a distribution does not depend on t, i.e., $F_\alpha(\tau|s{=}t){=}F_\alpha(\tau)$.

We now use this result to compute the CDF of the inter-arrival time between subsequent bearer instantiation requests at the MME, i.e., $F_\beta(\tau)$. We account for the fact that not all transitions to state A by an MIoT source lead to a packet transmission: after a transition in state A by an MIoT source, the probability of transmitting at least a packet is equal to $1{-}e^{-1}$. As a consequence, we can compute $F_\beta(\tau)$ as a sequence of z i.i.d. exponential time intervals, i.e., an Erlang(z, λ_α) distribution, weighted by the probability that two subsequent transmissions in the system are separated by $z-1$ transitions to state A without any transmission. Denoting the Erlang(z, λ_α) distribution with $E(z, \lambda_\alpha)$, we write:

$$F_\beta(\tau) = \sum_{z=1}^{\infty} F_{E(z,\lambda_\alpha)}(\tau)(1-e^{-1})(e^{-1})^{z-1}. \tag{4.4.14}$$

Using the above results, we can prove the theorem below.

Theorem 1. *When the IoT group offsets are independent of each other and Q grows large, $F_\beta(\tau)$ is given by:*

$$F_\beta(\tau){=}1{-}e^{-\lambda_\beta \tau} \tag{4.4.15}$$

with rate parameter $\lambda_\beta = \frac{Q(1-e^{-1})}{T}$.

Proof. First, we substitute in (4.4.14) the CDF of the Erlang distribution, thus obtaining:

$$\begin{aligned}
F_\beta(\tau) &= \sum_{z=1}^{\infty}\left[1{-}\sum_{j=0}^{z-1}\frac{e^{-\lambda_\alpha \tau}}{j!}(-\lambda_\alpha \tau)^j\right](1-e^{-1})(e^{-1})^{z-1} \\
&= \sum_{z=1}^{\infty}(1-e^{-1})(e^{-1})^{z-1} + \\
&\quad -\sum_{z=1}^{\infty}\sum_{j=0}^{z-1}\frac{e^{-\lambda_\alpha \tau}}{j!}(-\lambda_\alpha \tau)^j(1-e^{-1})(e^{-1})^{z-1}
\end{aligned}$$

where $\lambda_\alpha = Q/T$. We now solve the first summation, and invert the order of the two summations in the second term by exploiting the fact that $0 \leq j \leq z-1 < \infty$:

$$F_\beta(\tau) = 1 - \sum_{j=0}^{\infty} \sum_{z=j+1}^{\infty} \frac{e^{-\lambda_\alpha \tau}}{j!} (-\lambda_\alpha \tau)^j (1 - e^{-1})(e^{-1})^{z-1}. \tag{4.4.16}$$

Next, we isolate and solve the summation over the index z:

$$\begin{aligned} F_\beta(\tau) &= 1 - \sum_{j=0}^{\infty} \frac{e^{-\lambda_\alpha \tau}}{j!} (-\lambda_\alpha \tau)^j (1 - e^{-1}) \frac{(e^{-1})^j}{(1 - e^{-1})} \\ &= 1 - \sum_{j=0}^{\infty} \frac{e^{-\lambda_\alpha \tau}}{j!} (-\lambda_\alpha \tau e^{-1})^j . \end{aligned}$$

Solving the summation over index j, allows us to prove the theorem, i.e., $F_\beta(\tau) = 1 - e^{-\lambda_\alpha \tau} e^{\lambda_\alpha \tau e^{-1}} = 1 - e^{-\lambda_\alpha (1 - e^{-1}) \tau}$. $\square$

The above result states that the inter-arrival time between bearer establishment requests at the MME follows an exponential distribution, which implies that *the number of requests that the MME, hence the EPC, receives in a time interval follows a Poisson distribution*. This is a key result that allows us to characterize first the control overhead due to bearer establishment and forwarding, and then the delay performance of the EPC. Note that the above result holds also in more general scenarios where there are aggregators relaying the data packets generated by the MIoT sources toward the MME.

4.5 EPC Model and Analysis

In this section, we begin by showing the results of our experimental study, which highlight the following facts: (i) the bearer establishment takes a deterministic amount of processing, (ii) the variation in the delay of EPC entities other than the MME is negligible, (iii) a PS well mimics the MME serving policy. To perform our validation, we run and profile the components of a real-world EPC implementation called OpenAirInterface (OAI) [46], as described in Section 4.5.1. Then, based on the above key observations, we analytically characterize the EPC control overhead and, using a M/D/1-PS model, we derive an expression for the delay experienced by the MIoT traffic within the EPC.

4.5.1 Understanding the EPC through the OpenAirInterface

The OAI EPC is a software implementation of the cellular core network where the MME and the HSS are implemented as separate entities, while the S-GW and the P-GW are implemented as a single unit (called SPGW). To investigate the interaction between the EPC and the IoT sources, we connected the OAI EPC to a software simulator of the Radio Access Network (RAN), called Open Air Interface Simulator (OAISIM). Herein, UEs and eNBs communicate with the OAI EPC through an Ethernet cable, sending and receiving control messages as if a real RAN was in place.

Setup for OAI EPC Experiments:

The setup for this experiment is depicted in Figure 4.3. It shows the EPC components running in one machine connected to the OAISIM, where the UEs and eNB are run. In the experiment, we use Callgrind which is part of Valgrind tool suit to collect the CPU utilization (instruction fetch) for all components of the EPC (HSS, MME and SPGW). It is worth mentioning that the process of connection between OpenAirInterface EPC and OAISIM starts by attaching the eNB to the MME; then it goes on to process the connection requests of UEs via this eNB. Since, in our tests, we need to account only for the statistics of the performance counters that pertain to the handshake procedure for establishing connection to the UEs, we customized the source code to achieve this. To only collect the instructions fetch due to the connection establishment process of UEs, in each entity (HSS, MME and SPGW), we modified the source code of OpenAirInterface; we inserted a code in each entity to dump and clear the performance counters of the processes. This is done at two points in each of these processes. The first dump is done at the encounter of the first message sent to the entity in the connection establishment process. This clears the counters. The second dump is done after the entity sends the last message. In this way the number of instruction fetch obtained from the second dump will only account for the instructions fetch due to the handshake procedure messages sent, received and/or processed by the respective entity. In order for this process to be valid also for multiple UEs, we introduced global counters in the source code. The counters are used to keep track of the number of UEs processing the message we are interested in. In tests involving multiple UEs, the first dump is done for the first message from the first UE while the second dump is done for the last message from the last UE.

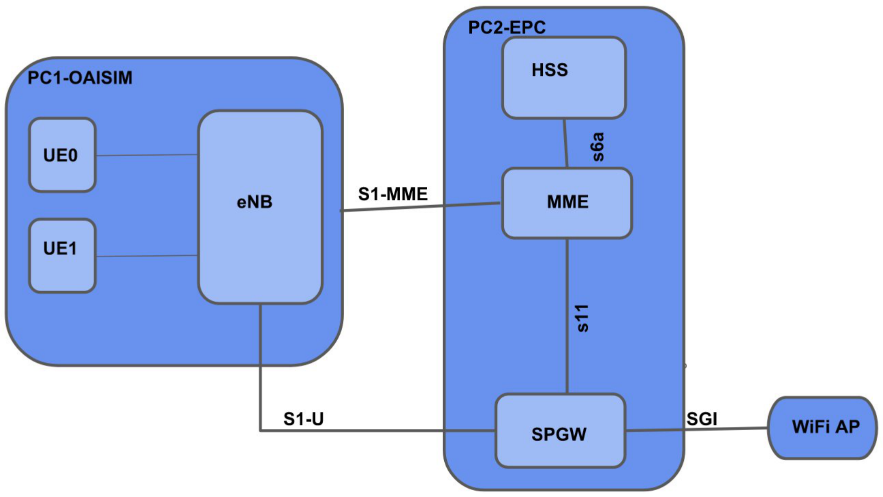

Figure 4.3: OpenAirInterface: Experiment setup.

The number of instructions fetch of the MME accounts for the CPU instructions between receiving the UE attach request message and the end of the bearer establishment; the bearer establishment ends after the MME has received and processed the Modify_Bearer_Response message from the SPGW. The number of instruction of the SPGW accounts for those instructions fetch between the receiving of the Create_Session_Request message from the MME and sending of the Modify_Bearer_Response message to the MME. Number of instructions fetch for the HSS accounts for those instructions between the establishment of connection between the MME and the HSS and just before the Update_Location_Answer message is sent to the MME.

The OAI EPC implements the standard bearer establishment procedure, which includes a superset of the messages exchanged between the EPC entities during the CIoT bearer instantiation procedure. However, below we report the results considering only the messages included in the CIoT procedure, as depicted in Figure 4.2. We subtracted from the total instructions fetch the instructions fetch due to the messages not part of the CIoT procedure.

Results of OAI EPC Experiments:

The total number of CPU operations for each EPC entity, obtained by profiling the OAI EPC with the Callgrind tool from the Valgrind tool suite [47], is depicted in Figure 4.4. Therein, the number of users attached to the EPC varies from 1 to 3 (the maximum number of users that OAISIM can support in our setting), and each data point has been obtained using 20 runs, resulting in a 95% confidence interval of up to ±0.7% of the plotted percentile values. Note also that, in each run, every UE performs one bearer establishment.

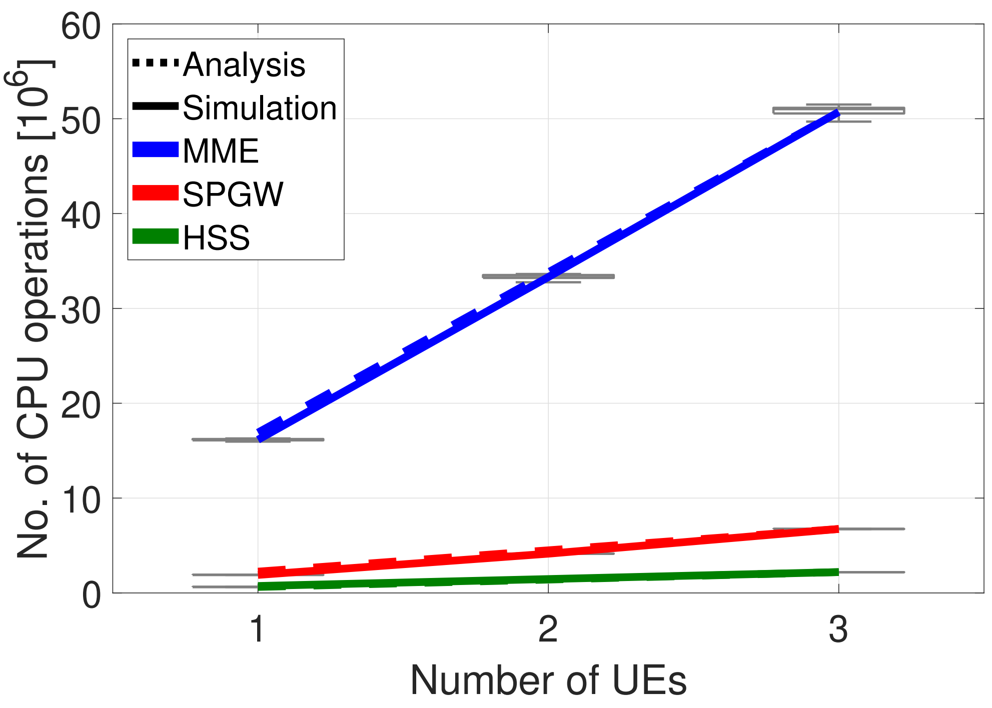

Figure 4.4: Number of CPU operations required by bearer establishment vs. number of UEs.

Figure 4.4 demonstrates that the job size associated with a bearer establishment procedure is fixed and deterministic. This is shown by two facts: (1) the number of operations required for a bearer establishment procedure grows linearly with the number of bearer instantiation requests, and (2) the variation of the number of CPU operations required by the EPC entities across different runs is negligible. The former is further highlighted in the plot by the excellent match between the solid line, showing the experimental values, and the dotted line, which represents a linear fit whose slope is forced to the average number of CPU operations required by a single bearer instantiation. The latter fact, instead, can be observed from the boxplots in Figure 4.4, representing the 10-th and 90-th percentile of the CPU operations distribution: the variance is very small in all analyzed cases and for any of the EPC entities in the system.

The second important observation we can make by looking at Figure 4.4 is that the MME is the dominant component of the performance for the EPC: the number of operations required by any other entity is at most the 13% of those needed by

Table 4.2: Bearer Establishment Time. 1 UE vs. 2 UEs

1 UE Bearer Time Average	2 UEs Bearer Time 1^{st} User - Avg.	2 UEs Bearer Time 1^{st} User - PS Avg.
0.84 ± 0.02 s	1.63 ± 0.03 s	0.87 ± 0.02 s

the MME. Given the fact that typical EPC implementations include entities with similar computational capability [48], as the traffic load changes, it is fair to neglect the variations in the delay introduced by entities of the EPC other than the MME.

Finally, we performed dedicated experiments to grasp some insights on the policy used by the OAI EPC implementation to serve packets that are simultaneously queued at the different entities. In this set of experiments, in order to avoid interference, we isolate MME, SPGW, and HSS, assigning to each entity a dedicated core of the PC acting as EPC. Note that all the UEs emulated through OAISIM make a bearer request almost simultaneously and it is not possible to determine beforehand their time of attachment. Therefore, users contend for the same resources during nearly the whole duration of the bearer establishment.

The first and second column of Table 4.2 show the average (over 20 runs) and the standard deviation of the time elapsing between the first and the last packet processed by the MME with one UE and two UEs, respectively. In the case of two UEs, we only consider the data relative to the bearer establishment of the first UE. In the third column, we demonstrate that it is fair to assume that a PS policy is in place. Indeed, considering the time in the second column, and halving the time in which the procedures of the two UEs overlap, we obtain a value that is very close to the one in the first column. Note that such an observation is consistent with the fact the PS policy closely emulates the behavior of a multi-threaded application running on a virtual machine instantiated on commodity hardware.

In addition, we tried to validate the analytical results we obtained for the delay experienced with OAI. This effort was not successful because of the absence in OAISIM for an option to configure and schedule the instantiation of a connection request for UEs in predetermined times.

4.5.2 Control Overhead and EPC Delay Characterization

As discussed above, the bearer establishment procedure in Figure 4.2 requires a deterministic number of CPU operations. Then, at every entity X involved in the procedure, each bearer instantiation is characterized by a fixed number of CPU operations O_X, which is the sum of the CPU operations required by the messages in Figure 4.2. It follows that the mean number of CPU operations per second that entity X has to perform is given by: $\mathbb{E}[\mu_X] = \lambda_\beta O_X$.

Next, we derive the pdf, $f_d(\tau)$, of the interval between a bearer request and its completion, i.e., the time passing from the first to the last message in Figure 4.2. To this end, we exploit the fact that the inter-arrival time of bearer requests at the MME follows an exponential distribution, as well as the observations set out below.

(a) The MME is the main bottleneck of the control plane. As shown experimentally in Section 4.5.1, the computational load requested to the MME for a single bearer implementation is roughly one order of magnitude larger than the computational load requested to any other entity. This implies that the CPU utilization of entities other than the MME is very low and variations of the control message processing times can be neglected, i.e., they can be considered as constant.

(b) It is fair to assume that the duration of a bearer instantiation procedure is very short compared to the timescale at which the MME load varies. Thus, each message within the same bearer instantiation sees a constant load at the MME. As a consequence, each bearer request can be considered as a single job, even if composed of multiple subsequent messages, with a computational load equal to O_{MME}.

(c) As shown above, the MME serving policy can be modeled through a PS discipline.

Given the above observations and the result in Theorem 1, we model the MME as an M/D/1-PS queue, where the deterministic service time depends on the capability of the MME, while the rate of arrivals of the bearer instantiation requests is equal to λ_β, as reported in Theorem 1. Then $f_d(\tau)$ can be written as,

$$f_d(\tau) = f_v(\tau) + K, \tag{4.5.1}$$

where:

- $f_v(\tau)$ is the pdf of the time spent by a bearer instantiation at the MME, i.e., the sojourn time of a job in the M/D/1-PS queue;
- K is the constant delay due to entities other than the MME (see our observation *(a)* above), which can be computed as:

$$K = \frac{O_{UE}}{C_{UE}} + \frac{O_{eNB}}{C_{eNB}} + \frac{O_{HSS}}{C_{HSS}} + \frac{O_{S-GW}}{C_{S-GW}} + \frac{O_{P-GW}}{C_{P-GW}}, \tag{4.5.2}$$

where O_X is the total number of CPU operations that entity X has to perform for each bearer establishment, while C_X is the computational capability of entity X, expressed in CPU operations per second.

To derive $f_v(t)$, we leverage the results in [49], which, owing to the complexity of computing such a distribution, provides the following approximation for the CDF:

$$F_v(\tau) \approx \psi e^{-\gamma\tau} \,. \tag{4.5.3}$$

In the above equation, ψ is given by [49]:

$$\psi = \frac{(1-\rho)(\lambda_\beta - \gamma)}{2\lambda_\beta(1-\rho) - \gamma\rho(2-\rho)} \,, \tag{4.5.4}$$

where $\rho = \lambda_\beta / D$ is the control traffic load at the MME, with $D = \frac{O_{MME}}{C_{MME}}$ being the deterministic service time of the bearer instantiation at the MME, and γ is the only positive solution of [49, Eq. (3.2)].

We remark that, given the pdf of the time interval between a bearer instantiation request and its completion (i.e., $f_d(\tau)$), we can compute the pdf of the delay that the control plane introduces in handling data packet forwarding at the MME when the CIoT optimization is supported. The derivation of the latter pdf implies considering only the messages in Figure 4.2 that are exchanged till the data packet transmission is completed. Then, based on our earlier observation *(b)* and given the number of CPU operations required by each message, we can obtain the pdf of the MIoT traffic latency by properly scaling $f_d(\tau)$.

4.6 Model Validation and Exploitation

In the following, we show how the behaviors – inter-arrival times and bearer instantiation delays – predicted by our analysis match those yielded by extensive simulations, using both synthetic traffic models [39] (Section 4.6.1) and real-world mobility traces (Section 4.6.2). Furthermore, we demonstrate how our model can be leveraged in the dimensioning and management of vEPC networks handling MIoT traffic.

4.6.1 3GPP synthetic traffic

We developed a Matlab simulator that accurately implements the 3GPP traffic model described in Section 4.2.3. The parameters we used are as follows: $T = 10\,\text{s}$

(as specified in [39]), $\delta = 10\,\mu s$, and group size equal to 50 MIoT sources.

Model validation. Here we validate the approximations introduced in our analysis as well as our main result in Section 4.4 (i.e., the inter-arrival time of bearer requests is exponentially distributed). To compare $F_\beta(\tau)$, computed as in Theorem 1, to the CDF of the inter-arrival time of bearer requests at the MME in simulation, we performed extensive experiments, varying the number of groups in the scenario and the offsets ω_q. In Figure 4.5, we present the results obtained with a specific set of offsets, as the number of groups served by the MME varies; however, similar results have been also obtained changing the ω_q values.

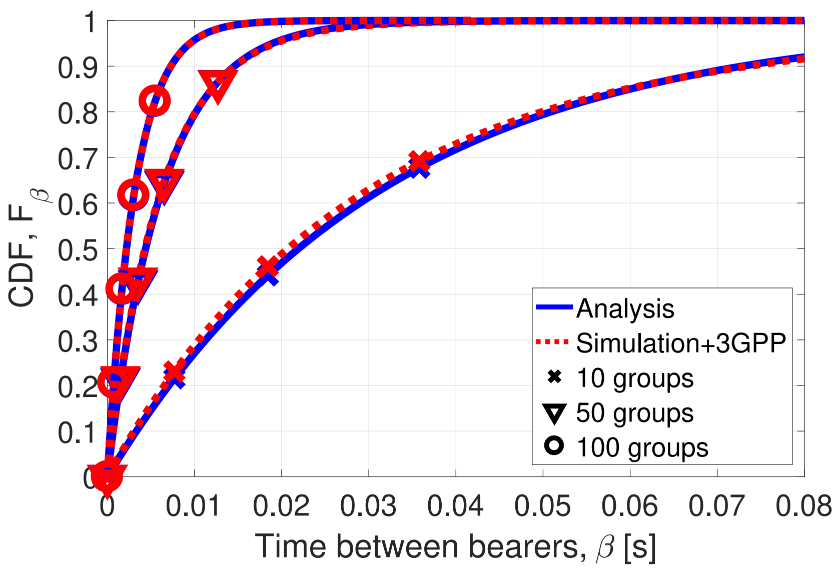

Figure 4.5: Inter-arrival time distribution of bearer requests: analysis vs. simulation using the 3GPP traffic model.

With as few as 10 groups served by the MME, Figure 4.5 highlights that simulation and analytical results closely match, thus showing that the exponential $F_\beta(\tau)$ captures very well the behavior of the 3GPP traffic model presented in Section 4.2.3. Furthermore, as expected, the match between the two curves improves as the number of groups served by the MME grows.

We now validate our delay model presented in Section 4.5.2. We first remark that, for the analytical derivation of $f_d(\tau)$, we neglected the load due to the integrity check and decryption, at the MME. Indeed, while a single control message requires (roughly) one million floating-point operations [50], studies on commodity processors show that nowadays a 50-byte packet (as in the case of IoT applications) requires few hundreds of floating-point operations for encryption/decryption [51]. In our simulations, instead, we account for data encryption/decryption as well as integrity check at the MME. Second, to compute the constant delay component of the delay distribution, K, in (4.5.2), we proceed as follows:

- we obtained the number of CPU operations, O_X, required at the EPC entities by a bearer establishment through our experimental measurements described in Section 4.5.1, and
- we leveraged the work in [48], which provides the computational capability of the EPC entities, C_X, based on real-world data from a large mobile network operator.

Finally, in order to validate the analytical expression of $f_d(\tau)$, we extended our Matlab simulator to perform the whole procedure in Figure 4.2, starting from the S1-AP Initial Message sent by the eNB. In our setup, all MIoT sources belonging to the same group, each containing 50 MIoT sources, are attached to the same eNB. Several eNBs may be attached to the same S-GW, while all S-GWs are attached to the same P-GW. Except for the RRC connection closing message sent by the eNB to the UE, all messages belonging to the same bearer instantiation travel sequentially between the involved entities. Each entity is implemented as a PS server whose service rate matches the processing capability provided in [48].

Figure 4.6 shows the analytical and experimental $F_d(\tau)$ in different scenarios. Specifically, we present the results of the CIoT optimization for two different values of traffic load, i.e., with $Q = 10{,}000$ and $Q = 15{,}000$. In the latter case, we also study two different configurations of the EPC to check whether changing the number of eNBs/S-GWs in the system has an impact on $F_d(\tau)$ or not.

First, we observe that the CDF of the bearer instantiation delay computed through (4.5.1)-(4.5.3) closely matches the experimental delay obtained via simulation – a fact that is especially evident looking at the tail of the CDFs. This result proves that considering the whole bearer establishment handshake as a single job at the MME, plus a constant delay due to the other entities, is a valid approximation. Small differences between the analytical and experimental CDFs for low values of delay, are mainly due to the model in [49], used to approximate the sojourn time in an M/D/1-PS queue. Indeed, due to the complexity of the M/D/1-PS characterization, [49] explicitly aims at modeling with higher accuracy the tail of the sojourn time CDF, which is what most matters in delay sensitive applications. Second, for $Q = 15{,}000$ the simulation results highlight that the two configurations with a different number of S-GWs provide exactly the same delay CDF, which validates our finding: $F_d(\tau)$ depends only on the number of MIoT

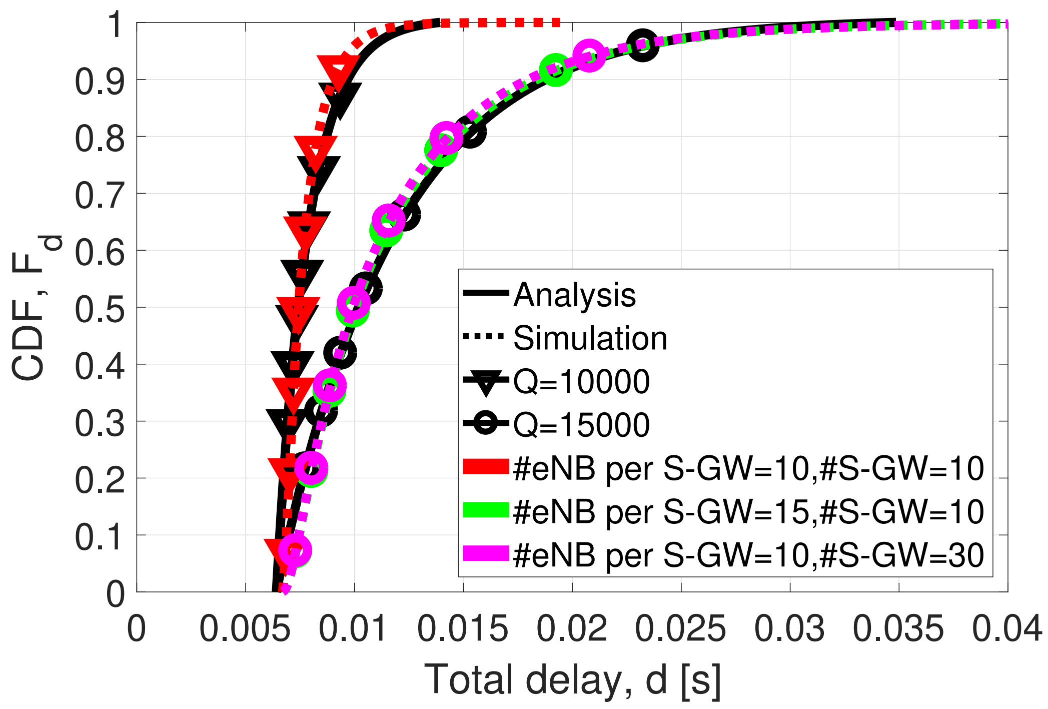

Figure 4.6: Delay distribution: analytical vs. simulation results, using the 3GPP traffic model.

sources in the scenario, and it is not affected by variations in the number of eNBs and S-GWs. This confirms that the MME delay component dominates that of the other EPC entities.

Model exploitation. We now show how our model can be used to develop efficient scaling algorithms for EPC networks serving MIoT traffic. Let us consider the following case, reflecting, e.g., a smart factory or cloud robotics application [52], where the delay introduced by the EPC should be less than 0.1 s with 0.99 probability. Since the delay performance depends on the number of MIoT sources served by the EPC and on the capability of the EPC entities, we need an algorithm that, given the IoT traffic, scales the capability of the EPC entities according to the number of MIoT sources in the system. Such an algorithm can leverage the analytical expression of $F_d(\tau)$ we obtained.

As an example, we considered a simple threshold-based algorithm, which, as the number of active IoT sources grows, increases the computation capability of the EPC entities by 30%, and then by 130%, with respect to the initial value, depending on the MME delay predicted by our model. (Note that increasing the EPC capability by 130% can be realized by creating a new instance of its components.) As shown in

Figure 4.7, such an algorithm meets the target performance. The figure also reports the delay corresponding to the cases when the capability values C_X are fixed to the initial value provided in [48], and to such a value increased by 30% or by 130%.

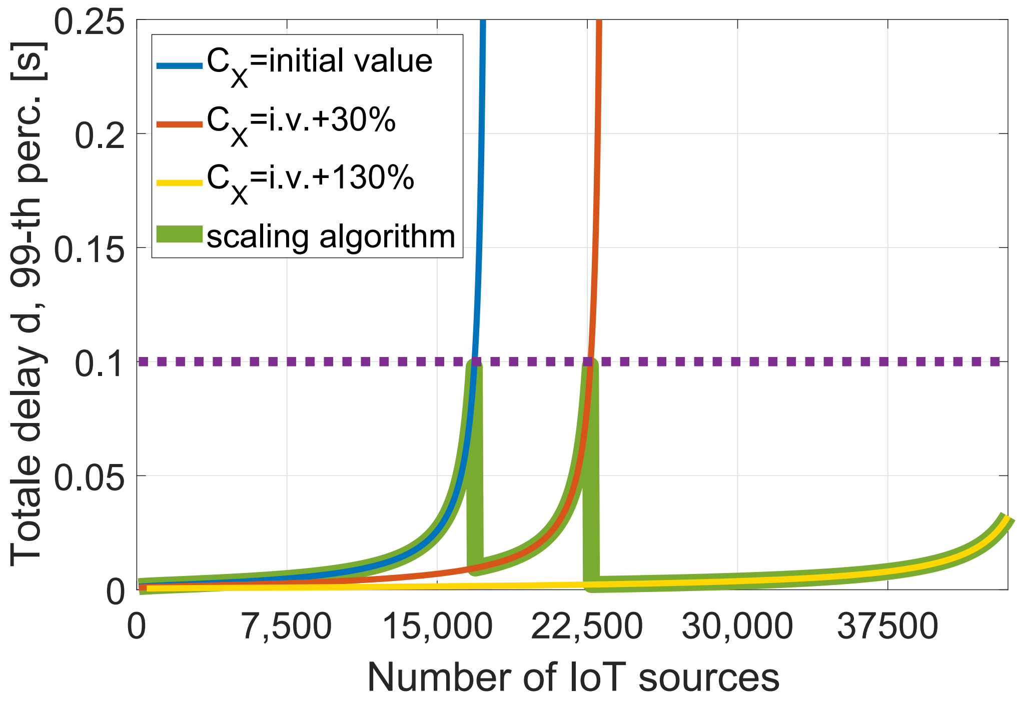

Figure 4.7: Analysis exploitation: 99−th percentile of the bearer instantiation delay vs. number of MIoT sources. Performance obtained with: the initial computational capability (C_X) of the EPC entities set to the value provided in [48] (blue line), capability increased by 30% (red line), capability increased by 130% (yellow line), and capability determined through the scaling algorithm (green line). Dashed, purple line: target value of the 99−th percentile.

Although more advanced scaling algorithms may be designed, we remark that, thanks to our model, even a simple threshold-based algorithm is able to meet the target delays and that our analysis, coupled with off-the-shelf virtualization tools like OpenStack, can be a key enabler to the support of IoT applications with delay guarantees.

4.6.2 Real-world trace

We now consider a real-world setting and leverage a large-scale mobility trace generated accounting for the MoST scenario [53]. The MoST scenario is a highly detailed representation of the mobility in the Monte Carlo urban area, including: (i) a multi-layered road topology, with tunnel and bridges; (ii) multi-modal mobility, e.g., users driving to a parking lot and riding public transportation thence; (iii) multiple types of coexisting users, e.g., commuters and tourists. The scenario models the mobility of a total of 27,967 users throughout an 8-hour period from 5 AM to 1 PM, and includes a total of 607 tagged points of interest (POIs) such as offices, restaurants, and tourist attractions. We assume that every time a user visits or stops at one of the POIs, a sensor, e.g., an identity-recognizing device, is triggered, resulting in a packet transmission, hence, a bearer instantiation request towards the MME serving the area.

Model validation. Our first objective is to establish whether the inter-arrival time between bearer requests obtained experimentally matches the exponential distribution $F_\beta(\tau)$ obtained through our analysis. To this end, we divided the time into 8 periods of one hour each, and computed the empirical distribution of the inter-arrival time of bearer instantiation requests in the MoST trace in every time period. The average arrival rates of bearer requests in the various periods are very different, reflecting the daily fluctuations in mobility. Nevertheless, as exemplified in Figure 4.8, the match between the analytical and the empirical distribution is excellent for all the time periods, proving that the inter-arrival time of the bearer requests obtained from the MoST trace follows an exponential distribution as well. This confirms that our analysis holds also for applications that do not follow the 3GPP traffic model described in Section 4.2.3.

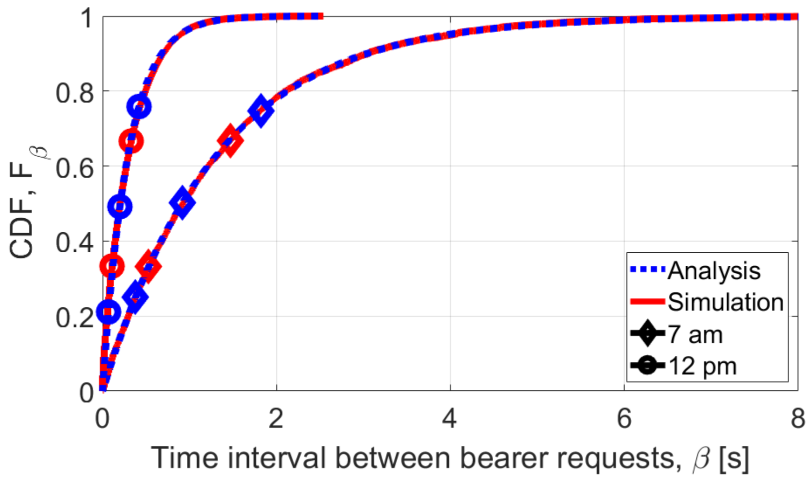

Figure 4.8: MIoT trace: Comparison between analytical and empirical distribution of the inter-arrival time of bearer requests at the MME, for two representative time periods.

Then we used the bearer requests obtained from the MoST trace to evaluate if the delay distribution of the bearer request procedures can be approximated with $f_d(\tau)$ (as in (4.5.1)) also in this realistic IoT scenario. We fed to the previously mentioned Matlab simulator the time instants of the bearer requests by sensors in the MoST trace and, since the number of bearer requests is rather small even in the rush hour, we reduced the EPC entities capability of one order of magnitude. The analytical and simulation results for the rush hour are compared in Figure 4.9, where the bearer establishment procedure delay is normalized by K (with K given in (4.5.2)).

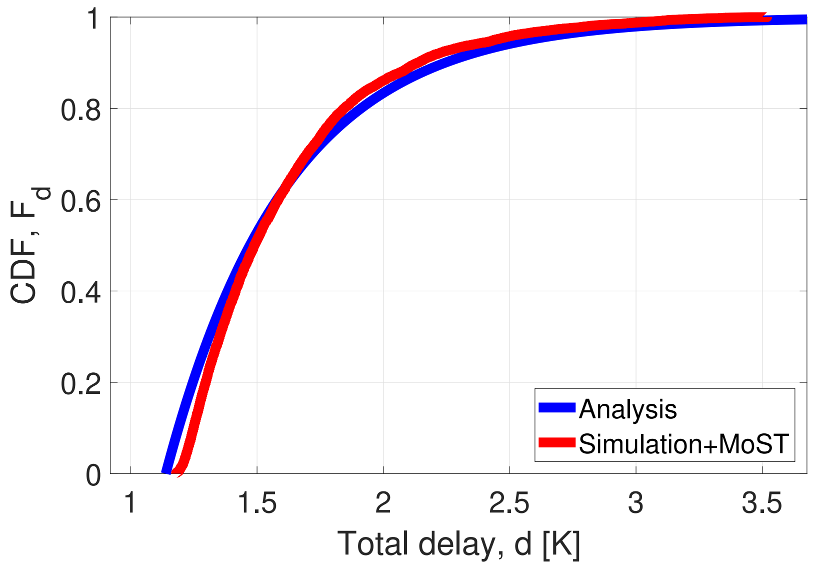

Figure 4.9: Delay distribution of sensor traffic: Analysis vs. simulation, using the MoST trace.

Given the fact that the largest difference between the two CDFs (which happens for low values of delay) is very small, we can conclude that our analysis well approximates the behavior of the EPC when serving MIoT sources, also in the case of a realistic scenario as the one of the MoST trace.

Model exploitation. We now present how our analytical results can be exploited under the MoST scenario. Figure 4.10 shows the time evolution of the 99−th percentile of the bearer instantiation delay, for different values of capability C_X. Considering a target performance of 0.1 s, we observe that the simple scaling algorithm employed when deriving Figure 4.7 (and which exploits our analytical results) successfully meets the delay requirement even under a sudden and very significant surge in the bearer request rate (see the black line in Figure 4.10).

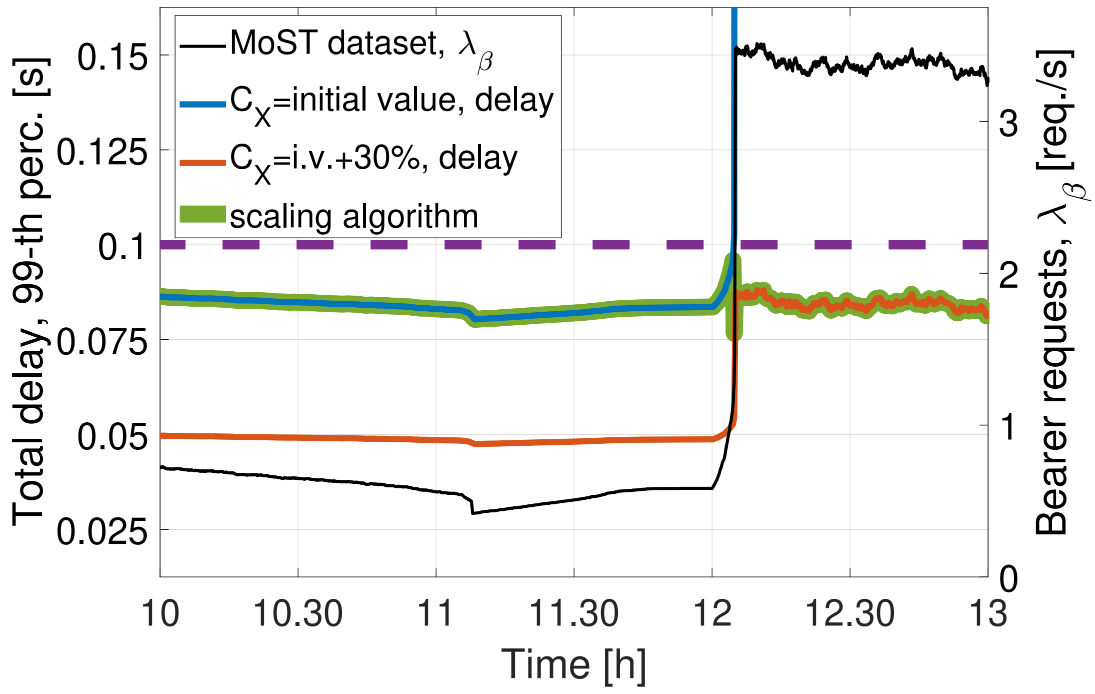

Figure 4.10: Analysis exploitation: 99−th percentile of the bearer instantiation delay vs. time, when the MoST trace is used. Black line: bearer request arrival rate from the trace; dashed, purple line: target value of the 99−th percentile. Performance obtained with: the initial computational capability of the EPC entities (blue line), capability increased by 30% (red line), capability determined through the scaling algorithm (green line).

4.7 Related Work

IoT support through cellular networks has recently attracted significant attention by both the scientific community and the standardization fora.

A first body of works deal with the requirements posed by IoT scenarios, and how 5G networks can cope with them. As an example, [54] quantifies the latency and capability requirements for the main IoT scenarios, from factory automation to parking machines, and discusses the improvement needed to both the radio access and core networks. The CIoT optimization and the role of MME are, however, not accounted for in [54], which mainly focuses on the SGW/PGW gateways. [52] has a narrower focus, namely, cloud robotics, and presents a working prototype; however, the latency introduced by the core networks and the entities therein is not taken into account.

Among the works that do account for the core network, most, including [56, 55], envision a *softwarized* network, where network functions are implemented through VNFs. Unlike our work, [56] does not specifically target EPC or any of its entities. The authors of [55], focusing on multicast traffic in IoT scenarios, specifically study the MME delay. Their proposed solution is to endow the SGW with some of the MME tasks, the opposite of the CIoT optimization we consider in our study.

The use of NFV and SDN, for the implementation of the EPC under massive IoT traffic conditions, has been discussed in [57], while enhancements to the standard EPC can be found, e.g., in [58]. That work introduces new entities in the network architecture, which are specifically devoted to the IoT support. Importantly, although such solutions yield a remarkable performance improvement, they inevitably involve significant changes to the standard.

Analytical models of IoT systems have been developed for specific application use cases, like management [59], opportunistic crowd sensing in vehicular scenarios [60], or ambient backscatter devices [61]. Other works have presented theoretical models for the study of networking aspects such as the performance of middleware protocols [62], implemented between the application and the transport layer, or of the random access procedure in Narrow Band (NB) IoT [64, 63]. Note, however, that none of the above works investigates the critical role of the EPC control plane in IoT-based systems; indeed few studies exist on the characterization of the overhead and service delay when the EPC handles massive IoT data traffic. In this context, the studies that are the most relevant to ours are [48] and its extension [65], which present a scheme for aggregating multiple IoT bearers and analyze the gain that is obtained with respect to the standard procedure. Such works, however, are based on deterministic inter-packet transmission time for MIoT sources and do not address the most recent and efficient 3GPP specifications for IoT support. Likewise, the study in [50] analytically evaluates the EPC control procedures (non specific for CIoT optimization) under a simple traffic model. A more comprehensive study on EPC control procedures has been presented [66], which, however, does not address any delay analysis.

To the best of our knowledge, our work is the first one deriving an exponential inter-arrival time for MIoT bearer establishments at the MME. It is important to stress, however, that exponential inter-arrival times are not an assumption, but the result of the analysis in Section 4.4, which is based on the 3GPP MIoT traffic model [44] and has been validated in Section 4.6. Finally, we remark that the goal of our work differs significantly from that of [45], which presents the Markov Modulated Poisson Process (MMPP) model for individual IoT sources that we adopt to develop our analysis and that is in accordance with the 3GPP traffic model for IoT. Indeed, [45] investigates large-scale IoT scenarios via simulation only: it does not present any analytical model of the EPC or of its control procedures under IoT traffic support.

4.8 Summary

Many massive Internet-of-things applications have stringent delay requirements, and effective dimensioning of EPC entities is crucial in order to meet them. In our study, we have identified the MME as the main source of control-plane latency, and sought to characterize the delay it introduces. Specifically, we have developed closed-form expressions linking the number of IoT sources to the inter-arrival times of bearer requests, and the latter to the MME latency. We have modeled the MME itself as an M/D/1-PS queue, where the processing time required by each individual bearer requests is deterministic.

We have confirmed the correctness of our model through a two-pronged study, including:

- measuring and profiling the performance of a real-world EPC implementation so as to characterize the actual behavior of the EPC entities;
- leveraging both the 3GPP traffic model and a real-world, large-scale mobility trace to verify that simulation results on the distribution of request inter-arrival times and of EPC delays agree with our analytical, closed-form expressions.

We further demonstrated how our model can be exploited in order to adapt the computation capabilities of the vEPC entities to the variations of incoming traffic.

Chapter 5

Conclusion

In this thesis, we proposed quite a few techniques to improve the energy efficiency, one of the Key Performance Indicators (KPIs) set by 3GPP for energy management, of the future generation mobile networks. We leveraged the flexibility provided by Software Defined Networking (SDN), and Network Function Virtualization (NFV), also referred to as network softwarization, to optimize resource usage in different parts of the mobile network. Specifically, we proposed the implementation, on top of an SDN controller, of an energy efficient routing algorithm in the backhaul network in Chapter 2, identified a virtualization environment in the context of Mobile-accesss Edge Computing, which entails lesser system resource overhead and thus power consumption in Chapter 3, and characterized control traffic and delay of IoT traffic and exploit this to scale resources according to traffic load in Chapter 4.

In Chapter 2, we addressed energy-efficient flow allocation in the 5G backhaul network. We first formalized flow allocation by formulating an optimization problem akin to a bin packing problem. However, the complexity of this formulation is unbearable in large-scale scenarios. We therefore resorted to a heuristic approach and developed an algorithm based on bin packing optimization problem. We specifically formulated a first fit bin packing heuristic algorithm implementation is deployed on an SDN platform.

In the energy efficient routing application, EMMA, presented in Chapter 2, traffic forwarding rules are created by an SDN controller. This application controls the operational states of the OpenFlow switches in addition to the (re-)allocation of traffic. We implemented EMMA on top of the Open Network Operating System (ONOS) SDN controller and derived experimental results by emulating the network through Mininet. The comparison between EMMA and the optimal solution (obtained in a small-scale scenario) showed that the EMMA performance is very close to the optimum. Also, in larger scale scenarios, emulation results highlighted that EMMA can provide a dramatic energy improvement with respect to our benchmark where switches are always on. The work in this chapter can be further investigated by considering the effect of the EMMA mechanism on flow QoS.

In Chapter 3, We set out to assess the additional power consumption due to VM- and container-based virtualization in MEC scenarios. Using VirtualBox and Docker, respectively from VMs and Containers, as reference technologies, we performed an extensive set of real-world measurements. The measurements are done for a wide set of synthetic and real-world workloads. The measurements involved recording of system resource usage, i.e., CPU, memory, disk and network as well as the response time for disk io operations. In all measurements the corresponding power consumption is also recorded. Through our measurements, we found that CPU usage is the main driver of the global power consumption, and the extra power consumption of container-based virtualization is not only lower than that of VM-based virtualization, but also grows more slowly as the workload increases. As it is evident from the results, container-based virtualization is an attractive technology for MEC scenarios, when large numbers of virtualized applications run on the same physical hardware.

In chapter 4, we analytically characterized the IoT control traffic and delay incurred by the EPC. Then we developed a simulation and run various tests with different setups. Moreover we proved the assumptions we made while driving the analytical results with a real software implementation of 4G radio and core network by OpenAirInterface.

Many massive Internet-of-things applications have stringent delay requirements, and effective dimensioning of EPC entities is crucial in order to meet them. In our study, we have identified the MME as the main source of control-plane latency, and sought to characterize the delay it introduces. Specifically, we have developed closed-form expressions linking the number of IoT sources to the inter-arrival times of bearer requests, and the latter to the MME latency. We have modeled the MME itself as an M/D/1-PS queue, where the processing time required by each individual bearer requests is deterministic.

We have confirmed the correctness of our model by (i) measuring and profiling the performance of a real-world EPC implementation to check that our assumptions reflect the actual behavior of the EPC entities, and (ii) leveraging both the 3GPP traffic model and a real-world, large-scale mobility trace to verify that simulation results on the distribution of request inter-arrival times and of EPC delays agree with our analytical, closed-form expressions. We further demonstrated how our model can be exploited in order to adapt the computation capabilities of the virtualized EPC entities to the variations of incoming traffic. The work in this chapter can be extended to verify the analytical and simulation results for the delay characterization using real world implementation of the EPC.

In summary, we proposed a set of techniques and approaches to be applied to the backhaul network, the edge network and the core network of 5G that could lead to an efficient use of the network infrastructure and resources, and therefore to a more efficient energy utilization.

Publications

Journal publications

- Senay Semu Tadesse, Carla Fabiana Chiasserini, and Francesco Malandrino, "Characterizing the power cost of virtualization environments". In: Transactions on Emerging Telecommunication Technologies (ETT) (2018).
- Christian Vitale, Carla Fabiana Chiasserini, Francesco Malandrino, and Senay Semu Tadesse, "Characterizing Delay and Control Traffic of Cellular MME with IoT Support", IEEE Transactions on Mobile Computing (IEEE TMC) (2020).

Conference publications

- Senay Semu Tadesse, Carla Fabiana Chiasserini, Claudio Ettore Casetti, Giada Landi, "Energy-efficient Traffic Allocation in SDN-based Backhaul Networks: Theory and Implementation". In: IEEE Consumer Communications and Networking Conference (CCNC). 2017
- Senay Semu Tadesse, Carla Fabiana Chiasserini, and Francesco Malandrino, "Energy Consumption Measurements in Docker". In: IEEE Computers, Software, and Applications Conference (COMPSAC). 2017.
- Senay Semu Tadesse Senay Semu Tadesse, Carla Fabiana Chiasserini, and Francesco Malandrino, "Assessing the Power Cost of Virtualization Through Real-world Workloads". In: IEEE International Symposium on Local and Metropolitan Area Networks (IEEE LANMAN). 2018.
- Christian Vitale, Carla Fabiana Chiasserini, Francesco Malandrino, and Senay Semu Tadesse, "Characterizing Delay and Control Traffic of Cellular MME with IoT Support". In: IEEE/ACM MobiHoc. 2019.

Bibliography

[1] *The 5G Infrastructure Public Private Partnership, KPIs.* 5G Use cases and Requirements," *White Paper*, http://networks.nokia.com/file/28771/5g-white-paper.

[2] *The 5G Infrastructure Public Private Partnership, KPIs.* https://5g-ppp.eu/kpis/.

[3] M. Usama and M. Erol-Kantarci. "A Survey on Recent Trends and Open Issues in Energy Efficiency of 5G". In: *Sensors* (2019).

[4] *Vision on software networks and 5G, 5G-ppp White paper, Software networks W.* January 2017.

[5] Tarik Taleb et al. "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration". In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 1657–1681.

[6] *ONOS.* http://onosproject.org.

[7] B. Lantz, B. Heller, and N. McKeown. *A Network in a Laptop: Rapid Prototyping for Software-Defined Networks.* http://mininet.org. Accessed: 2016-21-12.

[8] Mamta Agiwal, Abhishek Roy, and Navrati Saxena. "Next generation 5G wireless networks: A comprehensive survey". In: *IEEE Communications Surveys & Tutorials* 18.3 (2016), pp. 1617–1655.

[9] Bo Yi et al. "A comprehensive survey of network function virtualization". In: *Computer Networks* 133 (2018), pp. 212–262.

[10] Hyojoon Kim and Nick Feamster. "Improving network management with software defined networking". In: *IEEE Communications Magazine* 51.2 (2013), pp. 114–119.

[11] Sherif Abdelwahab et al. "Network function virtualization in 5G". In: *IEEE Communications Magazine* 54.4 (2016), pp. 84–91.

[12] Shancang Li, Li Da Xu, and Shanshan Zhao. “5G Internet of Things: A survey”. In: *Journal of Industrial Information Integration* 10 (2018), pp. 1–9.

[13] *5G-Crosshaul*. http://5g-crosshaul.eu/.

[14] A. Vishwanath et al. “Modeling Energy consumption in High-Capacity Routers and Switches”. In: *IEEE Journal on Selected Areas in Communications* (2014).

[15] “Simultaneously Reducing Latency and Power Consumption in OpenFlow Switches”. In: *IEEE/ACM Transactions on Networking* (2014).

[16] *Cisco Catalyst 3750 Series Switches Data Sheet*. http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-3750-series-switches/product_data_sheet0900aecd80371991.html.

[17] L. Chiaraviglio, M. Mellia, and F. Neri. “Reducing Power Consumption in Backbone Networks”. In: *IEEE ICC* (2009).

[18] B. Heller et al. “Elastictree: Saving Energy in Data Center Networks”. In: *USENIX NSD* (2010).

[19] W. Fang et al. “Elastictree: Saving Energy in Data Center Networks”. In: *Computer Networks* (2013).

[20] A. Gyarf and J. Lehel. “On-line and First-fit Colorings of Graphs”. In: *Journal of Graph Theory* (1988).

[21] R. Carpa, O. Gluck, and L. Lefevre. “Segment Routing based Traffic Engineering for Energy Efficient Backbone Networks”. In: *IEEE/ACM ICC* (2014).

[22] F. Giroire et al. “Minimizing Routing Energy Consumption: From Theoretical to Practical Results”. In: *IEEE ICC* (2010).

[23] O. Okonor et al. “Link Sleeping and Wake-up Optimization for Energy Aware ISP Networks”. In: *IEEE ISCC* (2014).

[24] Francesco Malandrino et al. “An Optimization-Enhanced MANO for Energy-Efficient 5G Networks”. In: *IEEE/ACM Transactions on Networking* (2019).

[25] Y. Zhang and N. Ansari. “HERO: Hierarchical Energy Optimization for Data Center Networks”. In: *IEEE Systems Journal* (2015).

[26] T. M. Nam et al. “Energy-Aware Routing based on Power Profile of Devices in Data Center Networks using SDN”. In: *IEEE ICC* (2015).

[27] W. Fisher, M. Suchara, and J. Rexford. “Greening Backbone Networks: Reducing Energy Consumption by Shutting Off Cables in Bundled Links”. In: *ACM SIGCOMM Workshop on Green Networking* (2010).

[28] A. Coiro et al. “Reducing Power Consumption in Wavelength Routed Networks by Selective Switch Off of Optical Links”. In: *IEEE Journal of Selected Topics in Quantum Electronics* (2011).

[29] Aman Kansal et al. “Virtual machine power metering and provisioning”. In: *ACM symposium on Cloud computing*. 2010.

[30] Bhavani Krishnan et al. “VM power metering: feasibility and challenges”. In: *ACM SIGMETRICS Performance Evaluation Review* (2011).

[31] Ata E Bohra Husain and Chaudhary Vipin. “VMeter: Power Modelling for Virtualized Clouds”. In: *IEEE/ACM GRID*. 2010.

[32] Maxime Colmant et al. “Process-level Power Estimation in VM-based Systems”. In: *IEEE/ACM GRID*. 2015.

[33] Gaurav Dhiman, Kresimir Mihic, and Tajana Rosing. “A System for Online Power Prediction in Virtualized Environments Using Gaussian Mixture Models”. In: *IEEE/ACM GRID*. 2010.

[34] Ramon Bertran et al. “Accurate energy accounting for shared virtualized environments using pmc-based power modeling techniques”. In: *IEEE/ACM GRID*. 2010.

[35] Roberto Morabito. “Power consumption of virtualization technologies: an empirical investigation”. In: *IEEE/ACM UCC*. 2015.

[36] Rattanaopas Kritwara and Pichaya Tandayya. “Comparison of disk I/O power consumption in modern virtualization”. In: *Computer, Communications, and Control Technology (I4CT), IEEE*. 2015.

[37] Giuseppe Avino et al. “Characterizing Docker Overhead in Mobile Edge Computing Scenarios”. In: *IEEE/ACM GRID*. 2016.

[38] M Maternia et al. “5G PPP use cases and performance evaluation models”. In: *see https://5g-ppp. eu/wp-content/uploads/2014/02/5G-PPP-use-cases-and-performance-evaluation-modeling_v1. 0. pdf* (2016).

[39] 3GPP. *Specification: 23.401; General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access*. 2017.

[40] *Small Data over NAS, S11-U and SGi Interfaces*. https://www.cisco.com/c/en/us/td/docs/wireless/asr_5000/21-3_N5-5/Ultra_IoT_CSGN/21-3-Ultra-IoT-CSGN-Guide/21-3-Ultra-IoT-CSGN-Guide_chapter_0111.pdf. Accessed: 2018-21-12.

[41] Andreas Baumgartner, Varun S. Reddy, and Thomas Bauschert. “Mobile core network virtualization: A model for combined virtual core network function placement and topology optimization”. In: *IEEE NetSoft*. 2015.

[42] R. Ratasuk et al. “NB-IoT system for M2M communication”. In: *2016 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. Apr. 2016, pp. 428–432.

[43] Y. -. E. Wang et al. “A Primer on 3GPP Narrowband Internet of Things”. In: *IEEE Communications Magazine* 55.3 (Mar. 2017), pp. 117–123.

[44] 3GPP. *Specification: 37.868; RAN Improvements for Machine-type Communications*. 2014.

[45] Markus Laner et al. “Traffic models for machine type communications”. In: *IEEE ISWCS*. 2013.

[46] *OpenAirInterface, 5G software alliance for democratising wireless innovation.* `http://www.openairinterface.org`. Accessed: 2018-14-12.

[47] Valgrind. *The Valgrind tool suite*. 2018. URL: `http://www.valgrind.org/info/tools.html`.

[48] Go Hasegawa and Masayuki Murata. “Joint bearer aggregation and control-data plane separation in LTE EPC for increasing M2M communication capacity”. In: *IEEE GLOBECOM*. 2015.

[49] Regina Egorova, Bert Zwart, and Onno Boxma. “Sojourn time tails in the M/D/1 processor sharing queue”. In: *Probability in the engineering and informational sciences* (2006).

[50] Jonathan Prados-Garzon et al. “Modeling and Dimensioning of a Virtualized MME for 5G Mobile Networks”. In: *IEEE Transactions on Vehicular Technology* (2017).

[51] *Encryption performance on commodity processors.* `https://calomel.org/aesni_ssl_performance.html`.

[52] F. Voigtländer et al. “5G for Robotics: Ultra-Low Latency Control of Distributed Robotic Systems”. In: *IEEE ISCSIC*. 2017.

[53] Lara Codecá and Jérôme Härri. “Towards multimodal mobility simulation of C-ITS: The Monaco SUMO traffic scenario”. In: *IEEE VNC*. 2017.

[54] P. Schulz et al. “Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture”. In: *IEEE Communications Magazine* (2017).

[55] M. Condoluci et al. “Enabling the IoT Machine Age With 5G: Machine-Type Multicast Services for Innovative Real-Time Applications”. In: *IEEE Access* (2016).

[56] R. Vilalta et al. “SDN/NFV orchestration of multi-technology and multi-domain networks in cloud/fog architectures for 5G services”. In: *IEEE OECC*. 2016.

[57] Aman Jain et al. “A Comparison of SDN and NFV for Re-designing the LTE Packet Core”. In: *IEEE NFV-SDN*. 2016.

[58] Vasudevan Nagendra et al. "LTE-Xtend: scalable support of M2M devices in cellular packet core". In: *ACM SIGCOMM ATC Workshop*. 2016.

[59] Y. Zhang et al. "IoT-Enabled Real-Time Production Performance Analysis and Exception Diagnosis Model". In: *IEEE Transactions on Automation Science and Engineering* 13.3 (July 2016), pp. 1318–1332.

[60] V. Petrov et al. "Vehicle-Based Relay Assistance for Opportunistic Crowdsensing Over Narrow IoT (NB-IoT)". In: *IEEE Internet of Things Journal* 5.5 (Oct. 2018), pp. 3710–3723.

[61] D. Darsena, G. Gelli, and F. Verde. "Modeling and Performance Analysis of Wireless Networks with Ambient Backscatter Devices". In: *IEEE Transactions on Communications* 65.4 (Apr. 2017), pp. 1797–1814.

[62] G. Bouloukakis et al. "Performance modeling of the middleware overlay infrastructure of mobile things". In: *2017 IEEE International Conference on Communications (ICC)*. May 2017, pp. 1–6.

[63] R. Harwahyu et al. "Repetitions Versus Retransmissions: Tradeoff in Configuring NB-IoT Random Access Channels". In: *IEEE Internet of Things Journal* 6.2 (Apr. 2019), pp. 3796–3805.

[64] R. Harwahyu, R. Ch eng, and C. Wei. "Investigating the Performance of the Random Access Channel in NB-IoT". In: *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*. Sept. 2017, pp. 1–5.

[65] Shuya Abe, Go Hasegawa, and Masayuki Murata. "Design and performance evaluation of bearer aggregation method in mobile core network with C/U plane separation". In: *2017 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE. 2017, pp. 1–8.

[66] C. Vitale, C. F. Chiasserini, and F. Malandrino. "On the Impact of IoT Traffic on the Cellular EPC". In: *IEEE Global Communications Conference (GLOBECOM)*. Dec. 2018, pp. 1–6.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.