UNIVERSITY OF THESSALY
GREECE

Service Differentiation in Virtual Service-Based Network Architectures

Doctor of Philosophy
in
Electrical & Computer Engineering
by

Kostas Sergiou Katsalis

June 2015

The Dissertation of Kostas Katsalis is approved by:

____

____

____

Committee Chairperson

Electrical & Computer Engineering,
University of Thessaly, Greece

ACKNOWLEDGMENTS

The research work made during the course of my Ph.D, has taken place at the Department of Electrical and Computer Engineering, University of Thessaly, Greece.

I would like to thank my advisor Leandros Tassiulas for his research example, support and guidance that he has given me since my masters thesis and during the course of my Ph.D. His work has been a true inspiration and working together keeps me excited after all these years later. I would also like to express my thanks and appreciation to the members of my dissertation committee, Cathrine Houstis and Yiannis Viniotis; Yiannis Viniotis, for his enormous help and enthusiasm on the whole thesis project and Cathrine Houstis for all her support and insight.

I would also like to thank my professors, my lab-mates, and everyone who have aided my research growth the past five years. They are many, but I must in particular mention Thanasis Korakis, Vasilis Sourlas, George Paschos, Lazaros Gatzikis and Apostolis Apostolaras. Besides their help, they made my staying in University of Thessaly much more pleasurable.

Finally, I would like to dedicate this thesis to my family and Barbara Popowicz for their love and support all these years. They mean the most to me and is their continuous support that made the accomplishment of this thesis possible.

This thesis was supported by the Heracleitus research project, co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) Research Funding Program: Heracleitus II - Investing in knowledge society through the European Social Fund.

European Union
European Social Fund

OPERATIONAL PROGRAMME
EDUCATION AND LIFELONG LEARNING
investing in knowledge society
MINISTRY OF EDUCATION & RELIGIOUS AFFAIRS
MANAGING AUTHORITY
Co-financed by Greece and the European Union

NSRF
2007-2013
programme for development
EUROPEAN SOCIAL FUND

Dedicated to my family and Barbara M. Popowicz.

ABSTRACT OF THE DISSERTATION

Service Differentiation in Virtual Service-Based Network Architectures
by
Kostas S. Katsalis

Doctor of Philosophy, Post Graduate Program in Electrical and Computer Engineering,
University of Thessaly, Greece.
June 2015

An end-to-end architecture that adopts the SDN paradigm, across all technology domains, for network virtualization, management and control, is absolutely essential towards the vision of building a complete cloud ecosystem. An end-to-end infrastructure facilitates the interconnection of data centers with fixed and mobile end users, through heterogeneous multi-domain networks, seamlessly integrating optical metro and wireless access network technologies. Given the emergence of cloud computing technologies as well as the diverse QoS needs of future cloud and mobile cloud services, new end-to-end architectures are required, that are able to facilitate network programmability and control. In addition, the relevant management and control mechanisms that are able to provide service guarantees to every virtual network, have not been adequately addressed, yet they are absolutely essential for the relevant network operations and crucial for the SDN paradigm to succeed.

The main goal of this dissertation, is to investigate and develop key network management functions for differentiating services between classes of customers, in virtual end-to-end environments. In principle and using the classical terminology, a differentiated service doesn't give service guarantees per tenant network, rather it differentiates traffic and allows a preferential treatment of one traffic class over the other. Due to multi-tenancy effects in the new virtualized environment and the presence of time varying workload conditions, the application of stochastic control theory is required in order to rigorously analyze the performance of policies that are able to achieve differentiation objectives. Due to the allure of the virtual world, there is always the danger of getting caught up in frameworks that lack the implementability perspective. Towards this direction, another goal of this dissertation is to come up with the design of a realistic an end-to-end, multi-domain SDN architecture, that spans from the wireless access network up to the virtualized data-center that can serve as the ground-floor over which the service differentiation problems are investigated.

We begin by presenting the design of this multi-domain SDN architecture, that spans from the wireless access network up to the virtualized data-center. We present the general concept, where are our research work targets the wireless domain of the architecture. We then proceed with the development and analysis of closed loop, feedback-based scheduling and en-queueing algorithms, that can be used to provide guaranteed service differentiation per customer class. The framework developed is applicable in in various control points on the architecture (e.g http routers, wireless driver queues, server systems) and various problems where guarantee service is required. Stochastic analysis is the tool that we used to prove a number of properties from the class of policies defined, like steady state analysis and speed of convergence. The proposed policies were evaluated both theoretically and by simulations, while also implementation of the policies was made in testbed environment. Furthermore, our research also involves the development of a mathematical framework, to construct efficient content distribution networks over the virtualized multi-domain environments that we study. The relevant optimization approach is presented.

Contents

List of Figures

List of Tables

List of Abbreviations

AMP	......	Absorbing Markov Process
AS	......	Autonomous Systems
CCN	......	Content-Centric Networking
CDN	......	Content Delivery Networks
CTMC	......	Continuous Time Markov Chain
DP	......	Dynamic Programming
IETF	......	Internet Engineering Task Force
ICP	......	Internet Cache Protocol
IP	......	Internet Protocol
ISP	......	Internet Service Provider
QoS	......	Quality of Service
QoE	......	Quality of Experience
ADC	......	Application Delivery Control
SLA	......	Service-Level Agreement
SLO	......	Service-Level Objective
WRR	......	Weighted Round Robin
DWRR	......	Weighted Round Robin
KPI	......	Key Performance Indicator
MVNO	......	Mobile Virtual Network Operator
MOVNO	......	Mobile Optical Virtual Network Operator
SDN	......	Software Defined Networks
ONF	......	Open Network Foundation
VNO	......	Virtual Network Operator
PHY	......	Physical (layer)
DCF	......	Distributed Coordination Function
API	......	Application Programming Interface
WFQ	......	Weigthted Fair Queueing
IaaS	......	Infrastructure as a Service
PaaS	......	Platform as a Service
SaaS	......	Software as a Service
NFV	......	Network Function virtualization
OSS	......	Operations Support System

Chapter 1

Introduction

1.1 Motivation

Up until recently, choosing between vertical or horizontal scalability was the only way around the spiraling demands for computational and network resources. With the emergence of cloud computing technologies, physical resources can be virtualized, efficiently shared between different users and effectively exploited, by eliminating idle time operations and under-utilization. On the network segment, with the propensity for ubiquitous Software Defined Networking (SDN)[1] in all network domains, an immersive virtual networking environment promises to debunk all the challenges encountered in network configuration and management and provide enhanced cloud network efficiency.

However, as a consequence of the millions of additional services that are now provided to communities, such as those of mobile users, the computational and network resources again seem to be insufficient. Although cloud and SDN technologies promise to leapfrog an entire generation of technology, the truth is that there are still many challenges to be solved, in order to apply the cloud computing paradigm in end-to-end solutions. Towards the move to a cloud ecosystem, along with the advertised advantages, cloud technologies bring strong challenges to the designers of the cloud environments. Such challenges are orchestrating the presence of multiple tenants in the same virtualized infrastructure, providing QoS guarantees, and savings on the operating cost.

In this thesis, we focus on the *service differentiation* design challenge, that refers to the problem of defining provisioning resources among competing users or classes of users, in virtual end-to-end environments. In principle, service differentiation refers to the problem of providing to different user classes, the service levels for which they contracted, in Service Level Agreements (SLAs). SLAs usually describe Service Level Objectives (SLOs) with key performance indicators (KPIs). In our modeling framework, users or classes of users may be different tenants in a datacenter, different virtual network providers, or different applications of the same enterprise; resources may be defined as CPU cycles in a server or as wireless network capacity or combinations; while the service offered may be quantified via metrics such as delay, CPU cycles or throughput.

1.2 Problem Description and Contributions

Efficient end-to-end, multi-domain, SDN-based network architectures, together with efficient control mechanisms, are keys enablers to deliver guaranteed service differentiation in cloud based environments. The network architecture must be agile, able to provide the necessary abstractions, while also support the necessary programmability features that will enable the application of multifaceted sophisticated control. Is the application of control and resource allocation mechanisms, together with new advanced cloud computing techniques, that is expected to offer maximum harnessing of SDN/NFV and cloud computing technologies.

More specifically, in this thesis we address the problem of providing guarantee differentiating services, between classes of customers, in virtual end-to-end environments. In the devised approach, we dissect the high level challenge into the following research problems:

1. Design of an end-to-end, multi-domain SDN architecture, that spans from the wireless access network up to the virtualized data-center. This end-to-end architecture serves as the ground-floor over which the service differentiation problems are investigated.

2. Dynamic resource provisioning, in cloud based environments. This involves the development of dynamic scheduling algorithms, that can be used to provide service differentiation guarantees. In addition, the dynamic resource provisioning problem involves the modeling of applications, where multiple virtual operators utilize shared physical infrastructures, with the objective to maximize the service provider profit.

Although, cloud technologies unleash the potential for extreme scalability and unlimited resources, multi-provider operations are expected to constrain each other and sometimes clash. Traditional problems, like scheduling for CPU or network resources, need to be set in a different context, the one of the new virtual world. Meeting these needs is absolutely essential for the success of SDN/NFV and cloud computing technologies, offering to the community a clean-slate. Thus we strongly believe, there is an urgent need to create new control and management policies, while also adapt well-known mechanisms, according to the new needs imposed by cloud operations, in all network segments.

1.2.1 Contributions

Regarding the design of an end-to-end SDN architecture, we assisted in the design of an open architecture, that is able to provide multi-domain virtualization, facilitating not only network virtualization but also network management and control. Our work was made in the context of the CONTENT SDN solution [2], that we presented in a series of papers like [3],[4],[5],[6] and required to exemplify all the fictions the research community encounters and debunks regarding multi-domain SDN control. The design of this novel SDN architecture, is aligned with the Open Networking Foundation ONF [1] guidelines, while the research work presented in this thesis is related to the wireless access domain of the architecture and is presented in Chapter 2. The proposed solution allows Virtual Network Operators (VNOs) to independently control and manage their end-to-end virtual networks. For example, as can be seen in Fig.1.1 in an ideal setting of an end-to-end virtual network architecture, multiple independent VNOs can own and control their own virtual infrastructure. The idea is to handle the network in a similar way with the one we treat Virtual Machine ownership, management and control using cloud systems like OpenStack [7]. This will allow the network segment to be keep abreast of current events in cloud evolution.

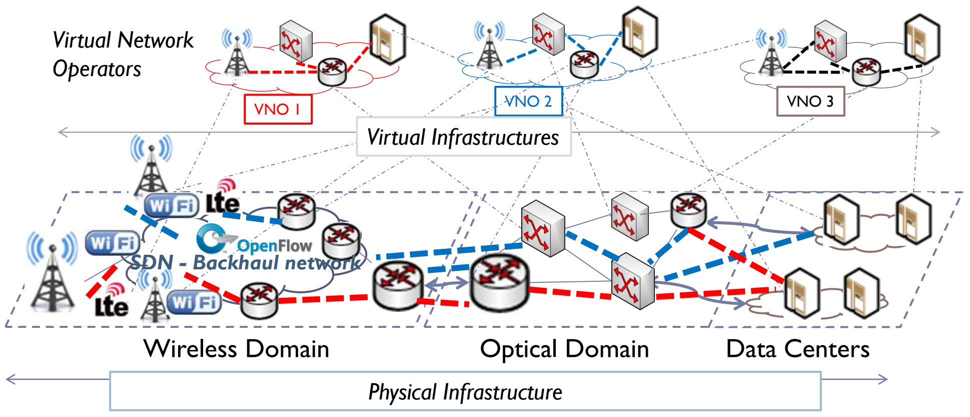


Figure 1.1: Virtual end-to-end network architecture

Cloud technology driven changes are inevitable and we believe that multifaceted analysis is required in order to come up with a framework that is robust, agile and scalable while is able to provide multi-domain virtualization by following the SDN paradigm. The proposed solution serves not only as the necessary, but also as the ideal ground-floor to apply new policies that we devise, for guaranteed service delivery of different virtual traffic flows.

The other research activity in this thesis, is related with dynamic resource provisioning problems, in cloud based environments. The contributions in this field, are also the main contributions of this thesis. We develop and analyze new dynamic scheduling algorithms, that are able to satisfy specific differentiation objectives, between competing customer classes. Different customer classes utilize network and processing resources (virtual or physical) that span end-to-end, from the wireless access up to the data-center. Due to multi-tenancy effects and the presence of time varying workload conditions, the application of stochastic control and queueing theory are required, in order to rigorously analyze the performance of policies that are able to achieve the differentiation objectives. Considering the modeling of applications where multiple virtual operators utilize shared physical infrastructures, we address the emerging content replication problem a service provider needs to consider, when deploying its network over virtual, multi-domain, heterogeneous environments. In our proposed framework, the benefit that the provider enjoys may be different per network operator for the same request, while our model takes into account the replication cost to every domain, as well as the user mobility, besides physical storage limitations.

More specifically, we propose and analyze a class Π of negative-drift Dynamic Weighted Round

Robin (DWRR) policies that can be used to satisfy specific differentiation objectives. A general mathematical framework is developed that can be applied to a broad family of scheduling problems, where differentiation of resources must be provided, like in OS thread/process scheduling or multi-processing systems. The decomposition of DWRR policies was introduced in [8] and [9], where we presented the theoretical analysis of convergence to the goal in the Cauchy sense, under saturated arrival conditions, while we investigated various properties related with speed of convergence and overhead reduction. In the following we extended this mathematical framework to include the general case of stochastic arrival/stochastic service per customer class. In [10] we present the proof of convergence for the non-work-conserving mode of operation, feasibility space analysis and dependence on the service redistribution algorithm on final performance. The analysis of DWRR policies is the subject of Chapter 3. An example SLA that the DWRR classs of policies can be used to satisfy, could be cast along the following lines: "*without any knowledge of statistics regarding the arrival and the service process in a cluster of servers, guarantee* 20% *of CPU power to requests of class A,* 30% *of CPU power to requests of class B and* 50% *of CPU power to requests of class C, in the long run*.

The following theoretical results can be collectively deduced from our analysis, of a class Π of negative drift DWRR policies. Let $\tilde{p}_i$ denote the maximum resource utilization a domain can enjoy, assuming no competition and let p_i to denote the resource share objective. Then for every policy $\pi \in \Pi$:

1. In the case of saturated arrivals, every policy $\pi \in \Pi$ converges with probability 1 (w.p. 1) to the goal percentage, p_i.
2. In the case of stochastic arrivals, when operating in non-work conserving mode, steady state exists for any policy $\pi \in \Pi$. Any policy converges with probability 1 (w.p 1) to the minimum between the goal percentage and the maximum utilization service.
3. Assuming steady state, the feasibility space for any policy $\pi \in \Pi$, in the case of stochastic arrivals and all modes of operation, can be clearly defined.
4. In the case of stochastic arrivals, under work conserving mode of operation:
   a. For any policy $\pi \in \Pi$ when $\tilde{p}_i \leq p_i$, domain i utilization convergences to $\tilde{p}_i$.
   b. For any policy $\pi \in \Pi$ when $\tilde{p}_i > p_i$ the convergence point depends on the service redistribution algorithm.

Besides the development and analysis of negative-drift DWRR policies, we also present new stochastic en-queueing techniques and prediction algorithms, that are able to provide differentiating services between competing customer classes. These techniques are related with the contributions presented in Chapter 4. The devised en-queueing algorithms were presented in [11], [12], while a prediction-based methodology was presented in [13]. For both the en-queueing and prediction techniques no steady state analysis is provided; nevertheless through extensive simulations, we present that these are acceptable approximations to the differentiation objectives. In the stochastic en-queueing approach, we present an approach to provide guaranteed service delivery to each customer class by selecting the queue to store incoming requests. This scheme is evaluated in two application scenarios: a) in server systems, where guarantees are provided on CPU usage, and b) in 802.11 Access Points, where the technique can provide guaranteed throughput to individual customer classes, without statistical knowledge of the channel conditions or the arrival process. In the predictions based approach, in contrast to existing works where estimation and prediction techniques are used to estimate average values, we propose a opportunistic scheduling scheme, where we adjust scheduling probabilities, based on system dynamics and deviation from the goal.

Regarding the contribution of this thesis, related to applications over the proposed network virtualization and control system, a mathematical framework for efficient content placement in multi-domain environments is proposed. This work was presented in the series of papers [14] and [15] and is the subject of Chapter 5 of this thesis. The services we focus are content distribution services provided over a cloud system that exploits the virtualized, multi-provider CONTENT architecture. The model takes into account the probability distribution of users belonging to one access network or another, while the network providers that the users are associated with, have different business relationships with both the physical providers and the content distribution providers. The proposed model also considers the content placement cost in every domain, besides physical storage size limitations, while the objective is to maximize the benefit that the provider enjoys.

In the following we present an introduction for the problems studied in this thesis, with the necessary references to related work.

1.3 Related Work

1.3.1 Multi-domain Network Virtualization and Control

The goal of multi-domain virtualization is to build end-to-end paths from the access network up to the virtualized data center and allow for seamless orchestrated on-demand service provisioning. The idea is to follow the successful model of Infrastructure as a service (IaaS), used for resource sharing in data-center operations, combined with the Mobile Virtual Network Operators (MVNOs) model, mobile operators lease wireless capacity from pre-existing mobile service providers, in end-to-end fashion. Multi-domain cloud networks are expected to bring substantial benefits and extend the business models of the involved stakeholders, in the ever-growing cloud landscape.

Although MVNOs operational models [16] are out for many years now, a sign of strain and drastic decrease appeared in the end of the previous decade [17]. The main reason for this decrease was that the MVNO signed contracts with Telecom Providers, which have acquired mobile radio spectrum rights and which owned the mobile base stations and the network equipment, but the level of control a MVNO had over his network was minimal. The charges where just matter of OSS functionalities in the provider's systems, without giving to the MVNO the ability to actually control his network. With the emergence of Cloud computing, SDN/NFV technologies, new VNO business models have appeared, where every VNO can actually own, manage and control a virtual slice of network resources.

Virtualization and network control across multi-technology domains has gained significant attention over the last few years and several solutions already exist [18], [19], [20]. The CONTENT project [4][3] tries to meet the requirements set by an end-to-end SDN solution, by building the necessary abstraction mechanisms and the control and management interfaces among Wi-Fi/LTE, Optical-metro networks and the data center infrastructure, while taking into the account the different virtualization models of resource and service based virtualization [21]. Similar research efforts have been focused on embedding virtual infrastructures over converged optical data center networks [22],[23], [24], [25] or across multiple substrate networks [18],[26]. Other concepts relevant to end-to-end cloud based approaches, are mechanisms to support live WAN migration of Virtual Machines [27]; these schemes utilize cloud computing platforms connected over VPN-based network connections. On the data-center side, significant research activity exists regarding network virtualization and control. This type of control has become a necessity in order to keep up pace with the new virtual environment created by the advances in hypervisor technologies and servers/systems virtualization. For example, in PortLand [28] pseudo MAC addressing of the VMs for L2 routing is used, CloudNaas [29] relies on OpenFlow[1] style forwarding, Oktopus [30] is able to satisfy bandwidth guarantees and in [31], the Pulsar system is presented that is used to create virtual data-centers. Furthermore, recent advances in network virtualization in the wireless [12], [32], [33], [34] and optical domains [25] aligned with the application of the SDN paradigm, empower the creation of virtual networks that span end-to-end.

In the SDN landscape, currently, many open-source SDN controllers exists, like POX/NOX, FloodLight, Contrail [35], ONOS [36] OpenDaylight [37], etc. and many research paradigms on applying SDN have been proposed. Note that, in applying SDN methodologies, of most concern is the realization that the SDN paradigm is not bind to a specific protocol. According to ONF, four planes are described and three types of functional interfaces [1]: The *Data Plane*, related with Infrastructure Layer where the network elements reside; the *Control Plane* for translating the requirements from the SDN applications and providing an abstract view of the network; an *Application Plane* where the application logic resides; and a *Management Plane* used to provide management functions and establish policies. Regarding the functional interfaces, these are the D-CPI (Southbound interface), between the Data Plane and Control planes; A-CPI (Northbound interface), between applications and the SDN controller; and a management interface. Each network domain, such as access/backhaul network, metro/core IP/MPLS and data center interconnect, may have dedicated controllers in logically centralized, physically distributed designs [38], [39].

By means of interface modeling, in modern SDN controllers like OpenDaylight [37], the interfaces are defined by an information model and the way to exploit them is essentially the same; it is irrelevant if the plugin that implements the required functionality serves a northbound or southbound service. Modeling languages like the YANG language[40] are used for the definition and semantics of interfaces and network element configurations. Regarding southbound protocols, from the industry perspective, protocols like OpenFlow and SNMP are used, while solutions where the XMPP protocol is utilized for the interaction between the network elements and the SDN controller has been proposed, like in the Contrail controller [35]. In the Contrail architecture, the control plane interacts with its peer control planes using industry-standard BGP. Regarding northbound protocols there are no available standards; the most common approach is to use a REST or a XML-RPC API exposed to the application layer.

Up to a point the above considerations and research activities constitute the core ideas and the current landscape in end-to-end SDN control. Note that an efficient SDN end-to-end system must be designed and developed to meet a series of system and functional requirements related to data-plane and control-plane convergence. What is more, the reliability and effectiveness of such a complex system depends on the reliability and effectiveness of the individual components at every distinct domain.

1.3.2 Dynamic Provisioning for Service Differentiation

Needless to say, that resource allocation and provisioning are very well investigated topics (e.g., in OS operations and thread schedulers [41], hypervisor operations [42], queueing systems [43]). Nevertheless, the importance of dynamic control schemes that are able to guarantee CPU/Network performance/provisioning, has emerged again because computational resources are still not unlimited abundant and because of the increased complexities in today's virtualized cloud-based environments. In this context, we believe that our work will help cloud architects in the definition of meaningful, controllable SLAs and the exploitation of new resource allocation algorithms, that are either proven to meet SLA objectives or are acceptable approximations. The key elements of our architecture model are that we have multiple resources (e.g CPU servers), spread over service tiers; we have multiple control points (servers, routers, dedicated controllers); and we have classes of customers (the service domains) that utilize physical or virtual resources.

In this thesis, our goal is to provide service differentiation between competing customer classes, in highly non-linear systems that that do not obey a number of rules that can simplify our analysis, like the superposition principle. In the systems that we examine, there is no proportional relationship (in fact in the systems we examine there is no relationship with non-stochastic terms), that can be accurately calculated, between the input and the output of the system. The analysis of non-linear systems is very challenging and requires advanced stochastic techniques, since mathematical tools, like Z-Transforms, Laplace Transforms, pole placement etc. to derive for example stability conditions that can be applied in linear differential equations [44], cannot be applied. Furthermore, open-loop control cannot be used to provide absolute guarantees in performance for dynamic systems. In open-loop (or feed forward) control, there are no measurements of the system output that will assist the control decision (e.g. Admission control, Leacky Bucket,Token Bucket Traffic Shapers)[45][46]. For example in token buckets, tokens are generated periodically at a constant rate and are stored in a token bucket. If the token bucket is full, arriving tokens are discarded. A packet from the buffer can be taken out only if a token in the token bucket can be drawn. If the token bucket is empty, arriving packets have to wait in the packet buffer. Thus we can think of a token as a permit to send a packet. Although WFQ/Token bucket [47] approaches can be used to perform traffic shaping efficiently, to regulate for example transmission rate, adaptive schemes are required in order to perform stability analysis under unknown arrival and service process statistics [48].

In order to investigate policies performance and provide analytical expressions, we exploit stochastic control theory tools and queueing theory results [43], [49], [50]. Closed-loop control relies on feedback information to regulate the input in order to minimize the error between the output and a reference input value. See [51], [52], [53] for a detailed analysis of linear and non-linear systems, adaptive control theory. We focus on adaptive feedback-based schemes, in order to apply control mechanism and guarantee specific differentiation objectives, between competing customer classes. In more detail, we propose and analyze negative-drift dynamic policies in order to satisfy the differentiation objectives; negative drift has been applied in multiple fields; for example it is used in adaptive control theory, where the goal is to minimize the cost related to the error, by moving in the direction of the negative gradient [52]. Note that feedback based controllers, like Proportional-Integral-Derivative (PID)[44] are also not appropriate when there are stochastic parameters in the difference equation that defines a relationship between the input and the output.

In the case of stochastic control theory, techniques like Gain Scheduling [54], Self-tuning regulators [44], fluid flow analysis [47], fuzzy control [55] and Luapunov stability[56], [57] have been extensively exploited in stochastic systems analysis. Proposals to change priorities dynamically, using feedback-based stochastic control are presented in multiple works like [58] and [59]; the authors of these works use a similar approach to ours but focus on rate control through scheduling. Service differentiation based on feedback control is also investigated in [60], while in [61] feedback control together with rate predictions are used to provide service differentiation. In the series of works [62], [63], [64] a QoS framework is proposed to perform service isolation and service differentiation. On-line measurements with prediction and resource allocation techniques is examined in [65] and a prediction based techniques are presented in [44]. Related analysis regarding stochastic analysis tools (namely Lyapunov analysis in systems where negative drift is applied to change priorities) were used in numerous works like [56], [66], [67], [68] and [69]. Relevant work has also been presented in works like [70] and [70], where the Luapunov optimization technique is performed. In [62] and [63] aggregate packet rate guarantees to individual clients is combined with negative drift and measurement-based admission control techniques. In addition to the related work presented in this introductory section, in Chapters 3 and 4 we present the related work specific to our analysis.

1.4 Thesis Outline

The contributions of this thesis are related with meeting a number of challenges imposed, towards building the future cloud-based Internet ecosystem. More specifically we contributed in the design of multi-domain SDN architectures and the development of algorithms used to provide dynamic resource allocation among competing users or classes of users, in virtual end-to-end environments.

In **Chapter 2** we present the design of an end-to-end, multi-domain SDN architecture, that spans from the wireless access network up to the virtualized data-center. We present the general concept, where are our research work targets the wireless domain of the architecture.

In **Chapter 3** we develop and analyze Dynamic Weighted Round Robin scheduling algorithms, used for guaranteed service differentiation. The framework developed is applicable in various control points of the architecture (e.g http routers, wireless driver queues, server systems) and various problems where guarantee service is required per customer class. Stochastic analysis is the tool that we used to prove a number of properties from the class of policies defined, like steady state analysis and speed of convergence.

In **Chapter 4** we present stochastic en-queueing polices and prediction techniques for service guarantees in various applications scenarios. In the stochastic enqueueing approach, we are able to provide guaranteed service delivery to each customer class by selecting the queue to store incoming requests. In the predictions based approach we adapt scheduling probabilities according to system dynamics and deviation from the goal vector. In both Chapters 3 and 4, the mathematical principles and the framework developed are generic and can be applied to a broad family of scheduling problems where differentiation of resources must be provided.

In **Chapter 5** we focus on the application layer of the SDN architecture and we present a mathematical framework for efficient content placement in multi-domain environments, together with the relevant optimization approach.

Finally, in **Chapter 6** we conclude our study, summarize the main findings of our work, while a discussion is made for possible future directions.

Chapter 2

End-to-End Virtual Infrastructures

Network architectures that are compliant with the Software Defined Networking (SDN) design paradigm, are expected to provide extreme flexibility for service orientation and allow for efficient use of both network and computational resources of cloud systems. Nevertheless, radical rethinking and removal of boundaries set out when studying per-domain communications are required, in order to unleash the hidden potential of SDN and provide a "holistic" network view.

In this chapter we present the design of an open end-to-end SDN architecture, while focusing on the necessary abstractions and virtualization techniques to integrate virtual wireless and optical resources. We especially shed some light in the field of wireless network virtualization, from an end-to-end perspective. Our work was made in the context of the CONTENT SDN solution, that we presented in a series of papers like [3],[4],[5],[6]. As we will present the proposed system is open and completely aligned with the Open Networking Foundation ONF [1] guidelines[1].

In this thesis, we are particularly interest on the *service differentiation* design challenge, in virtual end-to-end environments, like the one studied in this chapter. Users may be different tenants in a datacenter, different virtual network providers, or different cloud applications; resources may be defined as CPU cycles in a server or as wireless network capacity or combinations; while the service offered may be quantified via metrics such as delay, CPU cycles or throughput. The design of an end-to-end SDN architecture is the subject of this chapter, while service differentiation techniques that are able to provide guaranteed service to each customer class, are presented in the chapters that follow.

2.1 Introduction

To address the requirements of future virtual network operators, a combination of knowledge obtained by the recent advances in virtualization, Software Defined Networking (SDN) technology and Network Funtion Virtualization (NFV), must be investigated. Furthermore, in order to build and deploy

[1]The Open Networking Foundation (ONF) is a nonprofit organization, founded by Deutsche Telekom, Facebook, Google, etc. to improve networking through software-defined networking (SDN) and standardizing the OpenFlow protocol and related technologies

efficient cloud based services, we must depart from models that consider intangible multi-domain end-to-end settings.

The solution we study in this thesis, is about interconnecting geographically distributed computational resources, through ubiquitous converged virtual network infrastructures. Our goal is to deliver a novel SDN-enabled, inter-domain cloud networking platform, engineered with a bottom up approach, to meet the requirements of modern end-to-end cloud computing environments. Thanks to the approach devised, the successful Mobile Virtual Network Operator (MVNO) model [16] that is used to provide wireless services over the physical network provider, can be extended to a Mobile-Optical Virtual Network Operator (MOVNO) model. In this model a virtual operator, besides the wireless provider "sliced" resources, will be able to use virtualized resources on the optical metro network, to seamlessly interface with virtualized resources in the data center.

In more detail, in this chapter, we describe a hybrid LTE/Wi-Fi network [71][72], virtualized and interconnected with optical TSON metro networks [25][73], that supports frame-based sub-wavelength switching granularity, in the light of the CONTENT technical approach. We focus on the wireless domain and we present how traditional resource allocation and management frameworks of a wireless network, are exploited in order to support a virtualization system. Furthermore, we present how a set of virtualization services can interact with the control plane of the architecture, regarding virtual resources specification, management and operation. A note also is provided on implementation issues regarding the infrastructure topology, as a realization of the proposed layered architectural model. As we will present, the proposed layered architecture adopts the SDN paradigm, since the control functions are separated from the physical devices and operate on top of programmable network elements, through open interfaces.

This chapter is organized as follows. In section 2.2 we present the motivation for end-to-end SDN solutions and related work. In section 2.3 the end-to-end virtualization architecture is presented. In section 2.4 we provide details on the wireless network virtualization approach and implementation considerations. In section 2.5 we present an analysis for providing end-to-end QoS over the proposed system. Section 2.6 concludes this work and presents our future plans regarding multi-domain SDN/NFV network designs.

2.2 SDN Architectures and the ONF Guidelines

The key motivation behind the proposed solution, is that multi-domain and multi-technology problems are not sufficiently investigated. Active research is striving to consolidate frameworks that, with respect to SDN philosophy, will be able to provide performance guarantees, not per segment but end-to-end. Great research activity exists in the field [18], [19], [20], new protocols and frameworks, like OpenFlow [1] and OpenContrail [35], OpenDaylight [37] and new DN/NFV architectures are proposed [74]. A note on multi-domain virtualization and the landscape in reloated work was presented in chapter 1. All these are emerging solutions, which seem promising to address these issues. However, global standards for inter-plane communications, over diverse technologies are still missing,

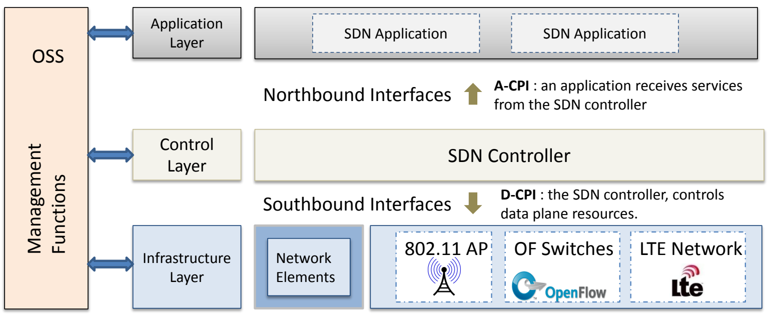


Figure 2.1: Layers of a SDN Architecture, according to the ONF guidelines.

there are no standards available for the northbound interfaces, while the technology to support QoS and performance guarantees in SDN networks is still in an immature state. A main component of the proposed solution is the communication framework that bridges the data and control planes, residing in different technology domains. More specifically, it investigates how TSON (Time Shared Optical Network) virtualized optical networks [25], that utilize GMPLS Control Plane mechanics, will be able to communicate with virtualized converged Wi-Fi/LTE networks, in such a way that traffic isolation between the virtual wireless networks is preserved and QoS guarantees are provided end-to-end.

2.2.1 The ONF guidelines

According to Open Networking Foundation (ONF) [1], four planes are described and three types of functional interfaces:

- The *Data Plane:* related with Infrastructure Layer where the network elements reside.
- The *Control Plane:* for translating the requirements from the SDN applications and providing an abstract view of the network.
- The *Application Plane:* where the application logic resides; and
- The *Management Plane:* used to provide management functions and establish policies.

A graphical representation of the architecture can be seen in Fig. 2.1. Regarding the functional interfaces, these are the D-CPI (Southbound interface) between the Data Plane and Control planes; A-CPI (Northbound interface) between applications and the SDN controller; and a management interface.

2.3 The Proposed Layered Architecture

The proposed layered architecture can be used to build end-to-end programmable networks, while supporting the Mobile-Optical Virtual network Operator (MOVNO) concept. The underlying physical infrastructure spans from convergent LTE and 802.11 wireless access networks, up to the virtualized data-center, through optical TSON backbone metro networks, Fig.1.1. In order to provide end-to-end network virtualization and programmability capabilities, agnostic to the underlying technological dependencies, the following layering approach is proposed, a visual representation of which can be seen in Fig. 2.2.

- *Heterogeneous Physical Infrastructure Layer.* This is related with *ONF's Infrastructure Layer*. The initial focus in our study, is upon a hybrid LTE/Wi-Fi access network, an optical metro domain that explores (TSON) technology, supporting frame-based sub-wavelength switching granularity and Data Center infrastructures.

- *Infrastructure Management Layer (IML) & Virtual Infrastructure Control Layer (VICL).* These layers are related to ONF's *Control Layer*. The IML is responsible for the creation and management of virtual network infrastructures, over the underlying physical resources. The VICL is responsible to provision IT and connectivity services, across the different network domains and expose functionalities to manipulate the network connectivity to the Application layer. In systems that support our architecture, the control is made over the IML resources.

- *Service Orchestration & Application Layer.* This is related with ONF's Application Layer. The Orchestration Layer is responsible to orchestrate network-based applications and coordinate the combined delivery of cloud and virtual network resources. The application layer is the layer where the application and business logic resides.

First note that the architecture is not bind to a specific network element, controller technology, southbound/northbound protocols or any control framework specific to the network deployed. This protocol independence, allows the proposed CONTENT architecture to be adopted by various wireless and optical network providers, that are interested to tender and build the underlay infrastructures for Virtual Network Operators.

For the data-center virtualization, we rely on available tools and virtualization systems like hypervisor technologies and OpenStack. Note that the virtualization layer, either type 1 or type 2, is a key enabler for building IaaS clouds. There are multiple virtualization techniques, such as Operating System Level (e.g., Linux LXC) or Hardware Assisted Virtualization provided by Intel VT-x (e.g., XEN, Linux KVM, eSXi) allowing for various execution requirements. For the interconnection of the virtualized data center with the optical domain, many solutions can be utilized like the one presented in [23]. Regarding the optical segment, TSON is used as the enabling technology at the physical layer. TSON offers multi-facaded resource virtualization based on FPGA technology, with guaranteed bandwidth shares per MOVNO, by scheduling both wavelengths and time-slots. More information on optical routers and switches and network virtualization using TSON technology can be found in [25], [73] and [75].

A main innovation in the proposed architecture, is that the control plane (VICL) will not communicate directly with the physical resources; the set of all management, control, configuration actions, including also the reservation of the resources, are mediated through the IML and performed over virtual resources. A better understanding of the architecture and the services it provides, along with

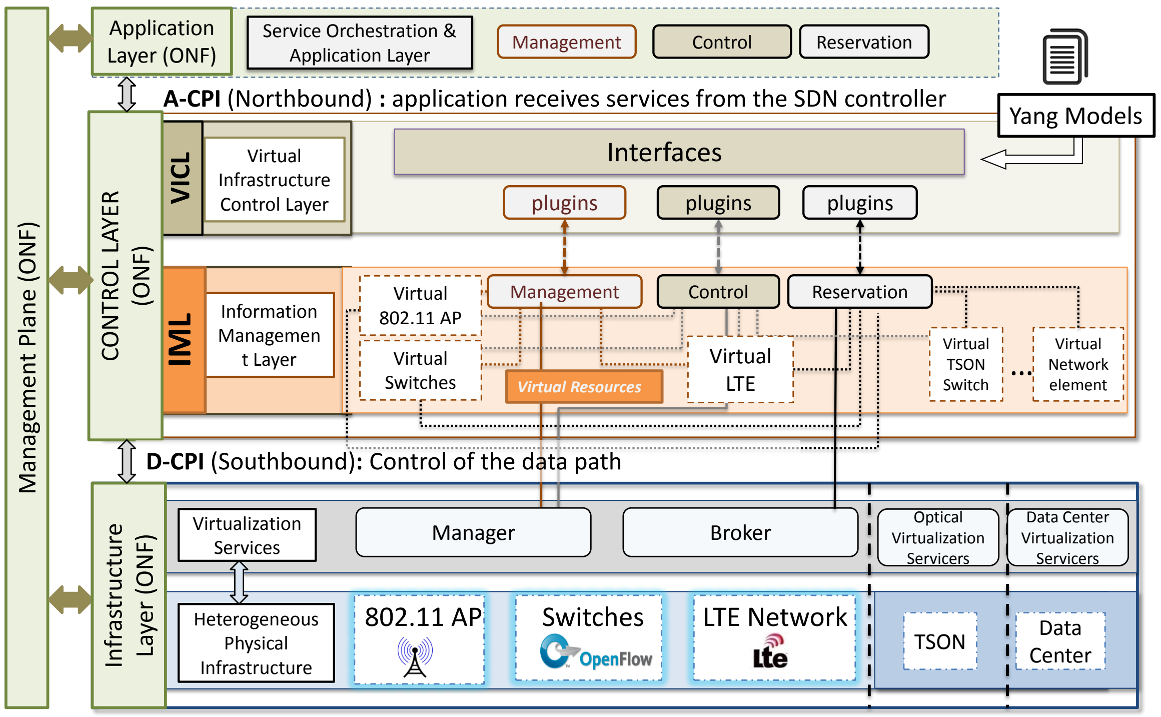


Figure 2.2: The proposed SDN-based layered architecture.

each architectural component, will be provided through the example of the wireless virtualization subsystem, presented in the following section.

2.4 The Wireless Network Virtualization Approach

We begin by describing the Heterogeneous Physical Infrastructure Layer components of the wireless domain. We note that during the curse of this thesis, the NITOS wireless testbed [76], [77] was used to evaluate the efficiency of the developed approach, in the wireless domain. In the NITOS testbed great effort is given in order to use open standards and open protocols in all the network segments of the wireless network, while also the control and management frameworks used. Thus the importance of the following analysis is similar for both wireless testbed infrastructure owners and commercial wireless providers. The basic components of a wireless network, related to the proposed solution are the following: a) *802.11 Access Network:* An 802.11 access network with multiple nodes, carrying multiple wireless interfaces. Compatibility with the 802.11 family of standards is promoted. b) *LTE Access Network:* The NITOS testbed exploits the installation of two $LTE245$ eNodeB units (Access Points) and a commercial EPC (S-GW, P-GW, MME, HSS) provided by SIRRAN [78]. c) *Back-haul Network:* A wired network with OpenFlow enabled switches. Used as the back-haul network between the 802.11/LTE Access Networks and a Gateway system. All the egress and ingress traffic between the wireless and optical networks, passes through the wireless gateway system. Furthermore, two widely used frameworks are utilized for the control and management of the wireless network elements. These are the cOntrol and Management Framework (OMF), regarding nodes management and control and Slice Federation Architecture (SFA), regarding resource abstraction and reservation. Detailed information regarding the OMF/SFA frameworks can be found under the project Fed4FIRE [79].

2.4.1 Wireless Network Virtualization Models

In principle, virtualization and slicing technologies of the wireless medium, are based on [80] FDMA, TDMA,Frequency Hopping (FH), CDMA, or Space Division Multiple Access (SDMA). Note that the proposed architecture, is not bind to a specific virtualization or "slicing" technique, as long as the virtual counterpart is properly defined in the IML. Furtermore, on a basic level, virtualization of a remote access system can occur not only in the physical layer, but also in the MAC layer, the network layer or even the application layer. In this thesis, we focus on the first three layers and the exploitation of two models of virtualization is discussed: resource based and service based virtualization [18]. In the former model we are actually virtualizing a physical resource, (similarly to a hypervisor that schedules CPU cycles to vCPUs), where in the latter we virtualize services (for example network services, using L3VPN). When virtualizing wireless access networks, both approaches are valid and the decision clearly depends on the requirements set by the virtual and physical infrastructure owners and of course from what the technology can support. For example, if a node carries multiple interfaces or/and multiple antennas, we can share spectrum usage by using multi-user beamforming or assigning different channels of operation to different interfaces. Then, we are talking for resource-based virtualization in the frequency domain. If we want to virtualize a single 802.11 Access Point (AP), then Multi-SSID technologies can be applied in order to assign users to different virtual APs. In this case we use service based virtualization, since the same physical resources (frequencies, time domain) are seamlessly controlled and the virtualization takes place in a logical layer. If again we examine the case where multi-SSID technologies are used like in [12], [32] or [33], where every virtual AP can be indecently controlled, while also its configuration affects the spectrum usage of other virtual APs, then we are discussing again for the case of resource virtualization. In 3G/4G networks, the current MVNO models available, follow the service based virtualization approach.

In the wired world, not only virtualization of the network resources, but also the way to achieve service guarantees and QoS are more or less straightforward. In contrast, in the wireless domain there are many limitations imposed by the stochastic nature of the wireless channel and in 802.11 networks, caused by the CSMA operation. Recent research works in 802.11 virtualization rely on the Beacon messages to adjust per-station minimum contention windows and transmit limits to affect the airtime usage to different group of users[32]. We can use VLAN to distinguish the flows, but actually there exists a sharing in the time domain of airtime usage. In virtualizing 802.11 networks the single-link and multi-station enhancements supported by protocols like 802.11ac can be also utilized. A more detailed discussion about QoS in the end-to-end architecture is made in subsection 2.5.

2.4.2 Provisioning of Virtual Resources

The main idea: Depending on the virtualization technique used in every network element of the wireless domain (L2/L3 switches, LTE network, 802.11 APs), every physical resource maps to one or multiple virtual resources in the IML. In addition, the VICL manages and controls the virtual resources, rather than the physical resources. The IML is responsible to hide the underlying specific wireless and optical network particularities. Furthermore, every MOVNO is able to provision his network directly utilizing IML services. In the NITOS case, the physical wireless resources are abstracted to virtual network elements in the IML, using two virtualization services, namely a *Brokering Service* and a *Management Service*.

Regarding data-plane operations, the actual data flows per MOVNO can be distinguished in the data-plane, using VLAN identification, used as the end-to-end identity of the flow[2]. It is important that VLANs are supported in both the wired backhaul and the LTE/802.11 access networks. A nice alternative would be to use GRE tunnels. In the case of 802.11 APs, programmable data-plane technologies, like software routers (e.g. Click Modular Router) and virtual switches like OpenVSwitch (OVS), are used to perform the necessary traffic shaping and tagging/un-tagging mechanisms. These are required in order to guarantee services and isolation per virtual flow. In LTE networks, QCIs identifiers are standardized and dedicated bearers can be configured with guarantee quality per flow. There are some native service guarantees available, that can be provided to each MOVNO (mapping different VLANs to different APNs-subscriber group and set the desired flow characteristics per VLAN). In the wired backhaul network, we rely again on programmable data-plane technologies and OpenFlow enabled switches, in order to perform forwarding operations, from/to the access networks to/from the Gateway system, to the optical side.

Every MOVNO will be able to provision and manage its virtual resources in all the domains (wireless, optical, data-center), without relying on domain specific interfaces and without affecting the operational state of other MOVNOs virtual resources. For example in an 802.11 AP, the IML is the layer a) where all Virtual APs (VAPs)[3] reside, b) that provides the necessary interfaces to the MOVNO and the VICL in order to perform virtual resource reservation and c) receive management and control commands by the VICL for some VAP.

Regarding network element virtualization and provisioning, SDN controllers like OpenDaylight, have already identified the need for resource virtualization. This would enable applications like: Virtual Tenant Networks (VTN) that perform virtual resource provisioning over a logical abstraction plane; Dynamic Resource Reservation for the reservation and scheduling features; or OpenDOVE that provides isolated multi-tenant networks on any IP network. Clearly, IML is part of the SDN control plane and depending on the architecture of the SDN controller in effect, can be an integrated part [4].

[2]QnQ service is used to identify flows over the GEANT network that interconnects the wireless testbed in Greece and a TSON testbed in UK. Thus, every packet carries two VLAN tags, when traversing the GEANT network; one that identifies the MOVNO and one used by the GEANT VLAN.

[3]we use VAP to note the virtual Access Point in the IML

[4]In the CONTENT solution, the initial IML implementation is based on the open source, under LGPLv3 license, Open-

In the context of the proposed solution, we decided to separate IML from the SDN controller and the reason is threefold. First, the complexities that arise when abstracting network elements and reserving virtual resources, do not only occur in the wired network, but also in wireless and optical domains. In a simple setup, where only L2/L3 wired network elements constitute the actual physical network, things are more or less straightforward. A limited set of resources, like ports and queues, are physical entities that are exposed as virtual and a limited set of functions, like VLAN mapping, are necessary. When we include wireless resources like 802.11 access points, LTE networks and FPGA-based optical switches into the picture, the construction of the virtual network is very complex. The second reason is to keep the architecture open to accept any SDN controller and not bind to a specific technology. As long as open interfaces, like REST, are exploited to facilitate the communication between the VICL and the IML, it is irrelevant how the VAP was created. Similarly, it is irrelevant which SDN controller (Python based, Java based etc), will implement the VICL functionalities. The third reason is related to protocols standardization. OpenFlow facilitates the management of forwarding operations in L2/L3 switches and protocols like NetConf, or OF-Config focus on the management of the switches. However, there are no available standards for the management, configuration and control of the wireless resources. In addition, we need to take into account, that every MOVNO does not have access to the physical resources rather to its virtual ones. Configuring information like SSIDs, channel, authentication/authorization information etc, requires a robust and extensible resource and service virtualization system, between the Control layer and the physical infrastructure. Our goal is that every MOVNO must manage, control and be aware of its virtual resources, be aware of their capabilities and interfaces, but should not be aware of the underlying hardware and should not maintain any information about the status of the other MOVNOs' networks. It's the responsibility of the IML to provide this functionality.

2.4.3 Virtualization Services

In this subsection, we present all the enhancements made in the NITOS control and management frameworks in order to be CONTENT-enabled. These enhancements are the *Broker Service* and the *Manager Service* and are reported as *Virtualization Services* in Fig.2.2.

The Broker

The *Broker*[5] is responsible for the resource advertising and the resource reservation, while it is the component that controls the slicing of the resources and guarantees slices isolation. It is the component that directly interacts with the IML (OpenNaaS) in order to build the resource descriptors. In more detail, it keeps an inventory with information regarding the available resources and their virtualization capabilities, which are then exposed through a REST interface. Through the same interface, it

NaaS framework [81]. In OpenNaaS resources are created by providing a resource descriptor to OpenNaaS ResourceManager. OpenNaaS expose services to the VICL using a REST API. More details on the OpenNaaS architecture can be found in [81]

[5]NITOS *Broker* is the successor of the NITOS scheduler that is now embedded in OMF Control Framework v6.

accepts requests for resource reservations, thus keeping each time the availability and the ownership status of every wireless resources. In addition, the *Broker* exposes a *Slice Isolation and Control Service* (SICS) to the *Manager* services, to infer if the requested actions to the physical resources should be authorized or not. The Broker' s *Scheduler* component is in charge of resolving any conflicting reservation requests that arrive and acts also as a policy enforcement point, regarding resource usage prioritization. It is important to note that the REST API is working side by side with a SFA interface. SFA is the de-facto framework for testbed federation, used for example in Fed4FIRE. This is achieved by having integrated these interfaces to the *Broker* and sharing a common inventory in the back-end.

The Manager

The *Manager* component exposes a secured REST interface (x509 certificates and HTTPS) to the IML and is responsible for the management, configuration and operation services of the physical nodes. An alternative design, without the Manager, would require direct access to the physical resources by the IML. With the *Manager* component the control of the actual resources is performed by a local testbed service, it is not bind to a specific IML technology implementation, while existing frameworks like the OMF can be utilized to perform actions like "image load" on 802.11 APs. This functionality now is exposed by the *Manager's* REST interface. The *Manager* operation relies on ssh-based access to the nodes or the OMF Aggregate Manager and remote commands execution, where the SICS service exposed by the *Broker*, guarantees that the caller of the management or operation activity is authorized to perform the requested actions. For example the REST interface can be used to turn a physical node OFF.

If the VICL sends a configuration command to some virtual resource on the IML, there exists affine functions that can cause inevitable ambiguity on the action that is actually performed on the physical node. We present how the *Manager* handles these cases of ambiguity, through an example scenario depicted in Fig.2.3. First note that a physical resource can have an one-to-one or one-to-many relationship with the virtual resources. This depends on the existence of a virtualization system and the abstractions provided by the *Broker*. For example in Fig.2.3 there are 3 MOVNOs, where each one owns a single VAP and every MOVNO issues a *turn VAP OFF* command. Note that the control of the VAPs can be made in the application layer, by using a single SDN controller for multiple MOVNOs or every MOVNO can utilize its own controller. Physical AP 1 maps to a single VAP, owned by MOVNO 1 (there is no virtualization scheme in frequency,time etc) and physical AP 2 maps to two VAPs, one owned by MOVNO 2 and owned by MOVNO 3. For example, assume that AP2 is able to perform Multi-SSID configuration and so VAP 2 and VAP 3 can have dedicated configuration regarding SSID information, encryption used, VLAN information etc. In all cases, the action triggered by the MOVNO application to close the VAP will change the operational status of the VAP to *OFF*. Nevertheless, only in the case of MOVNO 1 (message path 1 in the figure), the action can safely instruct the *Manager* to power off the physical AP. In the case where two VAPs operate on the same underlying hardware, a more complex procedure must be followed in order to protect the operational state (message path 2 in the figure). In all cases, it is the *Broker's* responsibility to

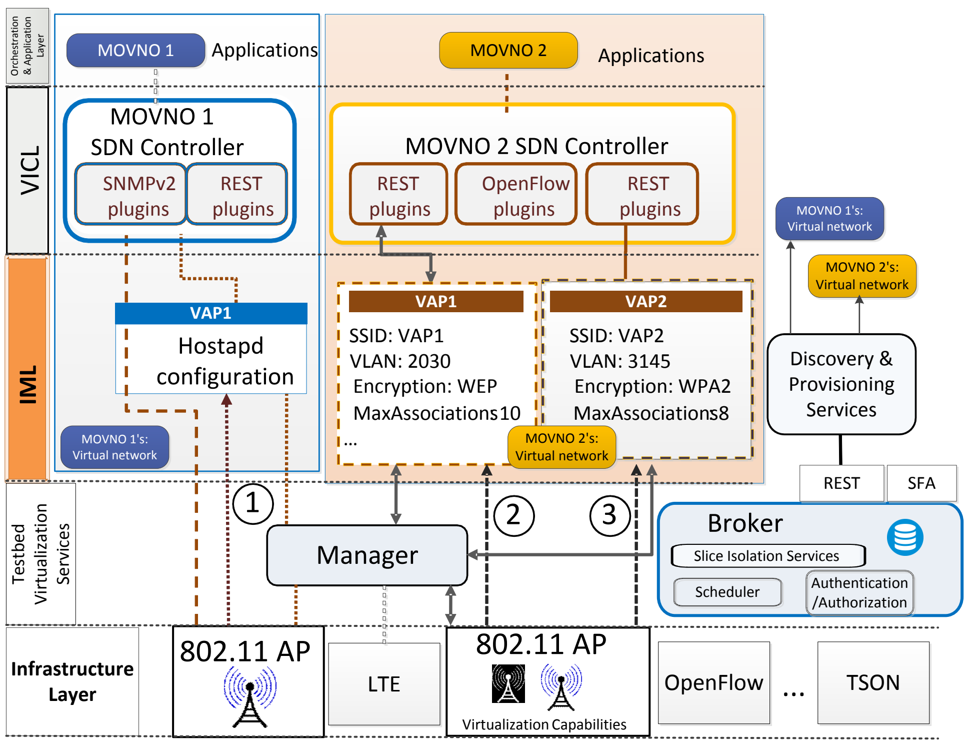


Figure 2.3: The *Manager Service*. A scenario where 3 MOVNOs send a command to turn off their VAP. VAP 1 is mapped to one physical resource where VAP 2 and 3 are mapped to the same physical resource.

advertise the capabilities, services and access rights on physical devices to the IML. This provides us with the extreme flexibility of integrating different types of hardware/software systems, using the same abstraction mechanism.

The second important point to exemplify, is the case where in the VICL a standard protocol like OpenFlow or SNMP is used in the Southbound interface. In the case of an OpenFlow enabled node (e.g. an 802.11 AP with an OVS used for VLAN tagging) or a native OpenFlow switch, the *Manager* is responsible to perform management operations (e.g. configuration of the SDN controller IP). Nevertheless, for the forwarding operations we make no OpenFlow commands-to-local translations (e.g. Manager's REST interface) and vice-versa. The raw commands are forwarded to program the data plane as is; the IML transparently handles the OpenFlow messaging from the VICL (message path 3 in the figure). We remind that OpenNaaS system is used as the IML implementation and the procedure of translating REST to OpenFlow besides tedious is error prone. Note that because the proposed architecture is open, different systems can be used to implement the IML that already have available REST to OpenFlow plugins. In this case, a different design of the management system can be adopted, regarding the actual control and configuration of the underlying physical infrastructure.

2.4.4 Virtual Infrastructure Control Layer

The VICL is responsible for the control and management of the virtual resources. In our VICL implementation, we utilize the OpenDaylight SDN controller [37], which has been extended to implement the VICL. At the southbound, the controller implements dedicated plugins to interact with the virtual resources that are exposed by the IML. The services offered at the plugins level, allow to manipulate the single virtual resources, with the highest level of details offered by the IML, following a protocol-independent approach. In fact, the structure and the semantics of configuration commands and resource capabilities available, are expressed in a vendor-neutral format by abstracting the specific protocol messages. This is achieved through a common data modeling language, like YANG [40]. The services offered by the plugins can be used by other controllers' internal modules to implement basic functions over each domain, for example edge-to-edge connection provisioning and topology or statistics services, and thus providing a further level of resource abstraction. The details of the D-CPI interfaces, in terms of compliance with specific protocols (e.g. OpenFlow, NETCONF, XMPP), is handled internally by the plugin itself and just reflected at the corresponding service model in terms of a) high-level resource characteristics and b) available actions expressed through the common modeling language. It is a task of the plugin implementation to guarantee the consistent mapping between the configuration actions or notifications exposed to the internal services in OpenDaylight and the specific protocol messages exchanged with the devices.

2.5 Service differentiation and QoS in End-to-End Infrastructures

As we described in the previous sections, the proposed end-to-end solution, can be used to build and support virtual network infrastructures, with isolated networks per Virtual Network Operator (VNO). The high level goal of every VNO, is to carry mission-critical and non-mission-critical traffic of its subscribers, from the access network, up to the data center and visa versa. In a holistic solution, VNO can cooperate with IaaS providers, that provide processing resources in the data-center. In our envisioned cloud ecosystem, a cloud infrastructure provider builds its infrastructure, by utilizing resources from VNOs and IaaS providers. A cloud provider is able to support not only the deployment of services and applications, but also enforce a level of control in the path from the end user device up to the server machines (virtual or not), in the data center. The motivation for this thesis and a major growing trend, is to adapt known QoS and service differentiation techniques, while also develop new, that are able to differentiate traffic in a cloud system, while at the same time mission-critical applications receive higher priority.

We begin our analysis, by following a top-down approach; we begin from the QoS contracts and service guarantees a cloud infrastructure provider usually signs, in todays cloud environments. Then, we proceed at each layer of control where QoS mechanisms can be applied. Note that in the multi-domain virtualized infrastructure that we examine, there are multiple control points in various network or processing elements, that affect the end-to-end performance. Is out of scope of this thesis, to provide a totalitarian listing of these points. Nevertheless, this step will assist us to identify

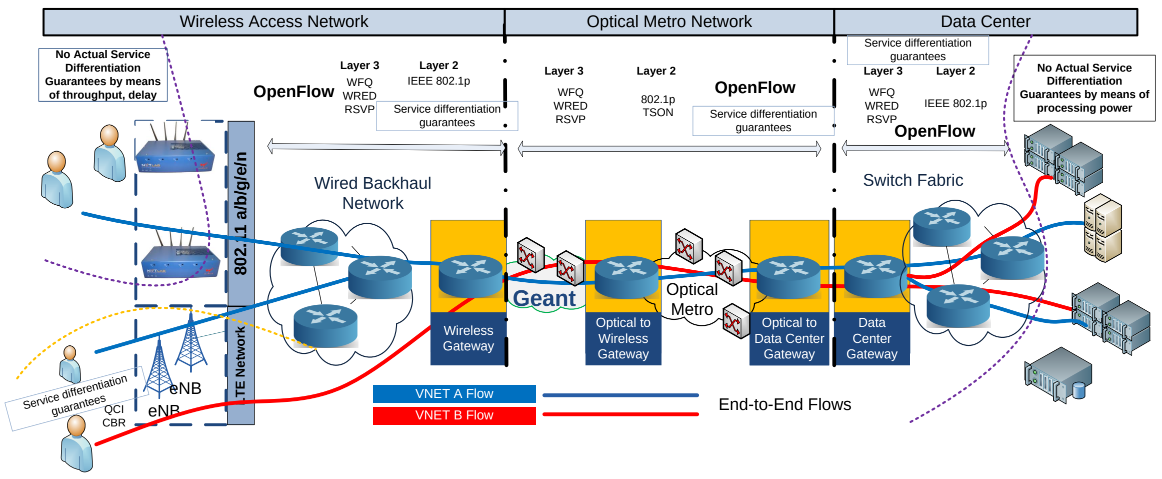


Figure 2.4: QoS considerations in end-to-end clouds.

the segments in the complex end-to-end architecture, where dynamic provisioning of resources is required. In the model that we consider, every cloud infrastructure provider utilizes (through ownership or leasing):

- Network resources. These resources, virtual or not, are exposed in the IML layer and be controlled using SDN control.
- Processing resources. In cloud environments, usually the terms processing resources and server systems are used interchangeably, to mean the combination of processing, storage, caching, memory etc. resources. Again, processing resources can be virtual infrastructures, that can be shared between different tenants.

Thus, in an end-to-end cloud system, using the proposed virtualization solution, a VNO can provide the necessary network infrastructure, while a cloud IaaS provider can provide the processing, storage etc. resources. Then, every cloud operator, can create virtual flows form the wireless access, up the virtual or physical server machines in the data-center. Over this end-to-end cloud system, after convergence is achieved in both the data-planes and control planes of the architecture, what we are really interesting in, is the service delivery guarantees we are able to provide. Actually, for what service guarantees an end-to-end cloud provider can sign for, with his customers. Generally speaking, SLAs with QoS guarantees required by cloud customers, typically include functional and non-functional requirements. Functional requirements include: response time, percentiles, throughput guarantees, error rate, server capacity etc. Non functional requirements include timeliness, scalability, availability, adaptability etc. Both of these lists can be quite extended, depending on the context.

Note that, virtual network overlays have become a very hot topic alongside SDN, in such a degree that many people think SDN and network virtualization are synonymous. Nevertheless, there is a clear and distinct relationship between SDN and network virtualization. Applying the SDN paradigm is about facilitating network management and control, separation between the control and data-plane operations by using open interfaces, while network virtualization is about network flow isolation, in multiple layers of the protocol stack (e.g. VLAN, MPLS, VPN etc.). Virtual switches can be thought

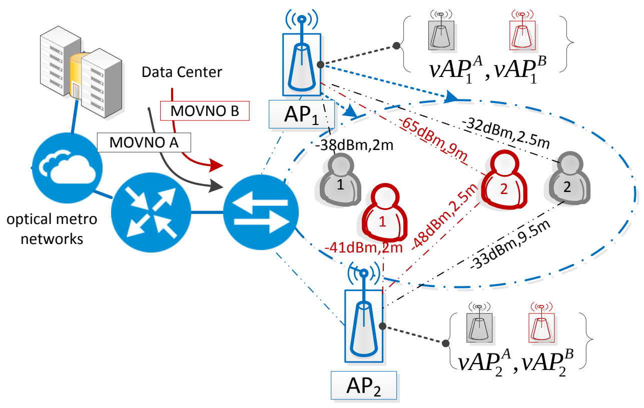


Figure 2.5: Evaluation scenario.

of as the connecting link between these two meanings, since they enable SDN functionality, while they are able to build virtual network infrastructures.

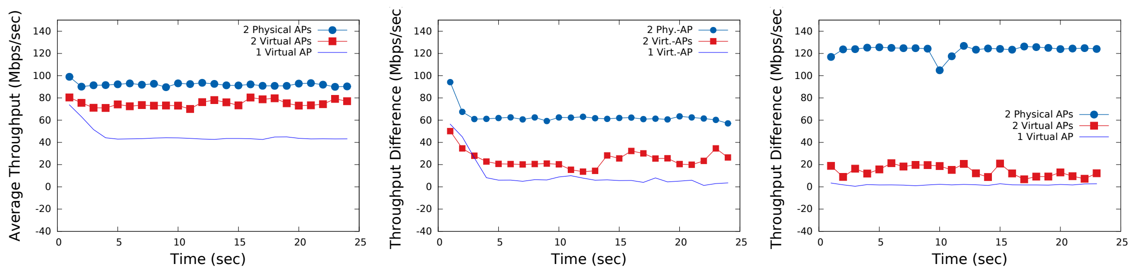
As we described, the proposed solution offers the enabling technology for virtual network provisioning, control and management, by exploiting available network virtualization techniques and modern SDN designs. The questioning and reasoning behind our research are the following. *In such a complex system how end-to-end service differentiation between different traffic flows can be provided? Which elements of the architecture and which mechanisms we must leverage and possibly extend, in order to perform control and reinforce end-to-end service differentiation operations?*

2.5.1 Evaluation of the Virtualization Approach

In this section, we demonstrate the CONTENT wireless virtualization system efficiency, by means of system utilization and throughput performance. The NITOS wireless testbed [76], is used to perform the following experiments. Our goal is to demonstrate that the use of virtualization techniques, can offer increased system utilization and fairness between MOVNO's flows, without sacrificing end user's QoE.

The experiment setup is presented in Fig. 2.5. We assume a system with two MOVNOs, *A* and *B*, each having two associated users, subscriber 1 and 2, all in the same geographical area. (S1:) In the first setup, two 802.11n Access Points (AP) operate without virtualization capabilities; AP_1 serves *A*'s users, and AP_2 serves *B*'s users (for both APs, subscriber 1: $\sim$ 2m distance and subscriber 2: $\sim$ 9m). (S2:) In the second setup, both the 802.11n APs operate with virtualization capabilities and each AP is able to serve both the *A*, *B* users. In this case, the association with some AP, is made on a signal strength/distance basis (we associate the second user with the closest AP). (S3:) In the third setup, a single 802.11n AP (AP_1), operates with virtualization capabilities, servicing all the users from both the MOVNOs.

In every experiment, Icarus indoor testbed nodes (APs) were used, each equipped with i7-2600 processor, 8M cache, at 3.40 GHz, 4G DDR3 RAM, Atheros 802.11a/b/g/n (MIMO) wireless interfaces, 1 Gbps Ethernet interfaces, while we operated the system in 2.4Ghz band. The virtualization scheme, was on a flow basis, without violating the DCF operations. We note that in all the experi-



(a) Number of VNets effect (b) TCP traffic (c) UDP traffic

Figure 2.6: Algorithm performance in different scenarios

ments, the APs where tuned for operation in a different channel, while all the experiments where contacted under heavy interfering conditions (other experiments were running in parallel in the testbed environment).

In Fig.2.6(a), we present the throughput performance, for all setups described, for a single flow: the aggregated throughput for MOVNO A's subscribers. In all cases max TCP traffic was generated using *iperf* tool. As we can see, in the case of virtualization, where we use two 2 vAPs, there is a slight decrease in performance ($\sim$ 20Mbps). When we use a single AP for both flows and all the subscribers, this reduction is even larger ($\sim$ 45 Mbps on average, since in this case 4 users share the channel). Nevertheless, even in this simple set of experiments, there is a great reduction on the throughput difference between the aggregated MOVNOs' flows. This phenomenon can be seen in Fig.2.6(b), where the results of *iperf* tests are presented for TCP traffic and in Fig.2.6(c), where the results of *iperf* tests are presented, for UDP traffic. In the Y-axis the throughput difference is presented for flows A and B. Even if MOVNO A, experience a small reduction on the total throughput his subscribers enjoy, this "extra" throughput is utilized by MOVNO's B subscribers, in a way where two MOVNOs flows are more balanced.

In cluster environments, because of the stochastics on the wireless medium and the great channel quality variations, it is very common that in a single-AP, single-provider scheme, the traffic from a single provider may dominate. Furthermore, the actual throughput per flow is not guaranteed. The idea is that wireless network virtualization, together with SDN control on the wired/optical networks can offer to cloud architects the mechanics to provide balanced throughput performance and service differentiation between different provider flows.

2.5.2 End-to-End QoS analysis

We begin by noting that even in such a complex system as the one we describe, with multiple operational layers both vertically and horizontally, a convergent, packet-based Internet Protocol (IP) network offers the foundation, over which all virtual networks operate. In the network architecture described in Fig. 2.4, essentially IP packets traverse the path from server machines up to the end users, connected to wireless Access Points. In all the network elements, forwarding instructions are based on the notion of flow, which consists of all packets sharing a common set of characteristics. Possible criteria used to define a flow, include the switch port where the packet arrived, the source/destination

MAC address, source/destination IP address, VLAN tagging, or other packet characteristics. In the backbone network, all the research activities focus on the limitations imposed by FIFO scheduling in the routers. Techniques like WFQ [47],[82] and WRED[83], have been effectively used to provide differentiated services in the backbone and are supported by all network vendors.

In the multi-domain SDN model that we consider, every flow is associated with a VNO. Thus all the traffic from all the subscribers associated with a VNO, constitutes the VNO's end-to-end flow. For every subscriber (a subscriber can be a single client, an enterprise and so on), the traffic carried by each flow, originates (terminates) from(to) end users applications or end stations in the data-center. Every application has different performance requirements by means of service needs in bandwidth, packet loss, delay, jitter etc.

As described in [84] the networks capability to deliver services is categorized in three service levels: *best effort*, *differentiated services* and *guaranteed service*. Our focus stays on differentiated services and guaranteed services. Actually, packet forwarding is not the problem in today's networks, even in multi-domain end-to-end setups. Legacy networking techniques are still extremely good at moving packets from the source to the destination. In addition, multiple techniques are available to provide QoS in Layer 2 and Layer 3 in all the domains of the architecture. For example, In Layer 2 IEEE 802.1p offers the functionalities to enable service differentiation (using a 3-bit Priority Code Point (PCP) field, also known as class of service (CoS), within an Ethernet frame header when using VLAN tagged frames), 802.1e is used for QoS in 802.11 networks, or for guaranteed services Subnet Bandwidth Manager (SBM), ATM Constant Bit Rate (CBR) are used. In Layer 3 when applying diff-Serv the IP DS field, Differentiated Services Code Point (DSCP) field is used to define priorities. CoS Committed Access Rate (CAR), Weighted Fair Queuing (WFQ), Weighted Random Early Detection (WRED) are used for differentiated services and Resource Reservation Protocol (RSVP) is the signaling protocol to establish end-to-end paths with guaranteed services. In metro networks MPLS QoS techniques are utilized that facilitate IP QoS or TSON scheduling functions, like the one proposed in CONTENT TSON-based optical networks [25]. We also reference SDN OpenFlow techniques that are able to use QoS and service differentiation, by using HTB scheduling and Linux TC functions[1] and is referenced as both Layer 2 and Layer 3 mechanism. Note that since in multi-domain internet based environments, diverse Layer 2 technologies are used, end-to-end QoS can be delivered only on the network layer protocol and inter-working with Layer 2 techniques is required.

In this thesis, we rely on existing service differentiation techniques for the backbone network and we identify two segments where new forms of control will have to come into effect, in order to provide end-to-end guaranteed differentiated services:

- The wireless access domain: Due to the stochastic nature of the wireless channel, interference and multiple access on the wireless domain, every VNO flow actually receives no true guarantees.

- Data-center server operations: Similarly, on the data-center due to multitenancy effects when utilizing server resources, a stochastic parameter is introduced and again there are no accurate guarantees that can be provided.

Indeed although the application type can be used to differentiate traffic efficiently in an end-to-end fashion by available technologies, service differentiation has not efficiently addressed for the edge of the multi-domain architecture, in the server machines and the wireless access. The reason is that both of these domains, are highly affected by stochastic disturbances, that are very difficult to control efficiently and in an accurate way. These disturbances can lead to unpredictable service delivery degradations. This realization leads to the need of re-evaluation of existing mechanisms and a different approach on applying QoS control at both edges of the architecture. For example, in the wireless domain, since channel capacity may greatly vary with time and because of the DCF operations in 802.11 networks [85], no service delivery contracts for every virtual network flow can be actually guaranteed, without the application of stochastic control. On the other edge of the architecture, on the data-center segment and more specifically in server operations, dynamic control is required in order to perform guaranteed service delivery between different flows. Although packets may arrive in a server system with some guaranteed prioritization, under stochastic workloads with unknown statistical knowledge of the distributions regarding the arrival and service process, is hard to provide guaranteed service differentiation between different flows.

In the analysis that follows in next chapters, we present new stochastic closed-loop control policies that are able to alleviate the problem of differentiating services, according to SLAs between cloud providers and cloud users. By utilizing queueing theory and stochastic control theory, in both edges of the architecture and by assuming a guaranteed path from one edge of the architecture (the wireless domain) to the other edge (the data-center), we can actually engrave the trails so that end-to-end service guarantees between different flows, come into effect. A service defines some metrics, such as throughput, delay, jitter, and packet loss that are used to quantify the QoS while in addition, a service can be characterized in terms of relative priority of access to network or processing resources. Another reason for service delivery degradations, is related to problematic policy and operational layers on top of traditional networking. These cause problems and slow down operations, but their study is out of scope of this thesis. All the approaches that we develop in chapters 3 and 4 can be applied towards this goal, while because of their generality can be applied in various application scenarios. In Chapter 5, we evaluate the proposed end-to-end architecture from the application perspective, in the case where multiple cloud operators build content distribution services, over the multi-domain virtualization system.

2.6 Chapter Conclusions

In this chapter we presented the design of a novel SND network architecture that tries to overcome the challenges imposed by virtualizing and integrating wireless, optical and data-center networks, in both the data and control planes, in a vendor and protocol independent way. The proposed CONTENT solution serves as the ideal ground-floor to apply policies that are able to offer guaranteed service delivery to different virtual traffic flows. The solution is aligned with the ONF/SDN approach, where control plane functions operate on top of programmable devices, in our case represented by the virtual network resources. The virtualization layer in the proposed architecture: a) generates isolated virtual infrastructures, b) provides an abstracted view of the different domains, c) hides the vendor-dependent details while exposing the full technology capabilities through a unified interface.

At the control plane level, on top of the virtual infrastructures, the wireless, the TSON and the data-center virtual domains are managed through dedicated lower-layer control frameworks: the wireless and the TSON Control Planes. These are customized to deal with the specific technology constraints (e.g. management of 802.11 Access Points, server systems) and implement a set of basic intra-domain mechanisms, like connectivity setup and monitoring. With reference to SDN architectures, the local control planes act as a sort of macro SDN controllers, responsible of the basic network functions in each segment. On the other hand, all the end-to-end enhanced functionalities for on-demand provisioning, maintenance, resiliency, monitoring and dynamic re-configuration of network connectivity in support of mobile cloud services are delegated to upper layer network applications.

Regarding our current research activity, the necessary data structures are being defined, describing the wireless virtualized resources, in terms of capabilities and availability in the corresponding models. Besides the necessary data-types and the operations that can be performed in both the wireless and wired resources, a notification modeling system is required to build the necessary interfaces for events like lost connectivity etc. A similar procedure is followed for the management of the LTE network, in order to provide for example the connectivity services. In addition, and towards 5G's holistic approach, network abstraction and virtualization will serve as key enabling technologies, for delivering consistent and enhanced end-user Quality of Experience in a highly heterogeneous environment. Our efforts are also placed towards enhancing the system with advanced data-plane/control-plane functionalities and Network Functions Virtualization services, used for example for building NFV-radio heads.

Chapter 3

Dynamic Weighted Round Robin Analysis for Service Differentiation

In this chapter, we present a class of Dynamic Weighted Round Robin (DWRR) scheduling algorithms, that is able to provide guaranteed service differentiation, between competing customer classes. The framework we develop is applicable in various control points of the multi-domain architecture presented in Chapter 2 (e.g http routers, wireless driver queues, server systems) and various problems where guarantee service is required per customer class. Stochastic analysis and queueing theory are the tools that we use to prove a number of theorems and lemmas, regarding steady state analysis and properties like speed of convergence.

Although the framework we develop is generic, the analysis we present focuses on the CPU power of server systems, as the resource of interest. The reason is that, in contrast to metrics like delay, CPU utilization is a metric that is easily observable, even in virtualized environments and CPU power is the dominant resource that determines the server/system performance. Therefore we believe our study is critical for many applications and systems.

3.1 Introduction

In complex systems, like the one presented in chapter 2, the infrastructure provider can agree with the clients to offer a different service quality level depending on the system load. The main idea is that under load stress, the system cannot effectively guarantee the same level of performance as the normal load metrics describe. Although it is quite challenging to offer guaranteed end-to-end delay performance to each customer class, in practice delay is not a metric that an administrator can guarantee. In principle, in such a highly dynamic environment, fine-tuned metrics like delay, response time or availability, are subject to uncontrollable parameters like overall (physical and virtual) server load, multi-tenancy effects, output and input traffic in queueing services or general network issues. An SLA with alternative metrics of performance must be clearly chosen.

In this work, we propose an SLA with guarantees on CPU sharing provided to the users or classes of users. An example SLA could be cast along the following lines: "*without any knowledge of statistics regarding the arrival and the service process in a cluster of servers, guarantee* 20% *of CPU power to requests of class A,* 30% *of CPU power to requests of class B and* 50% *of CPU power to requests of class C in the long run. Also assume that no statistical knowledge is available regarding the correlation between the request size and the service time*".

The main motivation behind this work and the selected SLA, is that in many applications scenarios, (like in VM scheduling, web servers operations, scheduling in the access tier of a CONTENT architecture, or Mobile Cloud Computing (MCC) operations), it is not easy to correlate actual processing time with the type of request or with the request/packet size. In addition, statistics about the inter-arrival and service time for a customer's requests are, in general, unknown in practice and may not be easily estimated or correlated to metrics that are easily observable in a *service processing* environment. For example, parsing and processing a small XML file may need more CPU cycles than a huge XML file, depending on the service requested and the iterations needed [86] [87]. Another example is VM scheduling in the hypervisor, where we want to guarantee a specific percentage of CPU power to each VM in the long run, without having a priori knowledge of the workload density, related to VM activity.

Although CPU provisioning is a very well investigated topic (e.g., in OS operations and thread schedulers [41], hypervisor operations [88], [89], [42], [90], queueing systems [43]) the importance of dynamic control schemes that are able to guarantee CPU performance/provisioning, has emerged again because of the increased complexities in today's virtualized cloud-based environments. With empirical data about the service time, when estimating the moments from the sample, we can derive a good estimate for the mean, but the estimates for variance and higher moments are not accurate [91]. As we will present, with the use of feedback-based stochastic control, we avoid dependencies on prediction errors on the service process evolution [61] and large dependencies on the history of the system.

Note that a "desirable" scheduling policy has several properties:

a. It achieves the technical SLA (*T-SLA*), meaning that it guarantees specific CPU utilization ratios.

b. It is agnostic to the arrival and service statistics.

c. It converges to the *T-SLA* fast, and

d. Requires a small number of calculations per time unit.

Apart from (a) which is obvious, the knowledge of service statistics (b) and the decision load (d) are both related to the communication overhead and CPU costs and are very important considerations for practical systems. Also (c) is crucial for achieving the target within short periods. To the best of our knowledge, no single scheduling policy exists that is superior in all these properties; we investigate policies that excel in some of the above criteria and give the designer the ability to trade off in order to satisfy the rest.

In this chapter, we define a class of "bound round-fair (negative drift)" Dynamic Weighted Round Robin policies, that can be used to provision not approximate but exact CPU cycle percentages between competing users in overload, while they fulfill a number of design criteria. In more detail, we prove that under some statistical assumptions, all the algorithms of this class have properties (a) and (b) and are able to trade off for (c) and (d). Furthermore, they can be used in a system operating both under "normal load" and overload conditions.

In the following text, with saturated arrivals we mean the special case where, at each control instant, requests are pending for service by all the domains and so there is no idle time in the system (infinite queues). In addition, following the classical terminology, a work-conserving policy is a policy that does not depend on the service time distribution and when some queue has available requests the server does not stay idle, regardless of the runtime statistics and other statistical knowledge [90]. In practical systems, the distinction between work-conserving and non work-conserving modes of operation is of paramount importance. The reason is that the actual performance is largely affected by the way service-capacity redistribution is made in run time. For example, in hypervisor technologies the selection of the mode can be made by tuning configuration parameters (e.g setting the "cap" in the XEN hypervisor, when using Credit scheduling [90]).

The contributions of the present work are the following, regarding the analysis of "bound round-negative drift" Dynamic Weighted Round Robin policies:

- In the general case of arbitrary arrival and service time distributions, under non work-conserving mode of operation, we prove that under the class of DWRR policies exists, the system not only regulates, but in the long run convergences to the goal.
- We define the feasibility space of the class of algorithms, for both of work-conserving and non work-conserving modes of operation.
- Under work-conserving mode of operation, we prove that the same class of DWRR policies can satisfy a minimum guaranteed service.
- We evaluate the dependence on the service redistribution mechanism under various performance metrics.
- We show that minor variations of the policies can be used in order to reduce overhead, without sacrificing convergence to the goal, but tradeoff with slower speed of convergence.
- In the special case of saturated arrivals we prove that all policies experience Cauchy convergence, the system experience no oscillations in the long run and also can be used to guarantee an exact CPU share for each domain.
- In the saturated arrivals case, we establish theoretically their sub-linear rate of convergence.

Currently, under stochastic workloads and bursty load, the analysis of the repercussions in performance of the service redistribution mechanisms are unknown, depending on multiple factors like I/O dependencies. Percentages are not actually guaranteed, rather there is a minimum performance threshold satisfaction, that is verified only by simulations [90]. The analysis we present, can exert a measure of control in such and also other concepts where dynamic prioritization is required. The system model we examine is generic and thus it can be applied, with some necessary modifications, in multiple application scenarios. We formalize this problem in the following section, using an abstract system model.

The rest of the chapter is organized as follows. In section 3.2 we present the related work. In section 3.3, we describe the system model, the detailed provisioning objectives and the corresponding mathematical model. In section 3.4, we define a class of DWRR scheduling policies and in section 3.5, we provide the theoretical framework regarding convergence and optimality of the defined class. In section 3.6 we present results regarding implementation considerations like speed of convergence. In section 3.7, we evaluate the performance of the proposed policies. We conclude our study and present future research directions in section 3.8.

3.2 Related Work

At a first glance, this type of SLAs may seem trivial to satisfy. In principle, however, when we cannot use knowledge about the arrival/service process, static algorithms are not suitable. Note that our study focuses on *overloaded* environments. In the case of underloaded systems (with total utilization less than 1), in a work-conserving system the utilization every class will receive, is equal to $\lambda_i \cdot E[S_i]$, where $E[S_i]$ is the mean service time of class i and λ_i its arrival rate, independently of the goal vector. Then let, for simplicity, $m = 1$ be the number of servers in an overloaded system and D the set of classes. It is well known that under a Weighted Round Robin (WRR) policy employing weights w_i, the utilization U_i^{WRR} of a given class, will converge to the constant $U_i^{WRR} = \frac{w_i \cdot E[S_i]}{\sum\limits_{j \in D} w_j \cdot E[S_j]}$[1]. Then, to achieve the goal under WRR (or similar policies), one needs to set the weights w_i such as U_i^{WRR} equals the goal utilization and solve a system of linear equations to find the weights. In the case where we have no statistical knowledge of the arrival and service process statistics, this approach cannot be applied.

First note that the closed-loop system we examine is nonlinear, with stochastic terms that define the input-output relationship. We explain this point in detail in the following analysis. In order to establish convergence conditions, results from ergodic theory and martingale theory have been applied in the past [52]. For example, the martingale theory has been used to prove convergence in adaptive, self-tuning systems, based on stochastic approximation estimations, where the random disturbances affect the state equation. The martingale theory is related to betting strategies and fair games [92], [93] (e.g a gambler doubles his bet after every loss), where in the case of super-martingales the current observation of the utilization is an upper or lower bound on the conditional expectation (i.e, a discrete-time super-martingale satisfies $E[U_{n+1}|U_1, \ldots, U_n] \leq U_n$). Results exists which state that a bounded super-martingale convergences.

In addition to the related work referenced in the introduction, our analysis is also related to the large family of GPS/WFQ scheduling principle [47], [65], [94] and Deficit/Weighted Round Robin

[1]According to the Strong Law of Large Numbers.

schemes [95], [96], [97] and [98]. For example, in [98] fair scheduling is applied and a weight is assigned to each thread based on its priority, where fairness seek to bound the positive and negative processing time lags. The token bucket shaper and weighted fair-queueing are used to guarantee that the packet delay across a network can be guaranteed to be less than a given value. Weighted Round Robin [99] and approximations of the proportional-share GPS/WFQ [82] in which weights are assigned statically, in the case where the statistics of job service time are unknown, are not able to provide the required differentiation and the same conclusion holds true for probabilistic/mixing policies [43], [100], that need to characterize the performance space first. Furthermore, static open-loop schemes can only reach arbitrary defined targets, in the case when the arrival and service process are known in advance; this is not the case in complex service-based end-to-end architectures that we examine. Also, an approach based on service time predictions according to system macro behavior, rather than the micro behavior, is subject to prediction errors [61] and usually is unable to provide predictable services.

All the *negative drift* policies use the same concept of dynamic weight allocation in DWRR [98]. In [98] the authors update the threads round slice, based on new weights, according to the deviation that the lag (processing time) had from the WFQ goal. We use ideas similar to WFQ and DRR with "queue skipping" based on some current metric, but in contrast with max packet length our "skipping" criterion is different, the goal is to achieve a different objective (CPU cycles share not throughput) and we also use a different notion of round. Also we provide not an algorithm but a class of algorithms that can be used to guarantee CPU shares. We show that the weight assigned can be an arbitrary, finite number greater than one, meaning a round can have multiple requests served per queue, without any calculations required for example for Quantoms [95] to quantify how far we are from the goal.

Studies in the area of guaranteed CPU performance through scheduling, have been extensively applied in cloud computing environments, by hypervisor technologies [88], [89], [42], [90], [101]. For example, the Xen platform uses Credit scheduling, which is the successor of BVT and SEDF[101] schedulers [42], [90] and VMware ESX server operates with a custom scheduling mechanism that uses the concept of reservations and shares [88]. In the VMWare approach, a "reservations" parameter specifies the guaranteed minimum and maximum CPU allocation and a "shares" parameter measures the current configured amount of shares of the VM (a MHzPerShare metric based on the current utilization of the virtual machine). Note that none of these techniques, although widely deployed, are based on a rigorous analysis regarding the proof of convergence to the goal. Because of the system dynamics, it can be very hard to produce well-defined policies to achieve the differentiation goals [90]. In addition, the actual performance in such systems, is heavily depended on I/O priorities and preemptive operation of the OS thread scheduler. Proportional differentiation guarantees by means of slowdown, are investigated in [61] where a feedback based mechanism is required to adjust processing rate using integral control; online measurements with prediction and resource allocation techniques is examined in [65], where a model that dynamically relates the resource requirements of each application to its workload characteristics is presented, but for response time satisfaction per customer. Related work in CPU power management policies for service based applications can be found in [102] and [103]. In [102] priority scheduling is proposed, with a penalty function that is used to adjust throughput per client; in [103] the authors propose a method for estimating CPU demand of service requests, based on linear regression between the observed request throughput and resource utilization level. Our work is also related to the emerging virtual network embedding problems [104], [105] and the resource allocation problems that arise due to physical limitations on CPU, network capacity etc. when sharing resources between in virtual tenants.

3.3 System model & Problem Statement

3.3.1 System model

We assume the system model depicted in Fig. 3.1. The system comprises a set of service domains, $\mathcal{D}$, indexed by $i = \{1,\ldots,D\}$ and a set of servers, $\mathcal{M}$, indexed by $m = \{1,\ldots,M\}$. Each service domain is associated with a given class, which specifies a percentage p_i of aggregate CPU resources to be committed for serving domain requests over an infinite time horizon. A router/controller with one FIFO queue per domain, distributes service requests from the domains to the cluster of servers, seeking to uphold the domain SLAs in the process. The controller has no a priori knowledge of the execution time of the requests. This parameter is provided a posteriori as feedback, in order to schedule future service requests more effectively. It is clarified that a service request is modeled as a non-preemptive process that will occupy a server for a given, non-infinite amount of time.

The following modeling assumptions are adopted, without loss of generality:

- We assume that the servers have no queueing capabilities. This task is handled exclusively by the controller.
- Whenever a server dispatches a given service request, it signals the controller for a new request/packet. The controller then assigns a new request to the server. This communication is instantaneous. In other words, the communication overhead between the controller and the servers is assumed to be a part of the requests.
- If all domains queues are empty, a server is fed with interruptible null-requests from a null-domain (Fig. 3.1). The null-domain is an artificial structure, employed as a means of facilitating the mathematical analysis only.
- Every domain can be served by any server.

Every domain i has a finite, aggregate arrival rate of requests, λ_i, which follows an unknown distribution with bounded inter-arrival times. Regarding the service process, we assume a non-i.i.d model, where the service time distribution can vary depending on the domain identifier, i. The service time of a request belonging to domain i, is described by a random variable S_i that follows some general distribution with mean $\mathrm{E}[S_i]$ and finite variance, that we assume to be bounded. Therefore, there exists some finite S_i^{max} such that:

$$0 < S_i \leq S_i^{max} < \infty \tag{3.1}$$

We note that the bounded inter-arrival times and the bounded service times, are the only requirements for the proposed mathematical framework. We define the CPU utilization, $U_i^{\pi}(t)$, of domain i up to time t, under a request-to-server assignment policy π as:

$$U_i^{\pi}(t) \triangleq \frac{\textit{total service time up to t}}{M \cdot t} \tag{3.2}$$

When the following limit exists, the system is said to have reached a *steady state*, and the *allotted percent of CPU capacity* to domain i is then defined as:

$$\widetilde{U}_i(\pi) \triangleq \lim_{t \to \infty} U_i^{\pi}(t) \tag{3.3}$$

The maximum achievable utilization for domain i, obtained when all of its requests can be served in the cluster for $t \to \infty$ is:

$$\tilde{p}_i = \frac{\lambda_i \cdot \mathrm{E}[S_i]}{M} < 1 \tag{3.4}$$

This can be thought of as the utilization obtained if there was a single domain in the system, utilizing all the resources. We note that servicing all the requests, does not mean that the queue will be empty from one point and on, but rather that the probability of noticing queue sizes bigger than zero is negligible (convergence in probability).

Furthermore, let l_n denote the idle time between round n and round $n+1$. Idle time can occur because there are no available requests in the system, or the system operates in non work-conserving mode and the available requests are not served because of the policy rules. We assume only systems for which

$$l_n < l < \infty, a.s., \tag{3.5}$$

where $l \in \mathbb{R}_+$ denotes the maximum observable idle time (or maximum observable inter-arrival time).

Finally, we denote as p_i the steady state percentage utilization goal, and the corresponding goal vector as $\mathbf{p} = (p_i)$, where $\sum_i^{|D|} p_i = 1$. The goal vector simply defines the requested percentage share of the aggregate CPU time per domain, eg. $\mathbf{p} = (20\%, 30\%, 50\%)$, *assuming steady state*.

3.3.2 Objectives

Our objective in this study, is to define control policies π that fulfill a number of criteria. We investigate policies that uphold the SLO for every domain i, under the assumptions of stochastic arrivals and unknown service time distribution. No workload prediction/estimation is assumed. The reason is that we want the mathematical framework to be general and depending on the context of the application, predictions may be difficult to be derived (e.g., in web server applications, when thousands of services are deployed with completely different semantics). In the following, we provide the following:

a. Analysis of the conditions for the existence of a steady state (i.e. that the limit in eq. (3.3) exists).

b. Analysis of the feasibility space of the class of policies defined and the analysis of the redistribution mechanism of the residual CPU service time required in order to achieve any custom but

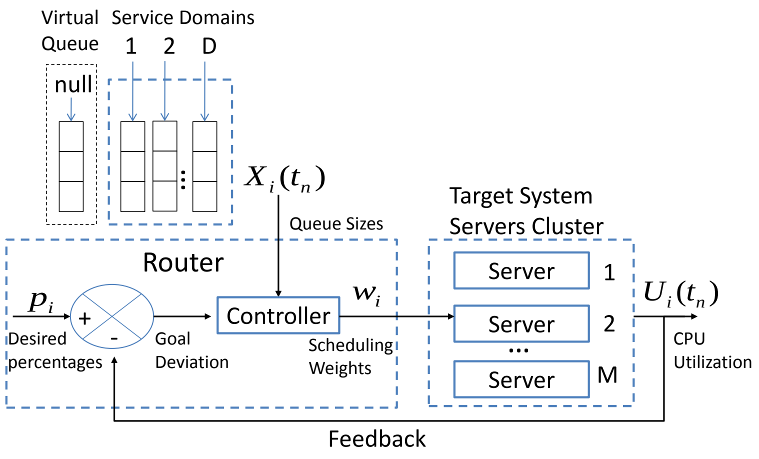


Figure 3.1: Overview of the system model, comprising a set of SLA-specific service domains and a set of servers. A controller enforces the SLAs using the reported processing times as feedback.

allowed steady state p_i^*. It will be shown that the redistribution mechanism greatly affects the actual percentage achieved.

Taking into account the redistribution process, the objective (SLO) can be expressed as equations of the form:

$$\widetilde{U_i}(\pi) = \begin{cases} \tilde{p}_i & \text{, if } \tilde{p}_i \leq p_i \\ p_i^* & \text{, if } \tilde{p}_i > p_i, p_i^* \in [p_i, \tilde{p}_i] \end{cases} \tag{3.6}$$

An exemplary SLO can be described as follows: “*without any knowledge of statistics regarding the arrival and the service process in a cluster of servers, guarantee* 20% *of CPU power to requests of class A,* 30% *of CPU power to requests of class B and* 50% *of CPU power to requests of class C in the long run. Also assume that no statistical knowledge is available regarding the correlation between the request size and the service time*”.

Note that in the above definition, there is no distinction between environments where mixed workloads are present and environments where the system is always saturated, meaning that there are always available requests by all the domains in the controller queues. In the case of dynamic arrivals, it is up to the policy to decide how to redistribute service, in the case where a high priority domain must be served but has no available requests.

We will show that a class of policies Π exists such that every $\pi \in \Pi$ achieves steady state in the saturated case and in the non work-conserving mode and upholds the domain SLOs, since the limit in equation eq. (3.3) exists and the differentiation objective in eq. (3.6) is satisfied. We also study the role of the redistribution mechanism in convergence of the policies in the work-conserving mode. In addition we present a thorough analysis on the feasibility space of the policies, deducing the achievable service percentages, p_i^*, for every domain.

3.4 The Proposed, Domain-Serving Policies

We propose a class of Dynamic Weighted Round Robin policies, where the idea of a round is used again, but the number of times each domain is served is controlled in a dynamic way; round by round a decision is made as follows. At the beginning of a new round, a list of domains accompanied by weights w_i^n for every domain i is selected by the controller. Then, all the domains in the list are scheduled for service that many times as the weight indicates. When the round is over, a new list as well as new weights are calculated. We investigate Weighted Round Robin policies due to their nice speed in convergence properties in contrast with probabilistic solutions [106].

3.4.1 Class Π of Dynamic Weighted Round Robin (DWRR) policies

We define the class Π of DWRR policies, which operates at rounds, with no loss of generality. In order to enable the work-conserving property, we decompose Π into two algorithmic components, that we denote as f and h respectively. Any policy $\pi \in \Pi$ obeys the following rules regarding f, while any arbitrary h algorithm can be used to enable the work-conserving property and support the redistribution mechanism.

At the beginning of round n, at time instant $t_n \in \mathbb{R}_+$, let $X_i(t_n)$ denote the queue size for domain i. We define:

$$\text{Class } \Pi = \begin{cases} \text{If } \exists i : U_i(t_n) < p_i, X_i(t_n) > 0 \rightarrow \textit{Algorithm f} \\ \text{If } \forall i : U_i(t_n) < p_i, X_i(t_n) = 0 \rightarrow \textit{Algorithm h} \end{cases} \tag{3.7}$$

The *algorithm f (negative drift)*, calculates the weights w_i^n as follows:

$$w_i^n = \begin{cases} 0, \text{if } U_i(t_n) > p_i \text{ or } X_i(t_n) = 0. \\ \min\{k(t_n,\pi), X_i(t_n)\}, \text{ if } U_i(t_n) \le p_i, X_i(t_n) > 0 \end{cases} \tag{3.8}$$

In other words, w_i^n job requests are scheduled for domain i, where $k(t_n,\pi) \le K$ and K is an arbitrary finite positive integer.

The *algorithm h (work-conserving)* is used whenever $X_i(t_n) = 0, \forall i : U_i(t_n) < p_i$, in order to enable work conservation. In case where there are no available requests from all under-served ("suffering") domains, the class of policies Π can utilize any arbitrary scheduling decision, as long as there are available requests to do so. For example, the algorithm can choose one domain j at random from the domains for which $X_j(t_n) > 0$ and serve it for a finite number of times.

This algorithmic decomposition defines a class of policies for two reasons. Firstly, regarding the general operation of the policies (Algorithm f) there is no restriction on the number of requests served per round, as long as over-satisfied domains are blocked when there are no available requests from a suffering domain. The only requirement is imposed by the artificial limit K, used to avoid the case of assigning an infinite number of requests to some domain. Secondly, there is no limitation, on the operation of the h-algorithm, used for enforcing work conservation via redistribution of residual service slots. In general, the f-algorithm of the scheduling class is used to guarantee the objective defined in eq. (3.6) and define the feasibility space, while the h-algorithm defines the actual state achieved within the feasibility space.

3.5 Theoretical Analysis of DWRR

Under any policy $\pi \in \Pi$ (either work-conserving or non work-conserving), the CPU utilization evolves in time according to the following recursive formula:

$$U_i(t_{n+1}) = a^n \cdot U_i(t_n) + b_i^n \tag{3.9}$$

where $a^n = \frac{t_n}{t_{n+1}}$, $b_i^n = \frac{S_i^n}{t_{n+1}}$ and S_i^n denotes the aggregate service time that domain i received during round n, expressed as a sum of service time random variables. This recursion is aligned with the classical feedback ARX models [44], where in this case both terms a^n, b_i^n are stochastic and thus utilization is also a random variable. In fact, this is the reason that stochastic analysis tools are required to effectively control the system, when the arrival and service process are unknown. Term a^n defines the way the history affects the system and b^n is related to the control input. Note that because of the bounded service-time and idle time assumptions, any round duration is also bounded. In addition, for any domain, the increase (or decrease) in utilization is also bounded according to the following lemma.

Lemma 1 (Utilization difference)**.** *The utilization difference between the beginning and the end each round for every domain is bounded according to*

$$|U_i(t_{n+1}) - U_i(t_n)| \le \frac{l + K \cdot S_i^{max} + \sum\limits_{j \in \mathcal{D}} K \cdot S_j^{max}}{t_{n+1}} \tag{3.10}$$

Proof of lemma 1. By definition S_i^n denotes the service time received for domain i within the time interval $[t_n, t_{n+1}]$. Then using eq. (3.9)

$$U_i(t) = \frac{t_n}{t_{n+1}} \cdot U_i(t_n) + \frac{S_i^n}{t_{n+1}}$$

and hence

$$|U_i(t_{n+1}) - U_i(t_n)| = \left| \left(\frac{t_{n+1} - t_n}{t_{n+1}} \right) \cdot U_i(t_n) + \frac{S_i^n}{t_{n+1}} \right| \tag{3.11}$$

Then

$$\begin{aligned}
|U_i(t) - U_i(t_k)| &= \left| \left(\frac{t_{n+1} - t_n}{t_{n+1}} \right) \cdot U_i(t_n) + \frac{S_i^n}{t_{n+1}} \right| \\
&\leq \left| \frac{t_{n+1} - t_n}{t_{n+1}} \right| \cdot U_i(t_n) + \left| \frac{K \cdot S_i^{max}}{t} \right| \\
&\leq \left| \frac{\sum\limits_{j \in \mathcal{D}} K \cdot S_j^{max}}{t_{n+1}} \right| + \left| \frac{K \cdot S_i^{max}}{t_{n+1}} \right| \\
&= \frac{K \cdot S_i^{max} + \sum\limits_{j \in \mathcal{D}} K \cdot S_j^{max}}{t_{n+1}}
\end{aligned} \tag{3.12}$$

In eq.(3.10) we additionally included the term l (an upper bound for all l_n), since idle time may be experienced due to lack of available requests from all the domains. In this case it holds that $t_{n+1} - t_n = l_n + \sum\limits_{i:w_i^n \geq 1} S_i^n$, where l_n denotes the sum of "null requests" duration in all the servers, between round n and $n+1$. $\square$

A standard way of proving the convergence of a work-conserving policy is to show that eq. (3.9) converges in the general case. This essentially means that the limit of eq. (3.3) exists and that the system will reach a steady state. The form of eq. (3.9) may predispose for convergence, since the control action has waning effect on utilization as time progress ($b_i^n \to 0$ and $a^n \to 1$ for $t \to \infty$). Before we proceed with the analysis, note that although the utilization evolution in transient state heavily depends on the time evolution and the past control actions, time evolution has no actual effect on convergence. It is the combination of the selected $\pi = (f, h)$ policies that is responsible for the long-run behavior. As we mentioned in the related work section, results from ergodic theory and martingale theory have been applied in the past [52]. For the system we study we tried to exploit sufficient conditions for almost sure convergence to a limiting random variable. The reason is that a martingale in the limit converges, according to the analysis and proofs presented in [107]. Nevertheless, for this classic approach to yield convergence, when trying to apply it in our system, if we do not take into the account the negative-drift characteristics of the policies, the error is a sum of terms whose individual limit to infinite is zero. This sum reassembles the properties of harmonic series (with terms that tend to zero) and the sequence actually diverges, since it tends to infinity.

In conclusion, in a work-conserving system following the f negative drift principles, although as the time progresses the control actions have waning effects, it is up to the redistribution policy h if convergence can be achieved in the Cauchy sense [52]. In the analysis presented in the following sections, we take a simpler approach and we exploit results from stochastic analysis in order to prove convergence.

3.5.1 Convergence and Feasibility Space Analysis

In the general case of unknown arrival process, $\tilde{p}_i$ (the maximum utilization percentage achieved in steady state according to eq. (3.4), may be higher, equal or lower than the percentage goal requested. A number of lemmas and theorems establishes the feasibility ("capacity") region formally, for every policy $\pi \in \Pi$.

Non Work-Conserving Mode: Convergence w.p1

As the following theorem presents, under non work-conserving mode, every negative drift DWRR policy reaches steady state. In addition, the capacity region is a vector.

Theorem 1 (Convergence when non work-conserving)**.** *In non work-conserving mode of operation, under any policy $\pi \in \Pi$, the limit in eq. (3.3) exists $\forall i \in \mathcal{D}$. Moreover the objective defined in eq. (3.6) is satisfied. Thus:*

$$\widetilde{U}_i(\pi) = \min\{\tilde{p}_i, p_i\} \tag{3.13}$$

In the non work-conserving mode, the policies operate according to the negative drift rules of algorithm f (i.e., we have $\pi = f$) and no service redistribution is allowed. If there are no available requests from all the domains that are below their goal, the system goes to idle state. Essentially, the class only requires to block over-satisfied domains at the control instances and serve one or more of the under-utilized domains. In addition, it imposes no restriction on the number of requests that are served during a round.

The proof of convergence is based on the following observation. Because of the negative drift given to the over-satisfied domains at every control instant, in transient state, the utilization trajectory for any domain oscillates around its goal percentage p_i. Since the utilization difference between any two successive rounds is bounded, the oscillation is continuously decreasing and there exists some point in time when the system converges in the Cauchy sense. In the case where $\tilde{p}_i < p_i$ then we prove that the trajectory approaches $\tilde{p}_i$.

To proceed with the proof, we introduce the null-domain concept of Fig. 3.1. Essentially, we model the idle time as a null-domain, and treat the system state as a saturated case variation.

Initially, since $\limsup_{k\to\infty} \frac{l+K\cdot S_i^{max}}{t_k} = 0$, we can write that:

$$\limsup_{k\to\infty} p_i + \limsup_{k\to\infty} \frac{l+K\cdot S_i^{max}}{t_k} = p_i \tag{3.14}$$

In addition, the following lemma holds:

Lemma 2 (Utilization upper bound)**.** *For any domain i and round k, the utilization is upper bounded by*

$$U_i(t_k) \leq p_i + \frac{K\cdot S_i^{max}}{t_k}. \tag{3.15}$$

This bound derives from eq. (3.12) and holds for the general case of stochastic arrivals and non-work work-conserving mode of operation. The reason is that all over-served domains are blocked by the policy (even if a number K of its requests were served during the last round).

Proof of lemma 2. First consider a round n for which the utilization is smaller than the goal. Then we readily get

$$U_i(t_n) \leq p_i < p_i + \frac{K \cdot S_i^{max}}{t_n}. \tag{3.16}$$

Next we study a number of consecutive rounds $k_1, \dots, k_\xi$ for which the utilization of domain i is above the goal. Observe that domain i does not receive service within rounds $k_2, \cdots, k_\xi$ since its utilization exceeds the goal (see policy definition). Therefore we can readily conclude that $U(t_{k_\xi}) < U(t_{k_{\xi-1}}) < \cdots < U(t_{k_2}) < U(t_{k_1})$. Therefore it suffices to prove an upper bound only for rounds that succeed a round where the utilization was below goal. For these specific rounds we perform the following analysis.

Suppose k is a round for which $U_i(t_k) > p_i$ and $U_i(t_{k-1}) < p_i$. Let $\tilde{t}_{i,k}$ be the largest instant within round k such that $U_i(\tilde{t}_{i,k}) = p_i$, clearly such an instance must exist. Also, let $\tilde{S}_{i,i}^k$ be the remaining service time of domain i in round k, after the time instance $\tilde{t}_{i,k}$. Then using eq. (3.9) for $t_k < t \leq t_{k+1}$ we derive

$$\begin{aligned} U_i(t) - p_i &\leq \frac{\tilde{t}_{i,k}}{t} U_i(\tilde{t}_{i,k}) + \frac{\tilde{S}_{i,i}^k}{t} - p_i \\ &= \frac{\tilde{t}_{i,k}}{t} p_i + \frac{\tilde{S}_{i,i}^k}{t} - p_i = \frac{\tilde{S}_{i,i}^k}{t} - \frac{t - \tilde{t}_{i,k}}{t} \cdot p_i \leq \frac{\tilde{S}_{i,i}^k}{t} \end{aligned} \tag{3.17}$$

Since $K \cdot S_i^{max}$ is a universal upper bound of service within a round, we have $\tilde{S}_{i,i}^k \leq K \cdot S_i^{max}$ and so

$$U_i(t) \leq p_i + \frac{\tilde{S}_{i,i}^k}{t_k} \leq p_i + \frac{K \cdot S_i^{max}}{t_k}. \tag{3.18}$$

□

Then, since all over-served domains are blocked by the policy (even if a number K of its requests were served during the last round), we can write that:

$$\limsup_{k \to \infty} U_i(t_k) \overset{\text{eq.(3.10,3.14)}}{\leq} \min\{\tilde{p}_i, p_i\} \text{ a.s} \tag{3.19}$$

where $\tilde{p}_i$ is the maximum achievable utilization for domain i. This quantity is related with the maximum-overshoot in classical control theory [44].

Proof of Theorem 1. We introduce the notion of a virtual queue holding *null* requests. The queueing structure of the controller, now becomes one queue per domain, plus one virtual queue for null requests (see Fig.3.1). A null request can be though of as a series of *nop* assembly commands. This is an artificial technicality, to model idle time in the system. Moreover, we assume that null requests always exists in the virtual queue.

In order to examine the case of idle state, say that at the end of some round there are no available requests from any of the domains ($X_i(t_n) = 0, \forall i \in \mathcal{D}$). Then the scheduler selects a null request from the virtual queue and sends it to the requesting server. We assume that the server preemptively replaces null requests. If a *null request* is in service and some domain's request arrives to the system, the controller sends the request to the idle server. Thus the null requests can be treated essentially as an additional domain in the system.

For the ease of presentation we define the following partition of the domains set:

- $\mathcal{D}_1$: includes all the domains for which $\tilde{p}_i < p_i$.
- $\mathcal{D}_2$: includes all the domains for which $\tilde{p}_i \geq p_i$. Then $\mathcal{D}_2 = \mathcal{D} - \mathcal{D}_1$.
- $\mathcal{D}_3$: a singleton set with the null requests domain.
- $\mathcal{F}$: $\mathcal{F} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$.

Now consider that the idle domain receives all the service time that cannot be utilized due to lack of requests from the "suffering" domains in $\mathcal{D}_1$. Let $C(t_n)$ denote the utilization of the null domain, then we can write:

$$\limsup_{k\to\infty} C(t_n) = \sum_{i\in\mathcal{D}_1} (p_i - \tilde{p}_i) \tag{3.20}$$

which is the upper bound for the null requests domain utilization. Then, the following is true:

$$\begin{aligned}
\liminf_{k\to\infty} U_i(t_{k+1}) &= \liminf_{k\to\infty}(1 - \sum_{j\in\mathcal{F}\setminus\{i\}} U_j(t_{k+1})) \qquad (3.21)\\
&\geq 1 + \liminf_{k\to\infty}(- \sum_{j\in\mathcal{F}\setminus\{i\}} U_j(t_{k+1}))\\
&= 1 - \limsup_{k\to\infty} \sum_{j\in\mathcal{F}\setminus\{i\}} U_j(t_{k+1})\\
&\geq 1 - \sum_{j\in\mathcal{F}\setminus\{i\}} \limsup_{k\to\infty} U_j(t_{k+1})\\
&\geq 1 - [\sum_{j\in\mathcal{D}_1\setminus\{i\}} \tilde{p}_j + \sum_{j\in\mathcal{D}_2\setminus\{i\}} p_j + \limsup_{k\to\infty} C(t_n)]
\end{aligned}$$

Case 1: $i \in \mathcal{D}_1$

$$\begin{aligned}
&\liminf_{k\to\infty} U_i(t_{k+1}) \qquad (3.22)\\
&\geq 1 - [\sum_{j\in\mathcal{D}_1\setminus\{i\}} \tilde{p}_j + \sum_{j\in\mathcal{D}_2} p_j + \sum_{j\in\mathcal{D}_1} (p_j - \tilde{p}_j)]\\
&= 1 - [(\sum_{j\in\mathcal{D}_1} \tilde{p}_j) - \tilde{p}_i + \sum_{j\in\mathcal{D}_2} p_j + \sum_{j\in\mathcal{D}_1} (p_j - \tilde{p}_j)]\\
&= 1 - [\sum_{\mathcal{D}_1\cup\mathcal{D}_2} p_j - \tilde{p}_i] = \tilde{p}_i
\end{aligned}$$

Case 2: $i \in \mathcal{D}_2$

$$\begin{aligned} &\liminf_{k\to\infty} U_i(t_{k+1}) \\ &\geq 1 - [\sum_{j\in\mathcal{D}_1} \tilde{p}_j + \sum_{j\in\mathcal{D}_2\setminus\{i\}} p_j + \sum_{j\in\mathcal{D}_1} (p_j - \tilde{p}_j)] \\ &= 1 - [\sum_{j\in\mathcal{D}_1} p_j + \sum_{j\in\mathcal{D}_2\setminus\{i\}} p_j] = p_i \end{aligned} \tag{3.23}$$

According to to these equations,

$$\liminf_{k\to\infty} U_i(t_{k+1}) \geq \min\{p_i, \tilde{p}_i\} = \limsup_{k\to\infty} U_i(t_{k+1})$$

But, axiomatically, $\liminf_{k\to\infty} U_i(t_{k+1}) \leq \limsup_{k\to\infty} U_i(t_{k+1})$ and therefore we conclude that $\liminf_{k\to\infty} U_i(t_{k+1}) = \limsup_{k\to\infty} U_i(t_{k+1})$ and so the limit exists in any case. □

We proceed to study the feasibility space for all DWRR policies in terms of achievable domain service vectors, with the focus being on work-conserving modes of operation.

Work-Conserving Mode

In contrast to the non work-conserving mode, under work-conserving mode, the capacity region is a hyperplane, as stated by the following theorem. Note that we restrict our attention to the policies $\pi \in \Pi$ that reach steady state, since the convergence depends not only on the f-algorithm, but to the h-algorithm also. Thus convergence analysis is specific to every $\pi = (f,h)$ pair.

Theorem 2 (Feasibility Region in work-conserving mode)**.** *The feasibility space is a hyperplane defined by the intersection of hyperplanes defined by the following constraints:*

$$\begin{aligned} \sum_{i}^{|D|} \widetilde{U}_i(\pi) &\leq 1 \\ \widetilde{U}_i(\pi) &\leq \tilde{p}_i \end{aligned} \tag{3.24}$$

and

$$\begin{aligned} \widetilde{U}_i(\pi) = \tilde{p}_i, if\, \tilde{p}_i < p_i \\ \widetilde{U}_i(\pi) \geq p_i, if\, \tilde{p}_i \geq p_i \end{aligned} \tag{3.25}$$

Proof of Theorem 2. The constraints defined in eq. (3.24) are irrelevant to the goal vector, while the constraints in eq. (3.25) derive from the relationships between the goal vector $\boldsymbol{p} = (p_i)$ and $\tilde{\boldsymbol{p}} = (\tilde{p}_i)$.

The first two constraints in eq. (3.24) state that the utilization achieved in steady state cannot exceed the physical maximum utilization of the system (which equals one) and the third constraint simply states that a domain cannot exceed the maximum utilization (Definition 3.2).

In contrast to the previous case of non work-conserving mode of operation, where the system may experience idle time even when there are available requests for service, in the work-conserving mode of operation, service redistribution takes place and is up to policy h to achieve convergence. As we present, the feasibility space of the class is actually defined by the f-algorithm and the final performance depends on h-algorithm selection. Constraints in eq. (3.25) are proven in the two lemmas that follow.

Lemma 3 (Domains that cannot reach their goals)**.** *In the case where for some domain i, $\tilde{p}_i < p_i$ (the percentage goal p_i was ambitious), under any policy $\pi \in \Pi$, steady state exists and the achievable percentage $\widetilde{U}_i(\pi)$ equals $\tilde{p}_i$.*

The proof of this lemma is analogous to case 1, presented in the non work-conserving mode of operation and thus is omitted. The difference in the work-conserving case, is that there is no “null” domain and we simply don’t know how the extra service $C(t_n)$ is redistributed in domains that can utilize extra service.

Lemma 4 (Domains that are able to reach their goals)**.** *In the case where for some domain i, $\tilde{p}_i \geq p_i$ (the goal is feasible), under any policy $\pi \in \Pi$, $\widetilde{U}_i \geq p_i$ is true.*

Proof of lemma 4. Assume that there exists some domain $i \in \mathcal{D}_2$ (includes all the domains for which $\tilde{p}_i \geq p_i$) and consider some very large round n_0, for which $U_i(t_n) < p_i$ is true $\forall n > n_0$. This means that the policy can serve less than it expected, but also less than it could have received. According to the policy definition, this means that for any successive round $n > n_0$, the policy will set constantly $w_i^n = 1$ (independently of which other domains will also participate in all or to a subset of the new rounds). The reason is that there will be unserved requests of this domain that can participate in the following rounds. This makes $l_n = 0$ for all successive rounds. With the same reasoning, all the domains that will participate in the following rounds and on will be served constantly. Assume that we define as $\mathcal{B} \subseteq \mathcal{D}_2$ as the set in which all these domains belong.

But this means that all other domains that do not belong in this “special” set $\mathcal{B}$ cannot exceed their targets (and receive the “lost” service from domains in $\mathcal{B}$), since whenever they do so they get blocked by the policy. Then, summing up the utilizations of all the domains, we get that for any $t > t_{n_0}$:

$$\sum_{i \in \mathcal{B}(t)} U_i(t_n) < \sum_{i \in \mathcal{B}(t)} \tilde{p}_i < \sum_{i \in \mathcal{B}(t)} p_i$$
$$\sum_{i \notin \mathcal{B}(t)} U_i(t_n) \leq \sum_{i \notin \mathcal{B}(t)} p_i,$$

and thus:

$$\sum_{i=1}^{D} U_i(t_n) \quad = \quad \sum_{i\in\mathcal{B}} U_i(t_n) + \sum_{i\notin\mathcal{B}} U_i(t_n) < 1$$

But this means that there must be idling, which is a contradiction. Thus, it must hold that for any domain $i \in \mathcal{D}_2 : \widetilde{U}_i(\pi) \geq \tilde{p}_i$. $\square$

Note that in this case we cannot guarantee that the output utilization actually reaches steady state, or convergences to some point, since this depends on the selected h-algorithm. The above proof, nevertheless, guarantees that the utilization trajectory will move above p_i independently of the selected h-algorithm. $\square$

The following theorem is a direct consequence of lemmas 3 and 4 and states that all the policies that belong in class Π can be used to satisfy the objective in eq. (3.6).

Theorem 3. *All the policies that belong to class Π can be used to satisfy the objective in eq. (3.6).*

For example, as shown in Fig. 3.2, in both cases of non work-conserving and work-conserving mode, in the case of two domains, the goal lies over the line that joins points (0,1) and (1,0) and the feasibility region is a single point. The point lies over the line in the case that the available load can support the maximum utilization sum that equals one. In the case where a very high goal was set for both domains, then the performance point would be a point that lies below the line that is defined by $\sum_{i\in\mathcal{D}} \widetilde{U}_i(\pi) = 1$.

For any policy $\pi \in \Pi$ under non work-conserving mode a domain i receives exactly the $\min\{p_i, \tilde{p}_i\}$ as can be seen in Fig.3.2(a). The feasibility region, in the general case of stochastic arrivals, is defined by equations (3.24) and (3.25). For example the shaded region in Fig. 3.2(b) is defined by the inequalities of eq.(3.24). Under work-conserving operation a domain cannot receive less than the $\min\{p_i, \tilde{p}_i\}$ as we can see in examples presented in Figs. 3.2(c) and 3.2(d) where different relationship exists between $\tilde{p}_i$ and p_i. In all cases, the presented class Π of policies guarantees that the objective in eq.(3.6) is satisfied.

Details on the Load Redistribution Mechanism

In the work-conserving mode, for policies $\pi \in \Pi$, $\pi = (f, h)$ the normal operation is performed according to algorithm f and the service redistribution is performed according to algorithm h. While the operation of algorithm f guarantees the validity of the constraints in eq. (3.25) and the definition of the feasibility space, the algorithm h is responsible for the convergence properties and the exact points p_i^* achievable by the policy in effect.

The total CPU cycles that can be redistributed, are offered by domains $i \in \mathcal{D}_1$ to domains $j \in \mathcal{D}_2$ and (as analyzed in section 3.5.1) equals $\sum_{i\in\mathcal{D}_1} (p_i - \tilde{p}_i)$. Since we consider only work-conserving policies, the service that is not utilized by some domains will be redistributed to other domains. According to the previous analysis, the actual performance is the following:

$$\widetilde{U_i} = \tilde{p}_i \text{ , if } i \in \mathcal{D}_1 \text{ (lemma 3)}$$
$$\widetilde{U_i} = p_i + C_i \text{ , if } i \in \mathcal{D}_2 \text{ (lemma 4)}$$

Note that that as the feasibility space was clearly defined, the total extra service that will be redistributed depends only on vectors $\mathbf{p} = (p_i)$ and $\tilde{\boldsymbol{p}} = (\tilde{p}_i)$ and is irrelevant of the actual weight (value $k(t_n)$) selected by algorithm *f* and algorithm *h*. It is the extra service C_i received by each domain, that depends on algorithm *h*. The independence between *f* and *h* algorithms can be expressed by the following lemma.

Lemma 5 (Service redistribution). *Under any policy in class Π and $\forall i \notin \tilde{\mathcal{B}}$, $C_i(t_n)$ redistribution depends only on the scheduling algorithm h and is irrelevant to algorithm's f $k(t_n)$ selection.*

Proof of lemma 5. In order to prove the above lemma, we will show that the service redistribution $C_i(t_n)$ does not depend on the number $k(t_n)$ selected at each control instant at the end of round t_n by algorithm *f*. Assume a system with two domains and two policies $\pi_1(f_1, h_1), \pi_2(f_2, h_2)$ such that $h_1 = h_2$. Both h_1, h_2 *choose the other domain if it has requests*, while $k_2(t_n) = k_1(t_n) + k$, where k is any integer that can be supported by the queueing dynamics. The same reasoning can be followed for D domains. Also assume that $\tilde{p}_1 < p_1$ and $\tilde{v}_2 > p_2$ so that service redistribution will occur, since domain 1 has not sufficient load to support the goal percentage. However, in both policies, lemma 1 guarantees that $\widetilde{U_1}(\pi_1) = \widetilde{U_1}(\pi_2) = \tilde{p}_1$, thus in both cases $C_2^{\pi_1} = C_2^{\pi_2} = p_1 - \tilde{p}_1$ and so $\widetilde{U_2}(\pi_1) = \widetilde{U_2}(\pi_2) = p_2 + \min\{C_2, \tilde{v}_2 - p_2\}$. The fact that, under policy π_2, at every round, domain 1 was given more CPU cycles, plays no role in the long run. The minimum in this equality simply states that domain 2 cannot receive extra service that cannot be supported by its load. □

In Fig. 3.3, a visual representation of the feasibility space is given, in the case where three domains are competing for service using *f* rules. In this example, domain 3 can utilize all the capacity, $\tilde{p}_3 = 1$, for domain 1, $\tilde{p}_1 > p_1$, while for domain 2, $\tilde{v}_2 < p_2$. According to the class of policies definition, theorem 2 and lemmas 3 and 4, load redistribution will take place while the feasibility space of the class of policies is defined as the intersection of the following hyperplanes: hyperplane 1 defined by points $(1a, 1b, 1c, 1d)$ that presents the utilization area domain 1 can exploit, hyperplane 2 defined by the line defined by restriction $\tilde{v}_2$ and hyperplane 3 (triangle) defined by points $(3a, 3b, 3c)$ that presents the utilization area domain 3 can exploit. This intersection is the line that joins points A and B in the figure.

Depending on the service redistribution algorithm, the long run performance and steady state (if it can be achieved), clearly depends on the selected *h*-algorithm. For example, if domain 1 utilizes all the extra service, the long run utilization point is point B (where domain 3 receives no extra service), while the long run utilization point is point A, where domain 3 receives all extra service. Intermediate points can be obtained, if the appropriate redistribution algorithm is selected. Nevertheless, as this study assumes the absence of statistical knowledge regarding the arrival and service process, it is not

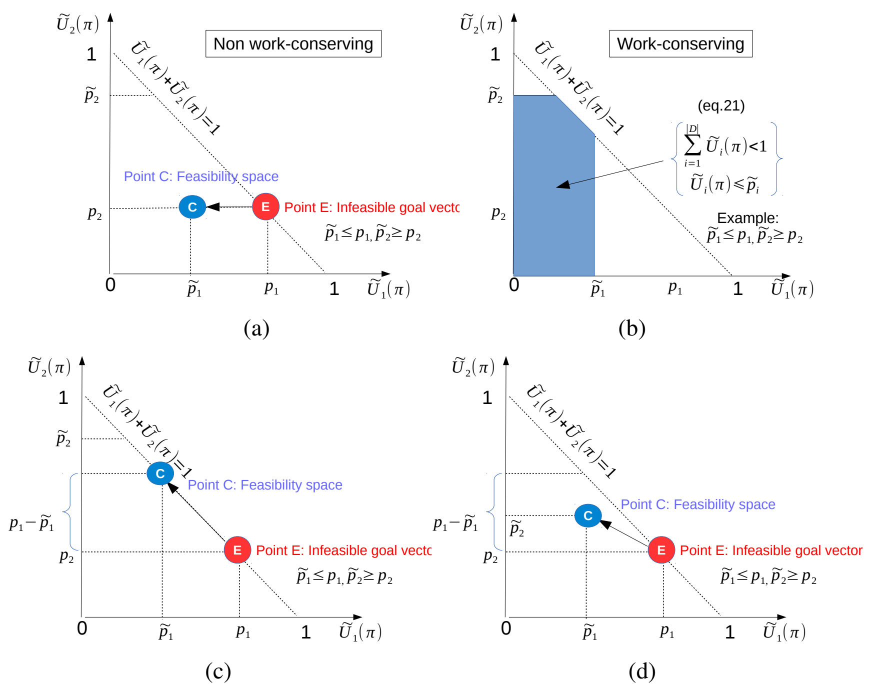


Figure 3.2: For any policy $\pi \in \Pi$ under non work-conserving mode a domain i receives exactly the $\min\{p_i, \tilde{p}_i\}$ (3.2(a)). The feasibility region, in the general case of stochastic arrivals, is defined by equations (3.24) and (3.25). For example the shaded region in 3.2(b) is defined by the inequalities of eq.(3.24). Under work-conserving operation a domain cannot receive less than the $\min\{p_i, \tilde{p}_i\}$ (examples 3.2(c) and 3.2(d)).

easy to find such a policy. Note that a second DWRR can be also deployed in order to precisely allocate this extra service as follows. After a long time has elapsed and $\mathcal{D}_1$ is not empty, set $\sum_{i \in \mathcal{D}_1} p_i - \tilde{p}_i > 0$. Then we can create a second DWRR list of weights with only $\mathcal{D}_2$ domains, define new goals regarding the extra service and redistribute service according to the policy's negative drift rules.

3.5.2 Notes on Ideally Overloaded Conditions

A special case of the above general analysis, is the case where the system is in an overload situation and at each time there are always available requests to schedule for all the domains.

Refinement of the T-SLA: In the overload case, the *T-SLA* now becomes "allocate an exact percent p_i of the CPU capacity in the server tier to service domain i, using an appropriate domain scheduling policy π at the controller". More, formally the *T-SLA* now becomes:

$$\widetilde{U}_i(\pi) \triangleq \lim_{t \to \infty} U_i^{\pi}(t) = p_i \tag{3.26}$$

For all the non-idling policies operating in rounds, it holds that at least one domain i with $w_i^n > 0$ participates in each round. In overload conditions, by following the analysis in the proof of Theorem 1 and letting $l_n = 0$, we can verify that the limit of $U_i(t)$ as $t \to \infty$ of every bounded-round policy exists almost surely. Also, for the case of work-conserving policies with overloaded queues, it is easy

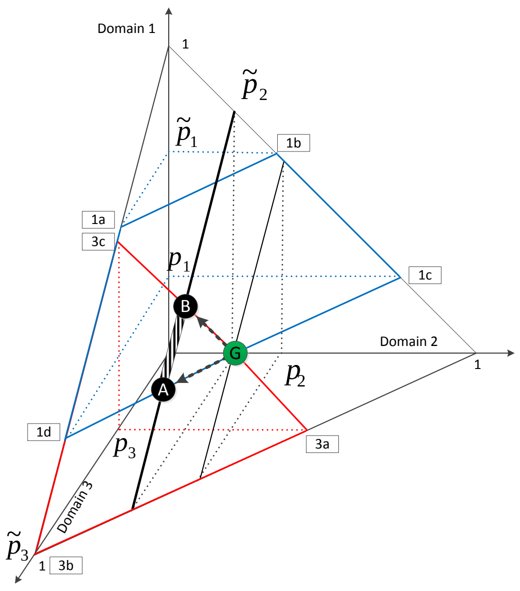


Figure 3.3: A three domains scenario, where $\tilde{v}_2 < p_2$ and redistribution must take place. The final performance depends on algorithm h rules.

to see that a fair policy is a non-idling policy. For all the bounded round-fair policies the following is true:

Theorem 4. *The utilization of every domain, under a bounded round-fair policy is a fundamental (or Cauchy) sequence and satisfies the T-SLA in equation 3.26 with probability one.*

The case of saturated conditions, is a special case of the non work-conserving case and analysis presented for the proof of Theorem 1. In the case of saturated conditions, only the "suffering domains" are served at each round, while $\tilde{p}_i > p_i, \forall i \in \mathcal{D}$. In this case, for every domain $\widetilde{U}_i = p_i$ w.p. 1. Two example policies that belong to the class of overloaded bounded-round fair policies, are described below. We study their properties extensively, in subsequent sections.

Overloaded Only Below Goal Participates (*OOBG*)

Under the Overloaded Only Below Goal Participates (*OOBG*) policy, in round n (that starts at time t_n), set $w_i^n = 1$ for all the domains where $U_i^{SOBG}(t_n) \leq p_i$; for the rest set zero weights. In other words, at every round only the domains that have allotted CPU time less than or equal to their *T-SLA* target are given one slot; the rest are given none.

Overloaded Only The Most Suffering (*OOMS*)

Under the Overloaded Only The Most Suffering (*OOMS*) policy, each round is composed of only one domain which is served once. The selected domain is the one with the largest amount of missing service. More precisely, at a decision instant t_n when the n round begins, we set $w_i^n = 0, i \neq j$ and $w_j(t_n) = 1$ where $j = \arg\min_{k \in \mathcal{D}}\{U_k^{OOMS}(t_n) - p_k\}$. *OOMS* resembles the family of *maximum weight* policies [56],[108]; that are well-known optimal policies and can be thought of as a degenerate case of the Round Robin scheduling policies.

3.5.3 Summary of Results Regarding DWRRR Analysis

The following theoretical results can be collectively deduced regarding the analysis of negative drift DWRR policies, under unknown service time and arrival process statistics:

1. In the case of saturated arrivals, $\forall \pi \in \Pi$ converges with probability 1 (w.p. 1) to the goal percentage, p_i.
2. In the case of stochastic arrivals, when operating in non work-conserving mode, steady state exists for any policy $\pi \in \Pi$. Any policy converges with probability 1 (w.p 1) to the minimum between the goal percentage and the maximum utilization service.
3. Assuming steady state, the feasibility space for any policy $\pi \in \Pi$, in the case of stochastic arrivals and all modes of operation, can be clearly defined.
4. In the case of stochastic arrivals, under work-conserving mode of operation:
   a. any policy $\pi \in \Pi : \tilde{p}_i \leq p_i$ convergences to $\tilde{p}_i$.
   b. for any policy $\pi \in \Pi : \tilde{p}_i > p_i$ convergence point depends on the service redistribution algorithm h.

Regarding point (1) an overloaded server system, ideally saturated, is clearly related with the way we define busy periods. In queueing theory, a busy period is the time between the arrival of a customer to an idle server and the first time the server is idle again. In the ideally saturated case we consider, we assume an infinite busy period where we extend this concept and furthermore we assume at any control instant t_n, $X_i(t_n) > 0, \forall i \in \mathcal{D}$ (there is an available request by every domain waiting for service), without however taking into account any load variations, burst sizes or arrival process and service time distribution details. In order to prove (2) we introduced a novel approach for proving convergence for the non work-conserving mode of operation, via the concept of null-domains. For point (3) we examined the feasibility space of the DWRR policies using a series of lemmas. We proved (4a) and we provided a detailed analysis regarding point (4b).

3.6 Implementation Considerations

In practical considerations, we are interested not only for the convergence of the sequence, but also how fast the sequence convergences to its target. In principle, similar to stochastic approximation for dynamic systems, a trade off between rate of convergence and convergence to the goal exists [67]. Under saturated conditions operation, the following analysis holds regarding the rate of convergence, the time and round of convergence estimation.

3.6.1 Rate of convergence analysis

The following theorem states, that under saturated arrivals conditions, DWRR converge to the objective sub-linearly.

Theorem 5 (rate of convergence)**.** *In ideally saturated conditions, all the policies in class Π experience sub-linear convergence to the target utilizations.*

Note that there exist standard techniques in the literature that can accelerate convergence [109].

Proof of Theorem 5. The rate of convergence of $U_i(t_n)$ sequence can be found by $\lim\limits_{n\to\infty} R_i(t_n)$, where $R_i(t_n)$ is defined according to eq. 3.27(this limit exists by Theorem 1).

$$R_i(t_n) = \frac{|U_i(t_{n+1}) - p_i|}{|U_i(t_n) - p_i|} \tag{3.27}$$

According to eqs. (3.9) and (3.27), we have that

$$R_i(t_n) = \frac{|\frac{t_n}{t_{n+1}} U_i(t_n) + \frac{S_i^n}{t_{n+1}} - p_i|}{|\frac{t_{n-1}}{t_n} U_i(t_{n-1}) + \frac{S_i^{n-1}}{t_n} - p_i|} \tag{3.28}$$

The numerator of the right part of the equation is equal to

$$\left|\frac{t_n}{t_{n+1}} U_i(t_n) + \frac{S_i^n}{t_{n+1}} - p_i\right| = \left|\frac{t_{n-1}}{t_{n+1}} U_i(t_{n-1}) + \frac{S_i^{n-1}}{t_{n+1}} + \frac{S_i^n}{t_{n+1}} - p_i\right| \tag{3.29}$$

After we replace eq. (3.29) in eq. (3.28) and perform some arithmetic operations, we have that:

$$R_i(t_n) = \frac{t_n}{t_{n+1}} \cdot \left|1 + \frac{S_i^n - L_n \cdot p_i}{t_{n-1} U_i(t_{n-1}) + S_i^{n-1} - t_n \cdot p_i}\right| \tag{3.30}$$

where we used the fact that $t_{n+1} = t_n + L_n$. Let

$$\begin{aligned} g(t_n) = \frac{N_i(n)}{D_i(n)} &= \frac{S_i^n - L_n \cdot p_i}{t_{n-1} U_i(t_{n-1}) + S_i^{n-1} - t_n \cdot p_i} \\ &= \frac{S_i^n - L_n \cdot p_i}{t_{n-1} U_i(t_{n-1}) + S_i^{n-1} - (t_{n-1} + L_{n-1}) \cdot p_i} \\ &= \frac{S_i^n - L_n \cdot p_i}{t_{n-1} [U_i(t_{n-1}) - p_i] + S_i^{n-1} - L_{n-1} \cdot p_i} \end{aligned} \tag{3.31}$$

where $N_i(n)$ denotes the numerator and $D_i(n)$ denotes the denominator. We want to show that $\lim\limits_{n\to\infty} g(t_n) = 0$. A minor technical difficulty arises from the presence of the term $t_{n-1}[U_i(t_{n-1}) - p_i]$ in the denominator, which leads to an $\infty \cdot 0$ term in the limit.

To overcome this technicality, we define next two random sequences $v_n(t_n)$ and $h_n(t_n)$ as follows:

$N_i(n)$	$D_i(n)$	u_i^n	h_i^n
≥ 0	> 0	$u_i^n > 0$	$-\frac{D_i(n)}{t_{n-1}} < h_i^n < 0$
≥ 0	< 0	$0 < u_i^n < \frac{\lvert D_i(n)\rvert}{t_{n-1}}$	$h_i^n < 0$
< 0	> 0	$-\frac{D_i(n)}{t_{n-1}} < u_i^n < 0$	$h_i^n > 0$
< 0	< 0	$u_i^n < 0$	$0 < h_i^n < \frac{\lvert D_i(n)\rvert}{t_{n-1}}$

$$v_n(t_n) = \frac{S_i^n - L_n \cdot p_i}{t_{n-1}[U_i(t_{n-1}) - p_i + u_i^n] + S_i^{n-1} - L_{n-1} \cdot p_i}$$

$$h_n(t_n) = \frac{S_i^n - L_n \cdot p_i}{t_{n-1}[U_i(t_{n-1}) - p_i + h_i^n] + S_i^{n-1} - L_{n-1} \cdot p_i}$$

where v_i^n,h_i^n are random sequences chosen to make the bracketed terms nonzero and in addition obey the following rules:

These rules guarantee that we can bound $g(t_n)$ as follows:

$$v_n(t_n) < g(t_n) < h_n(t_n), \quad \forall n.$$

We can easily see, however, that $\lim_{n\to\infty} v_n(t_n) = 0$ and $\lim_{n\to\infty} h_n(t_n) = 0$. (Both limits go to zero since by definition $\forall n$, $S_i^n < K \cdot S_i^{max} < \infty$, $L_n = \sum_{i:w_i^n \geq 1} S_i^n \leq \sum_{i=1}^{|D|} K \cdot S_i^{max} < \infty$). By the squeezing principle it follows that $\lim_{n\to\infty} g(t_n) = 0$ and thus we can calculate the limit in eq. (3.30):

$$\lim_{n\to\infty} R_i(t_n) = 1 \cdot |1 + 0| = 1 \tag{3.32}$$

The utilization approaches the target sub-linearly although variable speed may be observed during every sample path.

□

3.6.2 Time and Round of Convergence Estimation

We have seen that convergence to the desired targets can be slow (theoretically, sub-linear according to Theorem 5). Quite often, sub-linearly convergent sequences approach their limit fairly fast and then take a long time to achieve the theoretical value. From a practical standpoint, it may be satisfactory to know that "with high probability, we have come close to the target". In this section, we investigate heuristically the time and the number of rounds that is required to enter a "sphere of convergence around the target". We will try to find n_0 such that, with high probability,

$$g_i(t_n) = |U_i(t_{n+1}) - U_i(t_n)| \leq \varepsilon, \forall n > n_0 \tag{3.33}$$

We begin by writing the following expression, based on eq. (3.12) and following the analysis in the proof of convergence:

$$g_i(t_n) \quad = \quad |U_i(t_{n+1}) - U_i(t_n)| \leq \frac{\sum_j^D K \cdot S_j^{max} + K \cdot S_i^{max}}{t_n} \tag{3.34}$$

where we remind that $K \cdot S_i^{max}$ notes the maximum observable service time for all the requests of domain i and in all sample paths. If now we want $|U_i(t_{n+1}) - U_i(t_n)| < \varepsilon \; \forall i \in \mathcal{D}$, then from eq. (3.34), we can write

$$\begin{aligned} \frac{(D+1) \cdot \max_{i \in \mathcal{D}}\{K \cdot S_i^{max}\}}{t_n} &\leq \varepsilon \xrightarrow{\exists n_0} \\ t_{n_0} &\geq \frac{(D+1) \cdot \max_{i \in \mathcal{D}}\{K \cdot S_i^{max}\}}{\varepsilon} \end{aligned} \tag{3.35}$$

The time t_{n_0} is an estimation for the convergence time of all the domains to converge in a sphere of deviation ε.

Round of convergence n_0 estimation

Furthermore, an estimation on the round of convergence can be also made. If we know that the time of convergence is lower bounded when L_n is maximized, then the round of convergence is upper bounded when every round has the minimum duration (largest possible time of convergence with the maximum number of rounds). This happens when only one domain participates in the round and this happens with the minimum service time. According to this, in a worst case scenario, the round of convergence n_0 in a sphere of deviation ε is equal to

$$n_0 \leq \frac{t_{n_0}}{\min\limits_{i \in \mathcal{D}, \text{over all } n} \{S_i^n\}} \tag{3.36}$$

3.7 Evaluation

In this section we present the evaluation of the DWRR policies using an extensive set of simulations. The goal of this section is to verify the theoretical results obtained in previous sections, while also give the administrators interested to apply DWRR a better representation of the actual DWRR performance. A custom, JAVA-based simulator was build to perform the following experiments.

A large number of system (e.g., number of servers) and statistical (e.g., service time distribution) parameters are tunable and affect the performance of DWRR. We begin by focusing on the stochastic arrivals case. A single domain is selected at random when redistribution should be performed (we refer to this policy as *OBG* (serve Only Below Goal)). We present an indicative set of simulations regarding the following:

1. convergence of the policies in non work-conserving mode of operation and comparison with other schemes.

2. verification of the feasibility space analysis, by providing points outside the feasibility space and presenting the points where the system converges.

3. understanding of the *h*-policy effects on final performance, using the goals that fall outside the feasibility space, under different *h* algorithms.

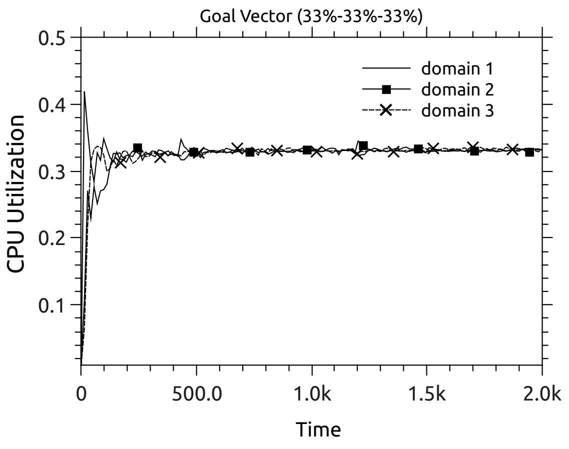
3.7.1 Convergence of the Policies in Work-Conserving Mode of Operation and Comparison with Other Schemes

In both saturated systems and systems with stochastic arrivals, DWRR can be used to satisfy the differentiation objectives. The only requirement to reach a predefined goal is that the goal is feasible. Figs. 3.4(a) and 3.4(b) are used to verify the DWRR performance in the following simulation scenarios, where the *OBG* is selected as the member policy from the DWRR class defined (in *OBG* when domain i is selected then $w_i = 1$).

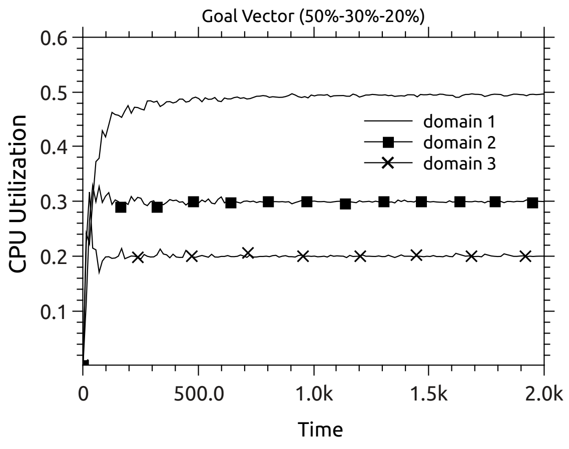
In Fig.3.4(a) the goal vector defines equal CPU shares for three domains $\{33\%, 33\%, 33\%\}$ where in Fig. 3.4(b) the goal vector was set equal to $\{50\%, 30\%, 20\%\}$. In both scenarios, poisson arrival rates where used for all domains, with mean rate vector equal to $\{0.5, 0.4, 0.3\}$ and exponential service time distribution with means $\{3.0, 2.0, 1.0\}$. Note that the reason for this selection was to use different workload characteristics per domain. As we can see in both cases DWRR are able to differentiate according to the requested goal vector since the policies adapt in real time in workload variations and the stochastics of the system.

In Fig. 3.4(c) a comparison is presented between DWRR (OBG), WRR [99], Random scheduling and Credit Scheduling [42], with the workload and the goal was set,to $\{50\%, 30\%, 20\%\}$. In this figure, the total absolute deviation is presented $\sum_{i=1}^{D} |U_i(t_n) - p_i|$. WRR is a static policy where the scheduling weights are defined according to the goal vector, so at each round, 5 requests where served by domain 1, 3 from domain 2 and 2 from domain 3 (if there were available); in Random scheduling a domain is selected at random; where we use a variation of the Credit scheduling policy (because in the XEN hypervisor it adapts weights every 30ms) to compare it with OBG. As we can see, WRR scheduling completely fails to align with the objective, since the service time per domain is different. In fact, in this example random scheduling performs even better. Adaptive policies like *OBG* and Credit can be used to satisfy the objective. DWRR may be more accurate when bursty traffic phenomena and ON/OFF periods are experienced. In smoother workload variations, OBG and Credit scheduler exhibit similar performance. Nevertheless, this increased accuracy comes at the expense of increased overhead, while when the application environment requires for I/O handling and preemptive operations.

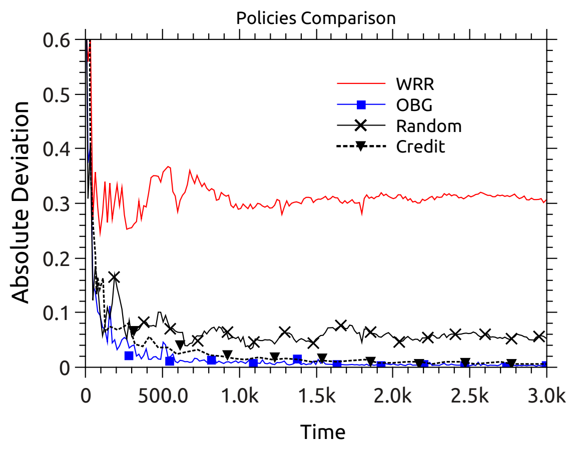
In Fig. 3.4(d) the effects of increasing the domains in the system are presented. We present the case where we want to load balance the traffic between 2, 4 and 8 domains respectively and the goal



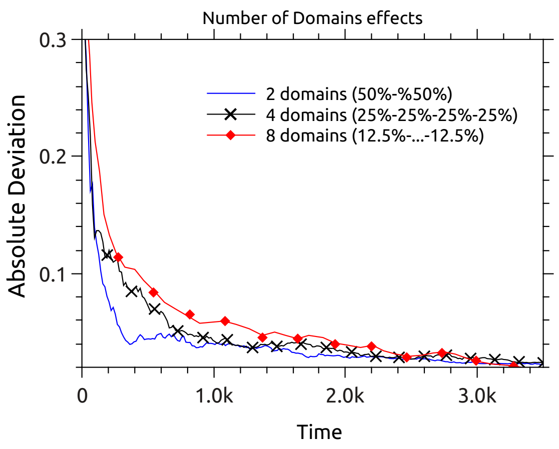
(a)



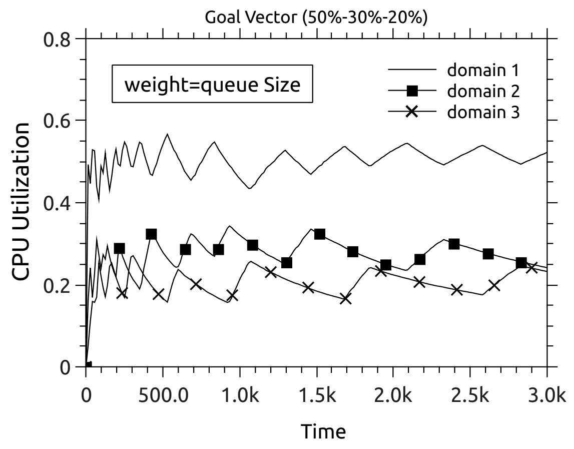
(b)



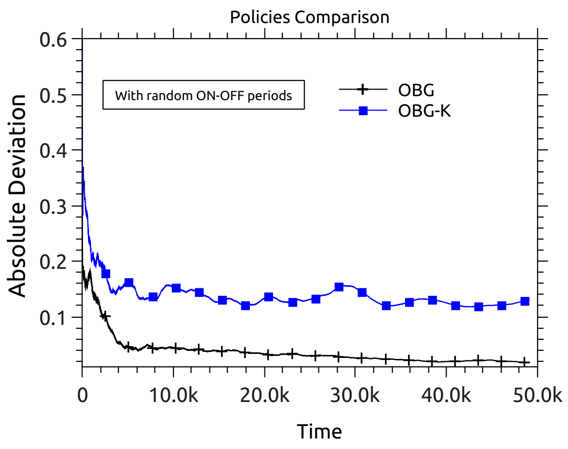
(c)



(d)



(e)



(f)

Figure 3.4: DWRR Performance

is feasible. As we can see, increasing the number of domains results in increased convergence time; increasing the number of domains, results in less scheduling opportunities per round per domain.

According to the class of policies definition, algorithm *f* sets no restriction on the number of requests actually served per round when a "suffering" domain is selected. In Fig. 3.4(e) we can see

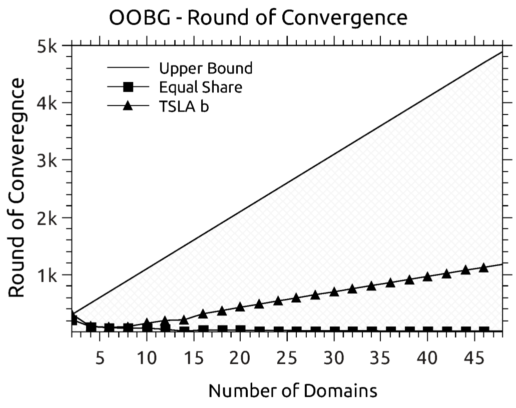
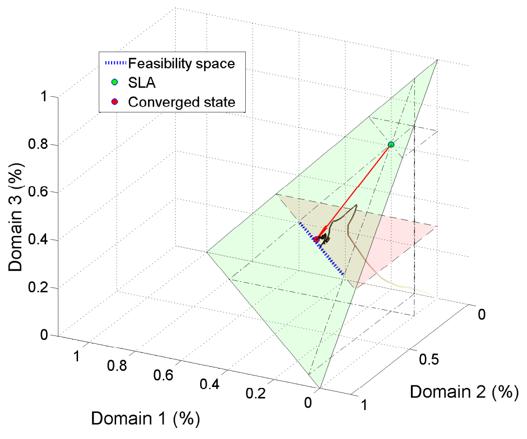


Figure 3.5

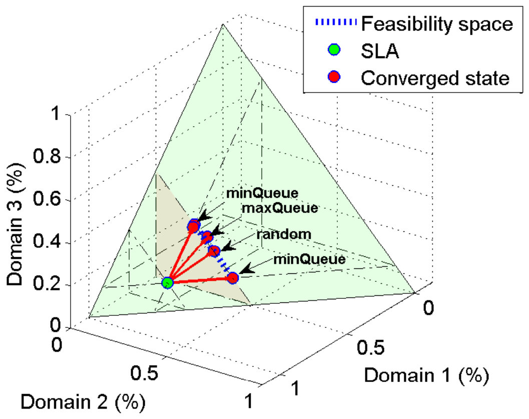
the effect of setting the weight of a suffering domain equal to the queue size of this domain. (E.g., if a domain is below its goal and has 5 requests in queue at the control instant t_n, then set its scheduling weight equal to 5 for round n). The setup of the experiment is the same as in Fig. 3.4(b). As we observe, this greatly affects the convergence period. The reason is that by giving increased gain at each control instant, we violently change the trajectory giving a different tendency to the system. Since the gain cannot be controlled by classical methods (proportional control, integrators etc. [44]), increasing the gain indirectly by large weights, leads to increased convergence times. In Fig. 3.4(f) we compare the *OBG* policy with the case where the maximum weight can be allocated according to policy f definition (*OBG*-K). Furthermore to stress the algorithms we create random ON-OFF [110] periods that essentially emulate bursts of requests. Evidently, the convergence properties of delayed decision making results in slow convergence.

In the following we investigate empirically the accuracy of the results in section 3.6.2. In Fig. 3.5 we demonstrate the number of rounds that are needed to satisfy two different SLAs, and also contrast it to the lower bound that eq. 3.36 provides. We present the average number of rounds n_0 required until $|U_i(t_{n_0}) - p_i| < 0.01$ holds for every domain i. The averages are calculated over 1,000 samples per configuration and plotted versus the number of domains. In all the scenarios, we use one CPU and service times with mean $E[S_i] = 1$ for all the domains. The first *T-SLA* requires equal share between all the domains, while in the second *T-SLA* one domain receives 20% and the rest share the rest 80% of CPU power. We plot the lower bound of eq. 3.36 for $S_i^{max} = S_i^{Min}$. In this case the service times are constant. From an extended set of simulations conducted and as we can see in this figure also, different SLAs require different number of rounds to converge but the upper bound that eq. 3.36 is still satisfied.

Remark: From a practical perspective, a major concern when applying DWRR policies, is the increased overhead requirements. The reason is that the policy requires to track at the the end of every round, the CPU usage per domain and update statistics accordingly, in order to take the control action for the next round. Selecting different weights is a way to track only the context switching instances (when we start to serve the other domains) and thus reduce overhead. Nevertheless, this comes at increased convergence periods.



(a) Convergence trajectory in the case of three domains. Domain 3 has insufficient load to meet its SLA, resulting into a different converged state. The trail starting from the origin $\emptyset$ and ending at the final state show domain utilization measurements as time advances.



(b) Feasibility space and converged states in a scenario similar to Fig. 3.6(a). The system always converges to the analytically projected spaced (dashed line). The exact point of convergence is defined by the employed h-algorithm.

Figure 3.6: DWRR Performance under various service redistribution policies.

3.7.2 Feasibility Spaces

As already analyzed theoretically, whenever a goal is feasible, DWRR class of policies guarantees that in the long run the objective is satisfied. Under some statistical assumptions regarding the arrival process or when in non work-conserving operation, it will actually receive exactly its goal percentage. Nevertheless, if for some domain i the goal is outside the feasibility region, then under DWRR the maximum achievable percentage $\tilde{p}_i$ is achieved and the difference $p_i - \tilde{p}_i$ will be utilized by some other domain according to the rules set by policy h. We can see this behavior in Fig. 3.6(a) where the goal is on purpose set outside the feasibility region. In this experiment a very high goal was set (the top bullet point in the figure) that defines the vector $(10\% - 20\% - 70\%)$. The arrival and service process where such that could not satisfy this objective. As expected theoretically, by theorem 2, the feasibility space is the dashed line defined by the intersection of hyperplanes set by equations (3.24) and (3.25). Note that the actual percentage achieved depends on the redistribution algorithm h (in this case was *OBG* algorithm that serves 1 request per domain when below goal, one domain at random when there are no available requests in the system from domains below the goal). DWRR scheduling guarantees that even when workload variations exists, ON/OFF periods or random disturbances, the feasibility space will be always given by equations 3.24 and 3.25 and so a minimum service is guaranteed per class.

3.7.3 Load Redistribution Mechanism

In Fig. 3.6(b) we present the effect of using different redistribution policies, in the case of infeasible vector goal. The goal vector was set equal to $(70\% - 30\% - 20\%)$ and is denoted with the leftmost dot

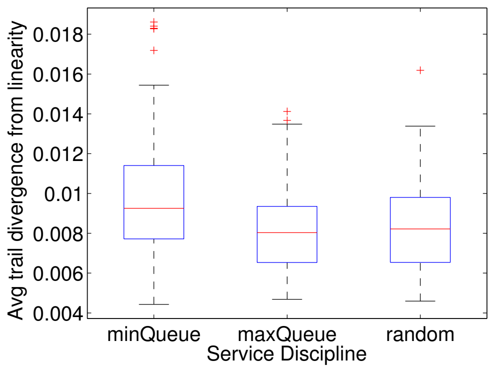


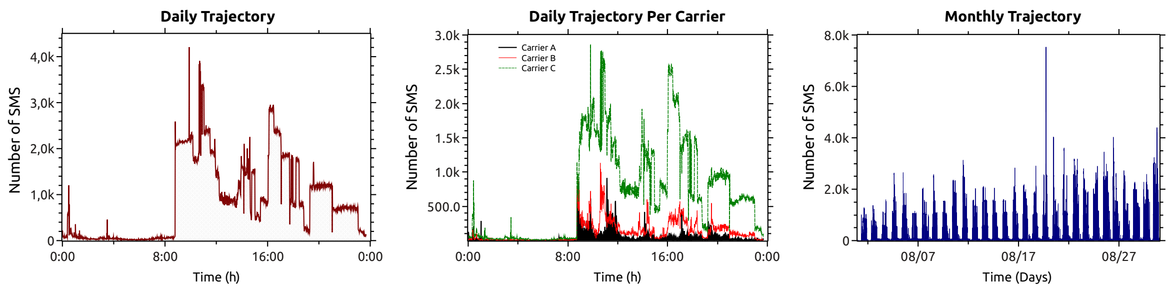
Figure 3.7: The selected h-algorithm affects the way the system converges to the final state.

in the figure that nevertheless is outside the feasibility region. As in the previous case, the goal was set on purpose outside the feasibility region in order to perform redistribution of service. We compare DWRR in the case of four redistribution algorithms. Whenever an over-satisfied domain is served in order to be work-conserving: a) choose one at random; b) choose the one with the minimum queue size; c) choose the one with the maximum queue size. As we can see, the feasibility space is the one defined according to theorem 2 and is irrelevant of the redistribution algorithm. The application of h is responsible for the actual percentage obtained.

An interesting observation can be made regarding the load redistribution mechanism. As hinted by Fig. 3.6(b) the “minQueue” redistribution algorithm exhibits a wider variation in terms of converged state placement over the feasibility space. To study this issue further, we study the form of the trail produced by “minQueue”, “maxQueue” and “random” over 100 simulation runs with the same configuration as Fig. 3.6(b). For every run/ h-algorithm, we log the average distances of the trail points from the line defined by the points $\emptyset$ and converged state. The results over the 100 runs form the boxplot of Fig. 3.7. In essence, the employed distance metric expresses the “wandering” rate of the system’s trail towards convergence. Ideally, a perfect f/h algorithm combination would yield a linear trail from the point $\emptyset$ to the final converged state. The “maxQueue” and “Random” h-algorithms yield approximately the same degree of divergence from linearity. However, the “minQueue” approach exhibits a considerably higher of divergence. As shown in Fig. 3.6(b), the “minQueue” approach is more exploratory in terms of convergence point within the feasibility space. By taking more subtle scheduling decisions (choosing the smaller queues), “minQueue” is able to achieve a wider set of final converged states. On the other hand, “maxQueue” and “random” take more crude scheduling decisions, resulting into smaller divergence in terms of converged states within the feasibility space. In other words, the h-algorithms also regulate the feasibility space “exploration” rate of the system.

3.7.4 Evaluation using real traces

The available traces at our disposal comes from a mobile computing services company, where three customer classes are competing for a single service deployed in a server machine. Our goal is to provide guarantee service by means of CPU power to three customer classes.



(a) (b) (c)

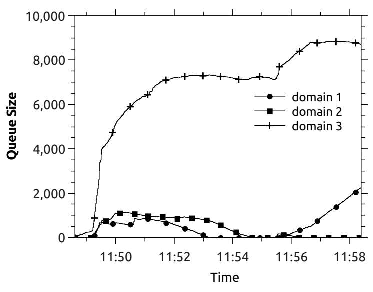
Figure 3.8: Incoming traffic from three carriers are depicted in a real mobile promotion campaign.

We use a custom service mediation application that can be used to emulate actual server operations. The goal of the following analysis is to demonstrate the performance of *OBG* algorithm in realistic conditions emulating an operational environment. Three service domains (clients) generate http/soap traffic (XML-RPC requests over http) and send it to an http controller (Java Servlet). The http controller is used to a) accept this traffic in a queue per domain, b) perform scheduling decisions and send traffic to the service mediation application in the custom ESB. The service mediator is responsible to: process the requests, forward them to an endpoint service and return to the client the response from the service. The trajectory of the total daily traffic is shown in Fig. 3.8(a) and the traffic pattern per domain is depicted in Fig. 3.8(b). In our effort to emulate the conditions of the actual application, we use a custom mediation service with an average service time of 20ms per request (which was the one also reported in real conditions). Furthermore, we consider the same type/size of request per domain.

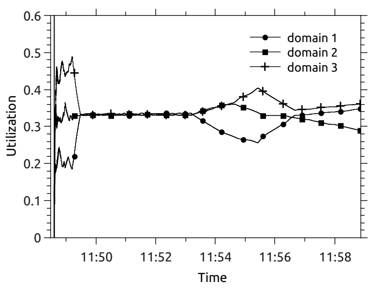
Implementation details: In practice, the scheduler could be implemented inside the mediation application, or outside by registering domains and assigning traffic per domain, where in this demonstration we follow the latter approach and we implement the scheduler inside the controller. This is similar to interposed request schedulers [111] where the resource control is applied externally and the server is treated as a black box. The http controller is a custom multi-threaded Java Servlet deployed in Glassfish 3.1.2 Application Server. The receiving process is separated from the scheduling process at the thread level. This way, while one thread puts requests in the queues the other is able to pull asynchronously. In principle, assigning different thread pools to different operations may lead to unpredictable behavior, since these operations rely on the way the OS performs thread scheduling [41]. The servers are hosted in guest VMs that reside in a single i5 - 3.2GHz, 8G memory Ubuntu server machine. The Servlet(scheduler), the mediation application and the endpoint service are hosted in a single guest VM to avoid communication overhead, while the clients are hosted in different VM machines (one per client), also hosted in the same physical machine.

Evaluation scenarios: We benchmark the *OBG* algorithm in *Equal share* and *Unequal share* scenarios:

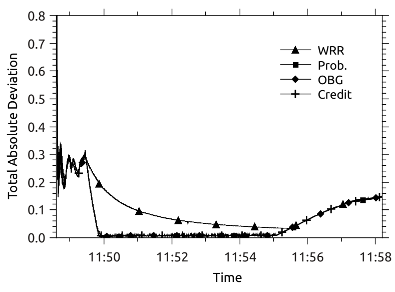
Equal share scenario: In this scenario, we want all the carriers (domains) to receive equal share from the CPU of the server hosting the mediation service. Also, according to the SLA defined, when the total traffic is less than the throughput capacity, every domain will use the maximum achievable



(a) Backlog evolution.

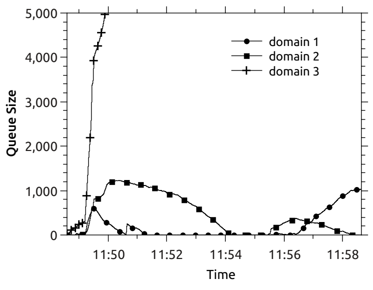


(b) Utilization achieved per domain.

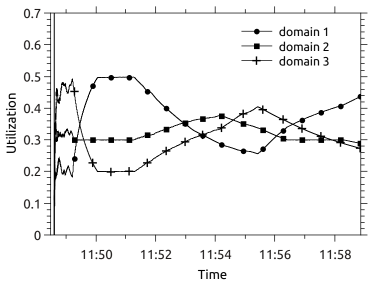


(c) Comparison between scheduling policies.

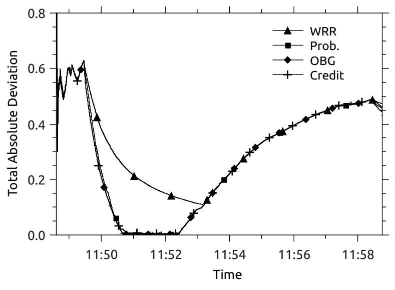
Figure 3.9: Study of stochastic arrivals using traces. The CPU utilization goal vector is set equal to 33%, 33%, 33%.



(a) Backlog evolution.



(b) Utilization achieved per domain.



(c) Comparison between scheduling policies.

Figure 3.10: Study of stochastic arrivals using traces. The CPU utilization goal vector is set equal to 50%, 30%, 20%.

CPU cycling share. The backlog evolution per domain for that interval is presented in Fig. 3.9(a) and in Fig. 3.9(b) the allotted utilization is presented. As we can conclude by these figures, for the period when all the domains have available requests in queues, domain 3 requests that tend to dominate the system, are held back. When the backlog is close to zero for domains 1 and 2 (at approximately 11:54), the utilization is proportional to the arrival rate of every domain and thus domain 3 starts to receive more utilization. In this scenario, *OBG* can be used to offer overload protection and load balancing functionality, while it converges fast (in less than 1 minute).

Unequal share T-SLA (50%-30%-20%) Scenario: In the second scenario, we want each domain to receive a predefined percentage $\{50\%, 30\%, 20\%\}$ from the server CPU cycles. As we can see in Fig.3.10(a), where the backlog evolution is presented and Fig. 3.10(b) where the allotted utilization is presented, the algorithm again blocks domain 3 and differentiates according to the *T-SLA*.

For both scenarios, a comparison is also presented between *OBG* , WRR, the probabilistic algorithm based on predictions, presented in [13] and a Credit Scheduling, using the workload traces. As we can see, in both cases where the goal was set equal to $\{33\%, 33\%, 33\%\}$ Fig.3.9(c) and $\{50\%, 30\%, 20\%\}$, Fig.3.10(c) respectively, the only algorithm that fails to meet the objective is the WRR. The reason is that is not able to adapt to the workload variations and satisfy the objectives. The remaining algorithms, although in a short time scale, are able to provide the required differentiation under heavy load. Note that in the comparison presented in Fig. 3.10, Credit scheduling and prediction techniques were able to differentiate traffic according to the objective, but not able to optimally control the allocation accuracy and the relative error, due to arbitrary ON/OFF periods for all the domains. In contrast, in the traces experiment a single domain dominates the system (domain 3), while the other two domains have low arrival rate. In this case of smooth workload changes, both Credit scheduling and prediction techniques experience almost identical performance with *OBG*. In conclusion, static WRR policies fail to meet the objective under dynamic workloads; in the case of complex workloads for multiple domains, Credit schedulers must be tuned for optimal performance, while prediction techniques and DWRR can handle the dynamics of the system better and increased accuracy, at the expense of increased overhead.

3.8 Chapter Conclusions

In this chapter we studied a class of negative drift Dynamic Weighted Round Robin policies, that is able to guarantee specific CPU shares in server systems, between competing domains. We provided a generic mathematical framework, where the class of policies requires no statistical knowledge regarding the arrival and service process and can be used in multiple application scenarios. The following conclusions hold for the proposed class of negative drift DWRR policies: In the case of saturated arrivals, any DWRR policy converges with probability 1 to the goal percentage. In the case of stochastic arrivals, when operating in non work-conserving mode, any policy converges with probability 1 to the minimum between the goal percentage and the maximum utilization service. The feasibility space for any DWRR policy, in the case of stochastic arrivals and all modes of operation, can be clearly defined. Finally, in the case of stochastic arrivals, under work-conserving mode of operation the differentiation objective is still satisfied, nevertheless the final performance depends on the redistribution mechanism.

Our future plans include comparison between DWRR and commercial schedulers in hypervisor systems; implementation of DWRR in web server operations, I/O handling enhancements and performance investigation under preemptive operation. Queueing analysis of the backlog evolution is also planned for future work. The main axis on which our efforts are placed, is towards reducing the increased policy overhead. Our future plans also include the implementation of the DWRR policies in various points of the end-to-end architecture described in Chapter 2, where control must be applied in order to differentiate services between competing customer classes.

Chapter 4

Stochastic Enqueueing and Predictions Techniques for Guaranteed Service Delivery

The objective we consider in this chapter, is related with service differentiation guarantees, like in the case of the previous chapter. Nevertheless, the approach we consider focuses on stochastic enqueueing techniques and prediction algorithms, that are able to provide differentiating services between competing customer classes, rather than scheduling. In the stochastic enqueueuing approach, we present a scheme where guaranteed service delivery can be provided to each customer class, by selecting the queue to store incoming requests. This scheme is evaluated in two application scenarios: a) In server systems, where guarantees are provided on CPU usage, and b) in 802.11 Access Points, where the technique is used to provide guarantee throughput to individual customer classes. In the predictions based approach, in contrast to existing work where estimation and prediction techniques are used to estimate average values, we adjust scheduling probabilities based on system dynamics. For both the enqueueing and prediction techniques no steady state analysis is provided, nevertheless through extensive simulations we present that are acceptable approximations to the differentiation objectives.

4.1 Bucket Queue System (BQS): a Stochastic Enqueueing Framework

4.1.1 Introduction

In modern cloud environments,like the one presented in Chapter 2, a virtualized infrastructure hosts several applications/services and the traffic entering the provider network (in the datacenter or the edge network), is naturally classified into classes - called *Service Domains* (SD). How processing time is allocated to these domains, clearly affects the performance seen by the application requests. As we also noted in the previous chapter, in typical SLAs, various metrics regarding this performance (eg. delay, cpu utilization, throughput etc.) are included and expressed in various ways. In this

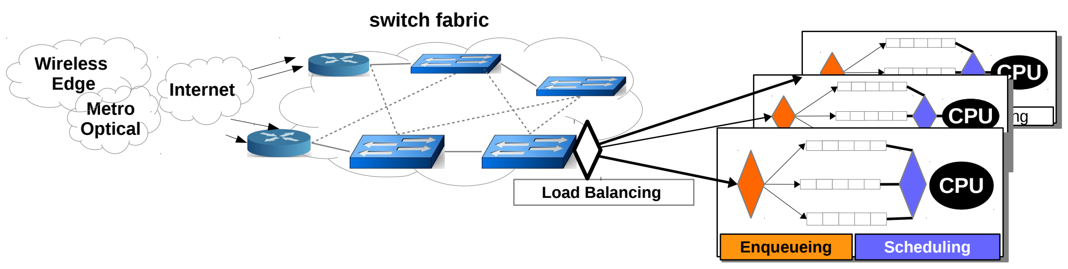


Figure 4.1: End-to-End Multitier Architecture, with multiple control points.

chapter, we focus on CPU utilization and throughput.

For simplicity and clarity of presentation, we first consider CPU utilization of the servers in the processing tier only. As an example, an SLA with CPU utilization metrics can be described as follows: guarantee 20% of total (preprocessing tier) CPU power to domain A and 80% to domain B. in the multi-domain, end-to-end network architecture that we investigate, a domain could be for example a large Virtual Network Provider. In Fig. 4.1, a typical multi-tier architecture is presented. Core router/switches accept external traffic through the Internet or a VPN and they route this traffic through the fabric to one of several servers. This is the place where the "main business logic" is executed.

Three elements have an effect on how CPU time is allocated to requests of a domain and thus can determine whether the SLA can be honored or not; an administrator can control them in order to provide service differentiation to the various domains:

- *Routing:* First, an arriving request must be routed through the fabric to one of the servers; we call this the *load balancing* control.
- *Enqueueing:* Once it is sent to a server, a request must be enqueued in a buffer; we call the design of the buffering scheme the *enqueueing* policy. This is the second control at the disposal of the administrator.
- *Scheduling:* The third control is the CPU *scheduling* inside the servers, i.e., deciding from which queue to forward a request for processing.

Note that the controls are distributed: two controls are implemented in the server (but we have many of them in the tier) and one is implemented in the switch fabric (at one or more tiers therein). Clearly *all* three controls have an effect on what percentage of the CPU time in the cluster of servers a given domain can have. In this chapter, we consider fixed load balancing and scheduling policies and focus on the effect of the enqueueing part of the triplet. Our contribution is twofold (a) we propose a simple design for the enqueueing policy that is geared towards distributed implementation: it has low communication and processing overhead, and, (b) we provide "rules of thumb" for when the design can achieve the SLA. We base these rules on both analysis and simulation.

The proposed scheme is similar to Stochastic Fair Queueing [112], in the sense that different number of queues are required per active flow, multiple sessions might end up in the same queue and queues are serviced in a Round Robin fashion. we present a buffering policy that guarantees transmitted bytes ratios without relying on complex scheduling policies or admission control or/and differential dropping [113].

4.1.2 System model, assumptions and Problem Statement

The system model is presented in Fig. 4.2. We begin by omitting the server tier and collapsing the network fabric into a load balancer function. Modeling the routing through the switch fabric as a single load balancer is sufficient, since we focus on the enqueueing policies; which switch(es) were responsible for forwarding the traffic to a preprocessing tier server is irrelevant.

The load balancer is responsible for distributing incoming traffic from a set $D = \{1,...,d\}$ of service domains into a cluster of $N = \{1,...,n\}$ servers, each with a CPU of capacity μ. Every server is modeled as Multiclass-multiqueue system (MCMQS), with $\mathcal{M} = \{1,...,m\}$ the set of queues that is same for all the servers. We assume that $m < d$, i.e., there are not enough queues available to separate traffic belonging to different service domains. This is a reasonable assumption in data centers that offer SaaS, IaaS or PaaS services.

Finally, we assume that the incoming traffic for domain i follows a general arrival and service process with (unknown) arrival rates λ_i and service rates $1/ES_i$, where ES_i is the mean service time of domain i. In addition, we assume non preemptive service for the CPU operation. We also assume that signaling and feedback across the servers take negligible time, however we note that the design of our control system is tailored to minimize these effects.

Control policy definition

A control policy π is a rule according to which control actions are taken at time t. We identify different control policies regarding different control points inside the network.

1. *Load balancing policy* π_r that defines a *forwarding action* $r(t_r)$. Say t_r is the time instant that the load balancer must forward an incoming request to the cluster of dedicated servers. The action space is the set $N = \{1,...,n\}$ of all the servers and $r(t_r) = i \in N$ at time t_r if server i is selected.

2. *Enqueueing policy* π_q that defines *Enqueueing Action* $q(t_q)$. Say t_q is the time instant that a server accepts a request from the load balancer. An enqueueing action determines the queue to which the request is forwarded. The action space is the set $M = \{1,...,m\}$ of all the available queues and $q(t_q) = i \in M$ at time t_q if queue i is selected.

3. *Scheduling policy* π_s that defines *Scheduling Action* $a(t_s)$. Say t_s is the time instant that a server CPU finishes request execution and is ready to accept a new request for service. The action space is the set $M = \{1,...,m\}$ of all the available queues and so $a(t_s) = i \in M$ at time t_s if queue i is selected.

The vector (π_r, π_q, π_s) is collectively referred to as the control policy π.

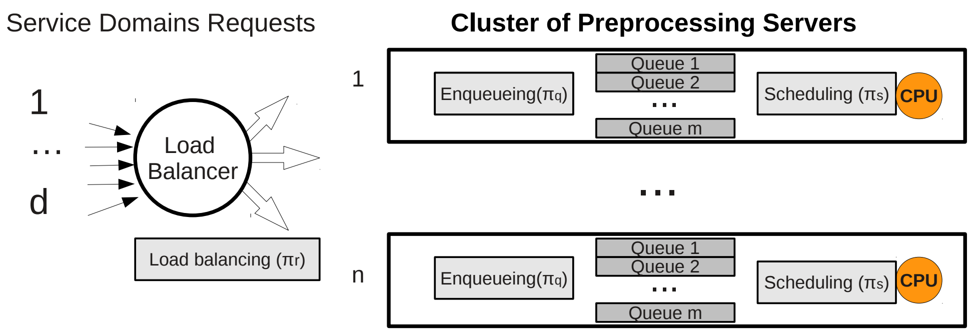


Figure 4.2: System Model for BQS.

For server j, we define the function $F_i(t)$ as the amount of time in $[0,t)$ that the server CPU was allocated to domain i, under policy π. Then for the total CPU power of the cluster of servers, we define the **total utilization** for domain i under policy π as:

$$u_i(\pi) \triangleq \liminf_{t\to\infty} \frac{\sum_{j=1}^{n} F_i(t)}{n \cdot t} \tag{4.1}$$

In the above definition we use liminf since we don't know a priori that the policy will reach steady state.

Problem Statement

The formal definition of the objective we study is the following. *Let $0 \leq p_i \leq 1$ be CPU utilization percentages defined in the SLA for every domain i. Design a policy π that will achieve the following objective:*

$$u_i(\pi) = \min\{\lambda_i \cdot ES_i, p_i\}, \forall i \in D \tag{4.2}$$

Roughly speaking, this is the same objective as the one studied in Chapter 3. The SLA in Equation 4.2 states that each domain must be given a predetermined CPU time percentage over a large time period, unless the request rate is not sufficient for this, in which case it is enough to serve all requested traffic from that domain.

4.1.3 Proposed Policy

First note that because of the joint control definition, the three control actions depend greatly on each other. For example the performance space of a dynamic Weighted Round Robin scheme used as the scheduling policy π_s is in correlation with the queueing dynamics that are dictated by the enqueueing policy π_q. In addition, there is a large set of system configurable parameters like the number of queues, or the number of service domains that greatly affect the performance space.

For these reasons, we focus only on the enqueueing policy π_q. We investigate the single server case ($n = 1$) and we gain the insights into the queueing dynamics effects in the system performance. The accompanying scheduling policy will be plain Round Robin. The aim is to avoid feedback signals

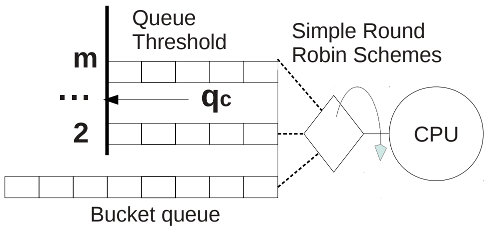


Figure 4.3: BQS queue Structure for CPU guarantees.

analysis between different tiers in the data center and feedback signals for centralized control of the cluster of servers. In addition, with simple scheduling we avoid designing scheduling algorithms that require knowledge of traffic statistics and we avoid the correlation analysis between enqueuing and scheduling decisions.

The Bucket Queue System (BQS)

Queueing configuration: The server is configured to accept traffic from all the domains, in a set of $\mathcal{M}$ queues. Set for all queues $i : 2 < i < m$ a queue limit q_c. We will use this threshold to limit the corresponding queue sizes, while queue $i = 1$ is allowed to grow without constraints. We call this queue the *bucket queue* (B queue); the configuration can be seen in Figure 4.3.

Operation: In the case where an incoming request from domain i must be enqueued, because the CPU is busy upon its arrival, the following actions take place: If the domain is underutilized and the minimum queue size does not exceed the limit q_c, the policy forwards the request to the queue with the minimum queue size. In all other cases the request is forwarded to the bucket queue. The algorithm operation is formally described as Algorithm 1.

Focus on the server CPU operation and let t_k denote the time instant that the CPU finished servicing a request and that the next request it will receive service comes from domain i. We define $u_i(t_k, BQS)$ to be the CPU utilization domain i received up to time t_k under policy BQS. Between two successive service events t_k and t_{k+1}, the following is true: $t_{k+1} = L_k + Z_i^k + t_k$, where $L_k \in R$ is a random variable denoting the CPU idle time between the service events. Also Z_i^k denotes a random variable of the service time the request of domain i received within this interval, while $Z_{j \neq i}^k = 0$. The CPU utilization performance can be expressed with the following closed recurrent form: $u_i(t_{k+1}, BQS) = \frac{t_k}{t_{k+1}} u_i(t_k, BQS) + \frac{Z_i^k}{t_{k+1}}$, $\forall i \in D$. Clearly BQS policy and the accompanying scheduling policy that will be adopted (Round Robin in our case), at any instant t_k affect the Z_i^k rv and so the utilization every domain will receive.

Intuition: When the system operates in conditions of overload, when $\sum_{i=1}^{d} \lambda_i / \mu \geq 1$, the B queue is unstable, while the remaining $m - 1$ queues remain stable. This way, traffic injected to the stable queues, receives guaranteed throughput while traffic injected to the B queue only receives a proportion of throughput in relation to what is injected. The policy is using a simple control mechanism to improve the utilization of domains that are left behind their goals, while penalizing those that have received more service than agreed.

Algorithm 1 BQS Algorithm

```
$t_k$ : *enqueueing decision instance*
$Q_j(t_k)$ : *the queue size of queue j in time.*
*Calculate* $u_i(t_k, BQS) < p_i, Q_j(t_k)$
**if** $u_i(t_k, BQS) < p_i, Q_j(t_k) \leq q_c$ **then**
    $q(t_k) = \arg\min_j Q_j(t_k)$
**else**
    $q(t_k) = 1$
**end if**
```

4.1.4 Policy Evaluation

With respect to simplicity of implementation, we examine the control policy triplet $\{-, BQS, RR\}$ i.e., single server, enqueueing is BQS and CPU scheduling in the server is Round Robin.

Theoretical Considerations

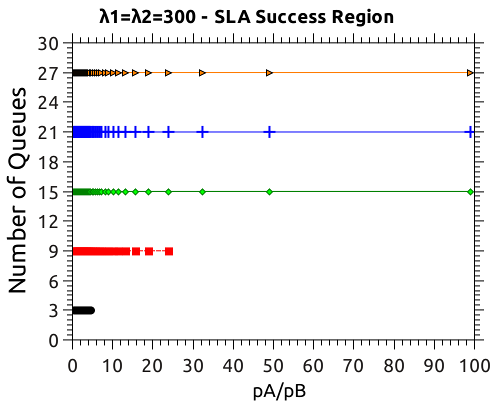
We will consider a single server with m queues served by a CPU of capacity μ in a round robin fashion, such that each queue receives service with rate μ/m. The $m-1$ queues are bounded (a job is routed to them only if their backlog is below the threshold q_c) and one is unbounded. We will study the case of two service domains, in order to derive the conditions for SLA achievability, i.e. the set of (p_1, p_2) targets for which the SLA is achievable under given -but unknown- conditions $(\mu, \lambda_1, \lambda_2)$. Generalizations to many domains and many servers are left for future work.

Let u_1, u_2 denote the utilizations and T_1, T_2 the throughputs, which are related in the following way $u_i = \frac{T_i}{\mu}$. Also, let $a_1, a_2 \in [0, 1]$ be the long-term average traffic splits of the arrivals, i.e. $a_1\lambda_1$ is the traffic directed to the B queue and $(1-a_1)\lambda_1$ the traffic directed to the $m-1$ queues for the service domain 1. In what follows, we will attempt a fluid analysis omitting the details regarding the arrival processes and avoiding the complications of a discrete time analysis.

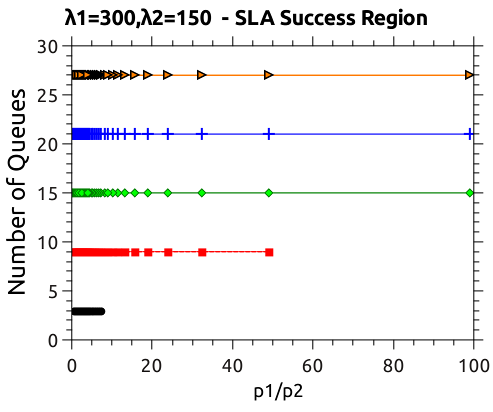
We derive necessary and sufficient conditions for the feasibility of the SLA target, under the condition that the queues are configured as explained above. However, we do not show that the algorithm indeed converges to the proper routing coefficients. The fact that the algorithm can achieve this feasibility region, will be shown by simulations. We have two cases that regard the comparison $\lambda_1 + \lambda_2 \lesseqgtr \mu$.

When the system is stable (i.e. $\lambda_1 + \lambda_2 \leq \mu$), the SLAs are always achieved, since the throughput of each user equals what is injected into the system, thus the first term of the minimum function in eq. (2) is always achieved.

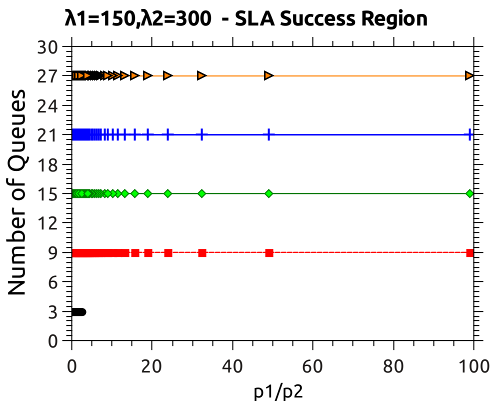
When the system is unstable, (i.e. $\lambda_1 + \lambda_2 > \mu$), the B queue is unstable, but the $m-1$ queues remain stable. We will inspect two further cases, a) when both domains request more traffic than their corresponding target $\lambda_i > \mu p_i$ and b) when one of the two requests more, but the other less. We also omit c) the symmetric to b and d) the case where none requires more than the target, which contradicts the instability condition.



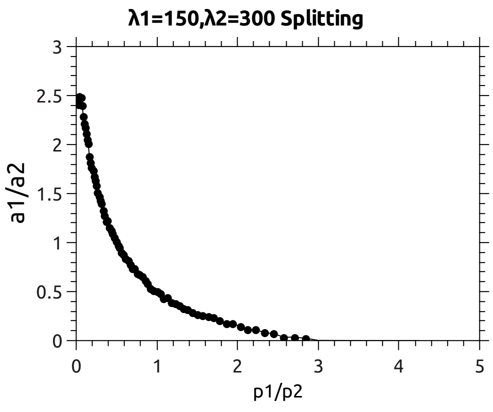
(a) SLA success region



(b) SLA success region



(c) SLA success region



(d) Splitting vs performance

Figure 4.4: BQS performance for domain requesting 0.7-0.3 of CPU Utilization - $\mu = 200$ requests

Case a: Assume $\lambda_1 > \mu p_1$ and $\lambda_2 > \mu p_2$. Note that the flow is conserved in the $m-1$ queues, which gives

$$(1-a_1)\lambda_1 + (1-a_2)\lambda_2 = \frac{m-1}{m}\mu. \tag{4.3}$$

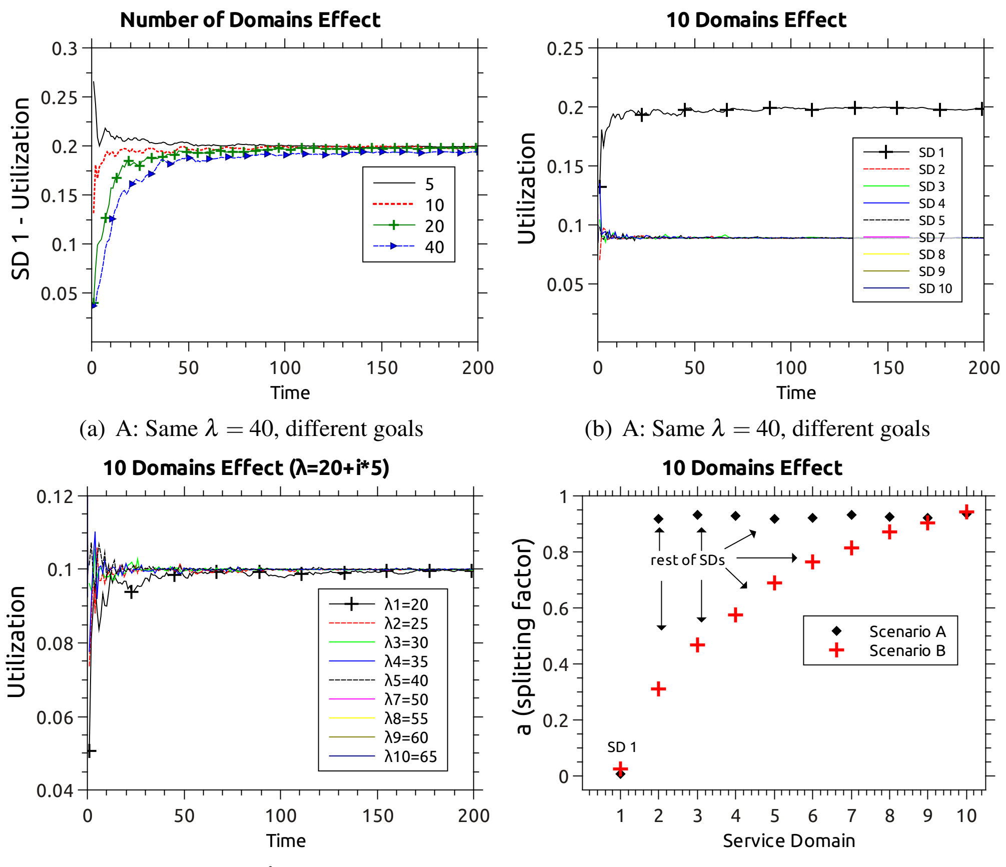
The service in B queue is proportionally allocated to the two domains:

$$\mu_1 = \frac{a_1\lambda_1}{a_1\lambda_1 + a_2\lambda_2}, \quad \mu_2 = \frac{a_2\lambda_2}{a_1\lambda_1 + a_2\lambda_2}$$

and the utilization of domain i should be

$$\begin{aligned} u_i &= \frac{(1-a_i)\lambda_i + \frac{a_i\lambda_i}{a_1\lambda_1+a_2\lambda_2}\frac{\mu}{m}}{\mu} \\ &\overset{(4.3)}{=} \frac{(1-a_i)\lambda_i(\lambda_1+\lambda_2-\mu) + \lambda_i\frac{\mu}{m}}{\mu(\lambda_1+\lambda_2-\frac{m-1}{m}\mu)}, \quad i = 1,2. \end{aligned} \tag{4.4}$$

The following are a set of necessary and sufficient conditions for the SLA to be satisfied under



(a) A: Same $\lambda = 40$, different goals (b) A: Same $\lambda = 40$, different goals

(c) B: Different λ, same goal (d) Proportion of traffic routed to B queue

Figure 4.5: Number of Domains effect ($\mu = 200$ requests)

the stated conditions (instability and both users providing sufficient arrivals):

$$\begin{cases} p_1 = u_1 \\ p_2 = u_2 \\ (4.3) \end{cases}$$

Dividing the first two, and using (4.4) and $p_2 = 1 - p_1$

$$\frac{p_2}{p_1} = \frac{\mu(\lambda_1 + \lambda_2 - \frac{m-1}{m}\mu)}{(1-a_1)\lambda_1(\lambda_1+\lambda_2-\mu)+\lambda_1\frac{\mu}{m}} - 1, \quad a_1 \in [0,1].$$

Denote $M \doteq \max_{a_1\in[0,1]} \frac{p_2}{p_1}$. Clearly this is achieved by $a_1 = 1$, in which case

$$M = \frac{\mu + m(\lambda_1+\lambda_2-\mu)}{\lambda_1}. \tag{4.5}$$

Note that M is the maximum achievable ratio of target utilizations under the conditions above and it can be achieved by an omniscient randomized enqueueing policy which selects properly a_1, a_2.

Case b: W.l.o.g. assume $\lambda_1 < \mu p_1$ and $\lambda_2 > \mu p_2$, thus an SLA-satisfying enqueueing solution will yield $(u_1, u_2) = (\lambda_1/\mu, p_2)$. Note, that if non-negligible fluid from domain 1 is routed to the bucket, then the throughput of domain 1 will be less than what sent, in which case the SLA has failed. Thus, $a_1 = 0$ and we obtain a first condition:

$$\lambda_1 \leq \frac{m-1}{m}\mu.$$

The domain 2 will receive the remaining CPU allocation, thus the SLA is satisfied if $p_2 \leq \frac{\mu-\lambda_1}{\mu}$, from which we conclude

$$\frac{p_2}{p_1} \leq \frac{1-\frac{\lambda_1}{\mu}}{\frac{\lambda_1}{\mu}} = \frac{\mu-\lambda_1}{\lambda_1} = M \tag{4.6}$$

which does not depend on m.

Verification of Theoretical Considerations

In Fig. 4.16 we present the maximum ratio of p_1/p_2, for which the SLA is satisfied, in the following scenario: a single server receives traffic from two domains. The service rate of the server CPU is set equal to $\mu = 200$ requests per time unit (we assume that time is a dimensionless quantity). The criterion we apply in order to note SLA success, is $u_i(\pi) \geq 0.95 \min\{\lambda_i/\mu, p_i\}, \forall i \in D$, which is the objective defined in eq. 4.2 plus a $\pm 0.05\%$ tolerance interval.

In Figs. 4.4(a), 4.4(b) and 4.4(c), we present how the SLA success region is affected by increasing the number of queues. The main outcome is that when m, the number of queues, is sufficiently large the algorithm is able to achieve the SLA even for extreme CPU differentiation goals (e.g., 90-10%). The value of m that guarantees extreme targets depends also on the relation between λ_i and μ, when the system is unstable. When the system is stable, the targets are always achieved.

For example, as we can see in Fig. 4.4(a), with 3 queues the SLA is achieved for a ratio $1/6$, meaning that an SLA $p_i = \{0.86, 0.14\}$ can be satisfied, while with 9 queues this ratio is $1/25$ meaning $p_i = \{0.96, 0.04\}$ can be satisfied.

Fig. 4.4(d) clearly validates the theoretical considerations of the previous section. We present the ratio a_1/a_2 for the two domains in the case where $\lambda_1 = 150$ and $\lambda_2 = 300$ requests per time unit (scenario of Figure 4.4(c)). As we can see, domain 1 takes the maximum CPU share it can receive and the maximum ratio is achieved, when domain 2 forwards all its traffic in the B queue. All the above observations are summarized as a *Rule of Thumb 1* in Section 4.1.5.

Increasing the number of domains: A key advantage of *BQS* policy is that for a large number of domains, a small number of queues is sufficient in order to meet the objective of eq. 4.2. We present simulation evaluation for two scenarios that a service administrator faces when the system is in overload conditions:

Scenario A: *An increasing number of domains request server resources. Service domain 1 is the most prominent client and the administrator wants to guarantee that during overload periods it receives* 20% *CPU share while the rest divide the remaining* 80% *equally.*

Configuration: $\lambda_i = 40$ and $\mu = 200$ for every domain i, while $m = 5$ is the number of queues (including the B queue). In Figure 4.5(a) we increase the number of domains and in Figure 4.5(b) we present in more detail the case where $d = 10$. In both figures we can see that BQS clearly meets the objective. Since we are in overload, regardless of the relationship between arrival and service rates and also regardless of the range of desired percentiles, the enqueueing algorithm satisfies the SLAs, i.e., domain 1 always receives the requested 20% while the remaining domains take their equal share.

In Figure 4.5(d) we can see that in scenario A (black bullets), a_1 factor of domain 1 is approximately equal to zero, while for the rest of domains it is almost equal to 0.9. This means that B queue serves about 10% of domain 1 traffic and the majority of other domains traffic.

Scenario B: *Again an increasing number of domains request server resources. The administrator wants to guarantee fair allocation of CPU power to all the domains, despite any arrival rate diversifications.*

In Figure 4.5(c) BQS policy also meets the objective and fair allocation is provided in the scenario where the arrival rate of every domain is different. The simulation configuration is $\lambda_i = 20 + i * 5$ and $\mu = 200$ domain while again there are $m = 5$ queues. As we can see in figure 4.5(d) for the B scenario case (red crosses), the mechanism to achieve the SLA is again the operation of the B queue. If the traffic from a domain is higher than the others, a higher percentage of its requests is served by the B queue to keep allocating CPU cycles fairly.

Concluding remark: Besides the above paradigms, an extended set of simulations was performed, for various scenarios with increasing number of domains, number of servers and arrival and service rate variations. The main outcome of the simulation procedure is that a small number of queues is sufficient for the simple proposed control mechanism to be effective, in the sense that the majority of SLA configurations can be achieved. In the above theoretical and practical considerations analysis, we presented the performance of the algorithm for the case where the number of servers is $n = 1$. Thorough theoretical investigation for the case where the BQS is applied in a cluster of servers will be presented in future work.

4.1.5 Practical Considerations

Due to the large number of parameters that affect the algorithm performance and the large number of trade offs existing, we summarize the evaluation procedure in the following rules of thumb, that can be used by a system administrator. Say d is the number of domains, λ_i is the expected arrival rate of domain i and μ is the service rate of the CPU:

Rule 1: *What is the maximum ratio we can achieve and how many queues are needed?* The answer comes directly from equations 4.5 and 4.6. An administrator can make a priori an arrival and service rate estimation of the expected traffic and apply these equations. These equations will show if the desired utilizations, defined in the SLA contract, will be satisfied. If the estimations are wrong, real time corrections with adding/removing of queues can fix the ratio M to the desired value. In the case where $d > 2$, the generalization of eq. 4.5 yields that the utilization domain i will receive under overload is:

$$u_i = \frac{(1-a_i)\lambda_i + \frac{a_i\lambda_i}{\sum\lambda_i - \frac{m-1}{m\mu}}\mu}{m\mu} \tag{4.7}$$

This is an upper bound that an administrator can use to jointly design the SLA for all the domains. This way our formula depends only on a_i which is better, but not conclusive. The administrator can use it as a parameter to bound u_i by setting $a_i = 1$ or $a_i = 0$.

Rule 2: *What queue limit to set?* In saturated conditions, if we don't use the queue limit q_c, all queues would be unstable and thus no speeding up could be applied for the underutilized customers. The queue limit q_c can be set to any "small" value an administrator chooses to avoid idle time.

With the above rules a system administrator is able to provide predefined CPU percentages, to every customer in cases of overload, without any scheduling or load balancing concerns and without knowing a priori (or posteriori) any arrival or service statistics. The main advantages of our approach are that no requests are dropped and that a limited number of queues is sufficient to provide the utilization defined in the SLA. The only necessary tool applied is CPU monitoring. Furthermore, in actual systems with injective allocation of domain/queue, BQS algorithm could be easily implemented, if *B* queue was added as a new software component, without disrupting the overall queue structure.

4.2 BQS: Throughput guarantees in Virtual Wireless Networks

In the following we present how software routers can be used in native 802.11 APs, without requiring any modifications on the driver and the client side in order to handle VNet flows. In addition, we present how the Bucket Queue System (BQS), can be used to guarantee transmitted bytes ratios, without relying on complex scheduling policies or admission control or/and differential dropping [113].

Note that depending on the virtualization approach, technology already exists to create 802.11 virtual wireless networks. Nevertheless, due to the varying channel conditions, the unpredictable behavior of the wireless medium and limitations in the access technology, it is very difficult to provide real time guarantees by means of delay, throughput etc., while at the same time differentiate services effectively between competing virtual networks.

As presented in chapter 2, the CONTENT [8] solution focus on converged virtual wireless networks, that explore multi-domain virtualization. The goal of multi-domain virtualization is to build end-to-end paths from the access network up to the virtualized data center and allow for seamless orchestrated on-demand service provisioning. The motivation for this work comes from such environments, where the end-to-end virtual path terminates not in the backhaul ethernet network, but to the virtualized wireless 802.11 APs.

In order to build 802.11 virtual wireless networks, spatial or temporal sharing of the wireless channel, using beamforming techniques in 802.11*n* MIMO systems [114] or tuning techniques of contention window size parameters and transmission opportunity limits in 802.11*e* [32], have been proposed. In the MAC layer, Multi-SSID virtualization is investigated in works like [115] and [33], to group users by assigning them to different Virtual APs (VAPs), where each VAP uses different SSID. By mapping each VNet to a different VAP, different handling per VNet is feasible (eg. authentication, encryption etc). Although this mechanism is widely used, it is prone to increased channel utilization, due to overhead caused by beacon frames that advertise the multiple SSIDs. In addition, regarding the differentiation objective we study, the previous approaches do not take into the account the dynamics of the wireless channel. This means that the actual throughput achieved per VNet, can significantly vary from the actual goal. To face these limitations, we consider a feedback based approach that adapts the system according to runtime performance. We explain this in more detail in the following. Similarly to [33], where the SplitAP architecture is proposed, we utilize software routers [116] to program forwarding operations. However, we focus on operations between MAC and IP layers, our focus stays on the downlink and our objective is not to allocate fairly the airtime usage to every VNet, but to provide specific throughput ratios by means of transmitted bytes.

We distinguish the traffic between different VNets, based on the VLAN identification. In a native 802.11 AP all the VNet flows must enter, prior to transmission, the single FIFO queue implemented in the driver and so be served in a "First come First served" fashion. A software router, like the Click router [116], can be used to implement user-defined queueing structures and sophisticated buffering and scheduling policies, as shown in Fig.4.6. The Click router is extremely extensible and can be used to perform actions like packet scheduling, traffic shaping, filtering, packet dropping and header

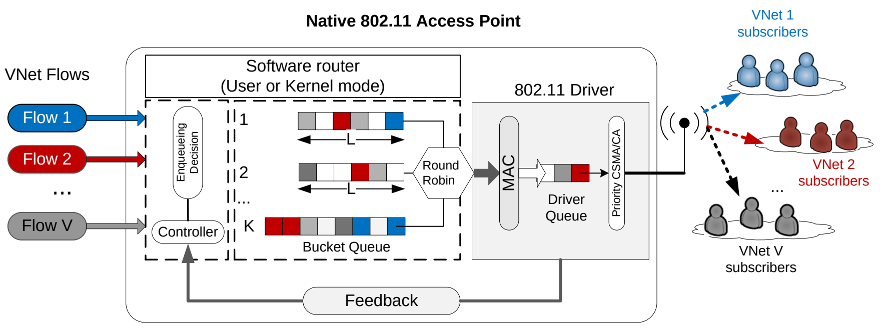


Figure 4.6: Software Routers for Virtual Access Points

rewriting. All these capabilities are valuable tools we explore in order to build the mechanism to handle each VNet differently.

4.2.1 System Model & Problem statement

A single 802.11 Access Point, is logically divided into a number of virtual networks $\mathcal{V} = \{1, 2, \cdots, V\}$ and is equipped with a single 802.11 wireless interface. Let $\mathcal{U}_i = \{1, 2, \cdots, U_i\}$ denote the set of users that are associated with each VNet i. The aggregated traffic from all the users in some VNet, defines the corresponding VNet flow. With respect to practical considerations, we assume that the total number of users does not violate the limits that are set by the AP administrator. The packet arrivals for every virtual flow have a general distribution with unknown mean. Furthermore, we assume operation in a completely stochastic environment where interference, collisions and congestions are difficult to model and so the corresponding parameters that could affect our model are considered unknown.

Let $B_i^j(t)$ denote the number of bytes transmitted to user j associated with VNet i. Then, the aggregated downlink traffic $B_i(t)$ of VNet i up to time t is equal to $B_i(t) = \sum_{j=1}^{|\mathcal{U}_i|} B_i^j(t)$. We also define $B(t) = \sum_{i=1}^{|\mathcal{V}|} B_i(t)$ as the total bytes transmitted by all users and by all VNets until time t and weight w_i as the proportion of $B(t)$ we wish VNet i to receive in the long run, such that $\sum_{i}^{|\mathcal{V}|} w_i = 1$. The problem under examination is to find a policy that in the long run (sufficiently large t) guarantees that every VNet i will have $w_i \cdot B(t)$ bytes transmitted. For example, in the case of two VNets and a single AP, if 10 GB were transmitted until time t, $w_1 = 0.2$ and $w_2 = 0.8$, the policy guarantees that 2 GB have been transmitted for VNet 1 clients and 8 GB for clients of VNet 2, without being affected by the channel conditions or arrival rate variations. In the following, the BQS policy presented in the previous section will be evaluated towards this goal.

4.2.2 The Proposed Queueing Structure and Solution

A software router is used to implement the following queueing structure in the AP, without affecting the driver. There is a number of queues accessible by all the VNets, where an artificial bound L is set to their size. In addition, we add one more queue, that all the VNets can also access, which we refer as the *bucket queue* and that we allow to grow to infinity (practically the excess load will be dropped if it violates the system limitations). Is up to the policy to decide in which queue to store an incoming packet; in a *limited size* queue or the bucket queue. All queues are served in Round Robin fashion and the first packet of every queue is forwarded to the single FIFO queue implementation of the driver. The queueing structure is depicted in Fig.4.6.

Bucket queue with Limited Size Queues (*BLSQ*): *For every incoming packet of VNet flow i that must be enqueued: If the number of transmitted bytes for VNet i are less than* $w_i \cdot B(t)$ *(it is below its goal) then the policy uses Join the Shortest Queue (JSQ) [117], between the limited size queues that have not reached the queue limit; in all other cases the packet is forwarded to the bucket queue*. The algorithm is outlined in Algorithm 2.

What will occur after the packet enters the driver's FIFO queue, is based on factors like interference, collisions and congestions that practically are difficult to model. In contrast to complex scheduler implementations (that require changing the weights of a weighted round robin scheduler or changing the probabilities of a probabilistic scheduler, or apply admission control and drop such packets (e.g [118], [113]), by using the proposed scheme with a simple Round Robin scheduler, the required ratio is preserved per flow. Although, this comes with the effect of packet re-ordering, that is not an issue in practical deployments, since (up to some point) it can taken care by higher layer protocols.

The ratio is preserved by indirectly speeding up the VNet clients in the *limited size* queues and by holding back in the bucket queue the VNet flows, which packets contribute in a ratio higher than the one defined in the SLA. This control decision is agnostic to the number of users per VNet, it only requires knowledge of the aggregated traffic served per VNet. We also point that although the decision is made per packet, this decision is agnostic to the packet size distribution, since it only requires knowledge of the bytes transmitted. If a packet arrives from a "suffering" VNet, we enqueue it in the limited size queues, independently of its size.

4.2.3 Benchmarking in a Wireless Testbed Environment

A large set of experiments were conducted in order to present the algorithm's efficiency and demonstrate how system and statistical parameters affect the algorithm performance. We present an indicative set of experiments using a single AP; extensions for distributed/centralized operation in a clustered environment are left for future work.

The environment

A prototype solution was implemented in a Commell node in the NITOS outdoor wireless testbed [76]. The outdoor NITOS wireless testbed (50 wireless nodes) was selected to demonstrate the algorithms efficiency in realistic conditions, since the testbed operates in an urban area and additionally, multiple experiments run concurrently from other experimenters. Thus, all our experiments faced

Algorithm 2 BLSQ - Algorithm Description

```
$w_i$ : transmitted bytes percentage goal for VNet $i$
$t$ : enqueueing instant
$L$ : a queue limit
$X_j(t)$ : the queue size of queue $j$ at time $t$ ($j$ not the bucket)
$Update\ B_i(t)$ (the measured throughput percentage)
if $B_i(t) < w_i \cdot B(t), \exists j : X_j(t) \leq L$ then
    JSQ between the limited size queues
else
    enqueue in the bucket queue
end if
```

uncontrollable collisions and interfering conditions from neighbor APs and users. As we show in the following, interference was also created on purpose to further stress the algorithm.

The Commell node selected is equipped with Core 2 Duo 2.26 GHz CPU, 2G DDR3 RAM, two Gigabit network interfaces, Atheros 802.11$a/b/g/n$ wireless interfaces, multi-band 5dbi and operates both on 2.4Ghz and 5Ghz antennas. The setup we present in the following, uses a single interface (802.11a in the 5Ghz band), while the queueing structure as long as the buffering control mechanism were implemented using the Click Modular Router [116]. The *BLSQ* enabled AP was used to send traffic to users that were logically associated with different VNets, using VLANs. The experimentation model is depicted in Fig.4.7.

Experiments Parameterization: In every experiment, specific percentage goals were set for every VNet. The arrival process was created with *iperf* (UDP traffic) and the measurements were collected using the *OML library*. The payload was set 1470 bytes for every packet and the AP was tuned to transmit in a physical rate of 12 Mbps (similar results were obtained when auto-rate adaptation was used). In all the experiments we wanted to stress the *BLSQ* algorithm in heavy load conditions, so the arrival rate for every user was set equal to 7 Mbps.

In order to investigate how the system and statistical parameters affect the algorithm performance, we used a basic scenario and each time we varied a parameter to investigate its effects on performance. The basic scenario is the following.

Basic scenario parameters: We used a single AP and we defined 3 VNets; VNet 1 (1 user), VNet 2 (2 users) and VNet 3 (1 user). The VNet goal vector was set $(20\%, 30\%, 50\%)$, while we made statistics updates whenever a packet was send successfully (ACK received). In the software router we used 3 queues in total (the bucket queue plus 2 limited size queues). In all the experiments presented, the bucket queue size was set to the maximum available (1,000,000 packets) and the limit for the limited size queues L was set equal to 100. This number was selected after experiments that presented good performance, but actually the rule of thumb is to select a number that will keep the limited size queues short, in order to be able to *speed up* the VNet we want.

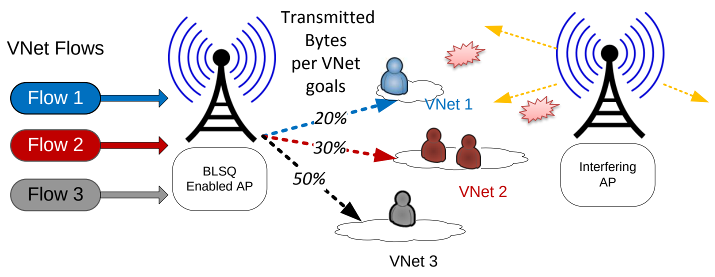


Figure 4.7: Experimentation model

Basic scenario

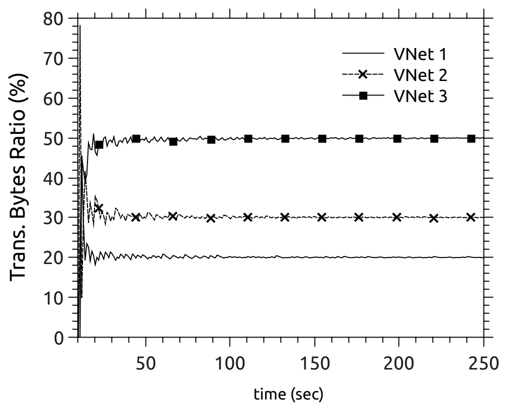
Fig.4.8(a) - This is the basic scenario experiment, where the VNet goal vector was set $(20\%, 30\%, 50\%)$. Fig.4.8(a) is used to demonstrate that the algorithm provides the requested ratio of transmitted bytes per virtual flow, without requiring knowledge of the channel conditions or a priori knowledge of the arrival process. It is interesting to note that VNet 2 serves two users. Since, the mechanism only checks the total number of transmitted bytes per VNet, in order to adapt its control decision, each individual user will receive a percentage proportional of the load transmitted to him, to the load transmitted to all the other users in the same VNet. The number of users per VNet and their transmitting rates, plays no role in convergence as long as their aggregated rate can satisfy a feasible goal. As we present in the following, for eg. a goal vector like 98%,1%,1% may not be feasible.

Varying the Arrival Rate

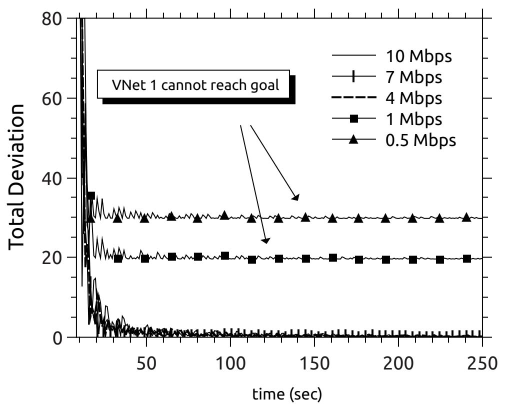
Fig.4.8(b) - In Fig.4.8(b) the total deviation from the goal is depicted $(\sum_{i=1}^{|\mathcal{V}|} |\frac{B_i(t)}{B(t)} - w_i|)$. We remind $B(t)$ is the total transmitted bytes and $B_i(t)$ is the ratio VNet i achieved until time t. In this experiment we used the basic configuration, where we varied the arrival rate of the traffic destined to the user of VNet 1, from 0.5 to 10. As we can see in Fig.4.8(b) the algorithm operation is insensitive to arrival rates variations, as long as the goal is feasible. When the sending rate to user/VNet 1 is very low (eg. 0.5 or 1 Mbps), the goal of 20% is infeasible. Nevertheless, in all other cases the goals are achieved. Similarly, if packet sizes significantly vary, because of the feedback control, the algorithm operation guarantees the requested transmitted bytes ratio.

Varying the number of VNets

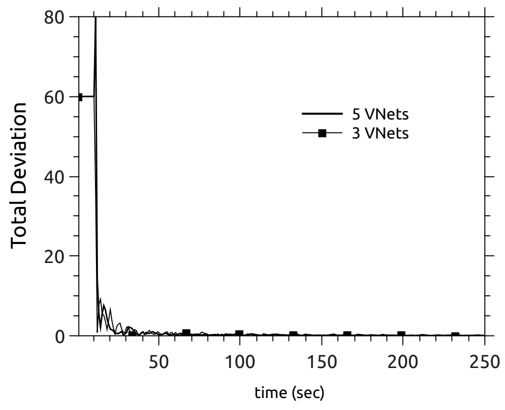
Fig.4.8(c) - In this setup we varied the number of VNets from 3 to 5 (one user per VNet), where the goal was to load balance the traffic between the VNets. The number of queues in both cases was set equal to 2, where again we present the total absolute deviation. One of the main features of the algorithm is that is able to differentiate, using a number of queues significantly less than the number of VNets. As we can see in this example, we can perform load balancing between the VNets using only 2 queues (plus the bucket queue). We also stressed the algorithm in different setups with multiple VNets with different goals defined and less queues. Again we report that the feedback mechanics



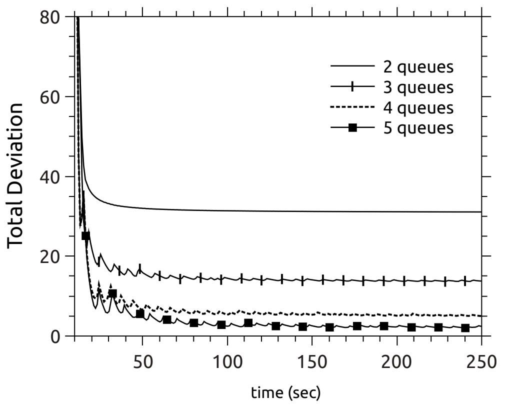
(a) Example scenario



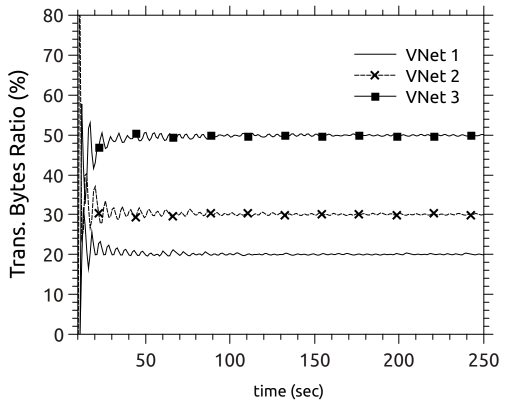
(b) Arrival Rate effect



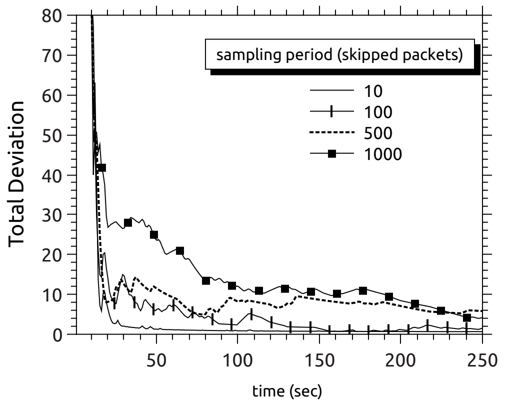
(c) Number of VNets effect



(d) Number of queues effect



(e) Interference effect



(f) Sampling effect

Figure 4.8: Algorithm performance in different scenarios

of the algorithm took into the account the dynamics of the system and adapted transmitting rates accordingly.

Varying the number of Queues

Fig.4.8(d) - As already discussed, the only requirement for the policy to work is to have a feasible goal defined. For e.g. a goal vector $(98\% - 1\% - 1\%)$ may not be feasible for the policy; this depends both on the queue structure and the system dynamics. One simple way to increase the feasibility space (succeed more *disperse* goals), is to increase the number of limited size queues. The bucket queue policy, gives more opportunities to adjust the "suffering" VNet percentage to a higher value, when increasing the number of the limited size queues. In Fig.4.8(d) the SLA throughput goal vector was set equal to $\{15\%, 15\%, 70\%\}$ and we varied the number of the limited queues. As we can see, for a small number of queues, the policy fails to meet the objective, nevertheless as the number of limited size queues increases the total deviation decreases. The reason is that by increasing the number of the limited size queues, we give more opportunities to the "suffering" VNets to increase their percentages in each round.

Increasing interference

Fig.4.8(e) - Increasing the number of users, results in increased congestions, increased interference and increased number of collisions. In such environment, in the same time window more retransmissions (MAC originating) take place. We observed these phenomena, by using the same configuration as the basic setup with the three VNets, while adding an additional interfere AP on the same frequency channel, transmitting to a single interfering node in the maximum rate (using Rate adaptation and downlink sending rate 50 Mbps). As we can see the feedback mechanics of the algorithm takes this into the account and adapts transmitting rates accordingly. In Fig.4.8(e) we see that increased interference results in "slightly" slower convergence, but does not affect convergence itself. This is very important since interference may not have a similar, uniform effect in all users/VNets transmissions. The dynamic feedback-based operation of the algorithm is the one that stabilizes the percentages achieved by each VNet around the desired value.

Increasing the statistics update period

In order to avoid updating the statistics based on all the transmitted packets information, using a sampling technique, we updated the statistics according to a sampled ACKed packet we select periodically, Fig.4.8(f) presents the effects of increasing the sampling period (in number of skipped packets). This directly correlates with how updated is the information that is used by the mechanism, the time instants a buffering decision is made. As we can see, and is also expected intuitively, there exists a trade-off between the signaling overhead required and the convergence speed.

Delay performance

Fig.4.9 - In Fig.4.9 we use the basic configuration to present the delay performance of every VNet, when the *BLSQ* is used and we compare it with the case where all the requests from all VNets, are

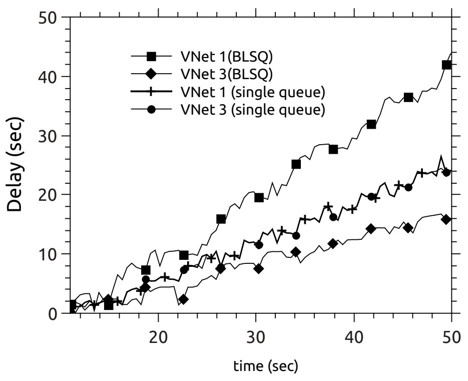


Figure 4.9: Delay

entering directly the single FIFO queue of the driver. As expected different VNets experience different delay. Note that the delay in Fig.4.9 is increasing staggeringly, since total arrival rate (28Mbps) over-exceeds the sending rate (12Mbps, in practice $\sim$ 10Mbps). Nevertheless, these values are not representative for a system that operates in stability region. Using *BLSQ* the delay for VNet 1 (requesting 20% throughput) is increased, in contrast with the native single queue implementation, while the delay for VNet 3 (requesting 50% throughput) is reduced. The reason is that most of the packets from VNet 1 are send from the bucket queue to the driver queue, while traffic to VNet 3 is mostly using the limited size queues, thus on average the delay will be better.

We also note that because of the algorithm's logic, in all cases and for all the VNets, we experience increased jitter. A factor that affects jitter and the order of delivered packets is the queue limit we set to the limited size queues. In this direction, we plan to enhance the *BLSQ* algorithm with a packet dropping scheme and congestion marking schemes, while also schemes that can safely drop over-delayed packets (in UDP traffic there are no retransmissions, but in TCP traffic when some of the packets in the sequence are over-delayed, these packets will be discarded and retransmissions will take place). We also plan to investigate the optimal limit size, that will allow for a smooth jitter distribution, while still providing the ability to differentiate traffic.

4.3 Prediction Techniques for Guaranteed Service Delivery

Like in the first case presented in this chapter, we consider a system that serves D service domains that competing for service resources. In this section, a minor variation of the SLA is studied. The SLA problem we study is the following: allocate a given percentile P_i^u of the CPU resources in the appliance server tier to domain i, when the system is in underload mode; the percentile changes to P_i^o, when the system is in overload mode. In this section, we examine prediction based techniques in order to satisfy the high level objective. Prediction techniques are very well investigated, like in [119],[44],[], []. In [119] the NWS system is presented with a collection of one-step-ahead prediction strategies, to time series and chooses the strategy used for the next prediction dynamically, according to which strategy has been most accurate over recent measurements. The prediction strategies used by NWS currently include running average, sliding window average, last measurement, adaptive window average, media filter, adaptive window media, α-trimmed mean, stochastic gradient, and autoregressive. From [44] it is examined how to estimate model parameters once data have been collected, using a commonly-used method called least squares regression. In contrast to existing work, where estimation and prediction techniques are used to estimate average values, we adjust scheduling probabilities based on system dynamics, in runtime while we try to control the system behavior through ports ON/OFF operations, affecting this way the scheduling probabilities for every domain..

The architecture of the system we study can be seen in figure 4.10. Multilayer switches (or http routers) are used to spread traffic from various domains or service classes, in a cluster of appliances that is responsible for preprocessing. After preprocessing is done, web requests are sent to a cluster of application servers to handle final processing. Server tier is not examined in this paper and we are only interested in the interaction between the appliance tier and the multilayer switches. Of course the appliance tier, is also a server tier, and the same algorithmic approach can be exploited in every processing tier. We just note the processing tier to point the fact in multi-domain architectures, like the ones presented in Chapter 2, multiple tiers of service exist in order to facilitate efficient delivery of services.

4.3.1 Design Requirements

In order to properly set the model on which the controller will operate, several requirements must be met by the appliance operation.

- Each appliance is capable of managing which domain to serve and can accept configuration changes in runtime.
- Each appliance has no knowledge of the state of the other appliances and the domains the other appliances are servicing.
- When a request enters the appliance, the CPU service time it will need is unknown.

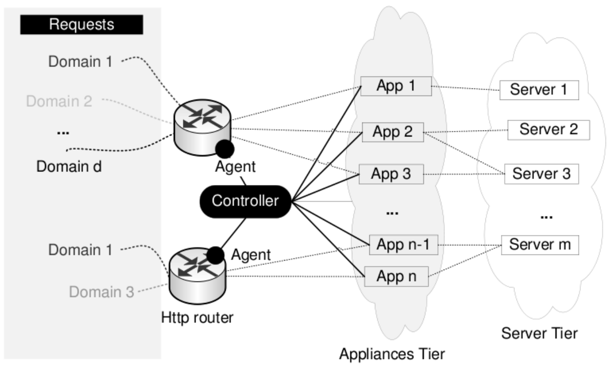


Figure 4.10: *System Architecture for the Predictions Scheme.*

- We model each appliance as a G/G/1 queueing system, served by a *FIFO* CPU scheduler.
- No admission control is assumed; all arriving requests for all service domains must be served.
- No preemption is employed when serving requests at an appliance CPU. Whatever the deviation from the SLA goal, processing a request cannot be stopped.

The FIFO assumption is realistic in computing centers that serve thousands or even millions of domains. Even in less-demanding situations, the FIFO scheduling assumption is a good system approximation for appliances that server more service domains than the internal queues the appliance supports. From a more practcal perspective, since we cannot rely on a CPU scheduler to allocate CPU time to competing service domains, the (only) alternative we have is to *control the rate at which traffic enters an appliance*. We propose a feedback-based control mechanism that is used to spread traffic from the router tier into the appliance cluster; the feedback is based on appliance CPU utilization. In a nutshell, this feedback is used at the router/controller to increase/decrease the rate of traffic from a specific domain sent to an appliance. This action is known in the literature as a router-to-appliance "open or close port operation".

4.3.2 Controller Design

In our model the controller periodically takes a decision every T_d seconds and every time a triggered change occurs between underload and overload mode. At the time of decision t_d the controller gathers runtime statistics through feedback from all the appliances in the cluster and then perform calculations to make a prediction of the CPU utilization vector the cluster must work from this time on. CPU utilization and queue state in each appliance are the statistics of interest, while the prediction is made for an interval of t_p seconds. The output of this calculations is utilization vector U_{PR}.

Since we don't know when the next mode change will occur, the t_p interval for our prediction can be short (e.g., on the order of one T_d interval) or long (e.g., on the order of the entire SLA observation period). T_e is the interval from the beginning of the observation to the time at which we make the prediction.

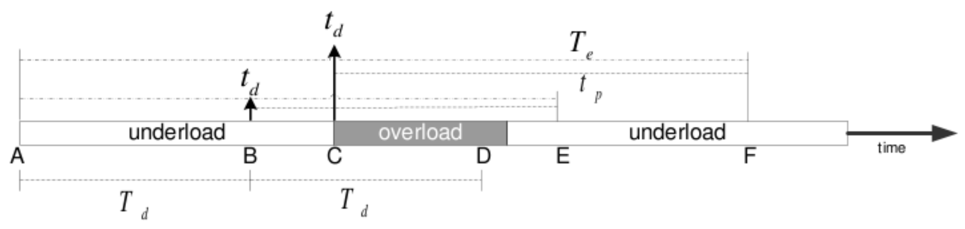


Figure 4.11: Time model. *In moment B the controller makes a decision because period expires. The prediction it makes is for moment E. In time C the controller detects that the system is in overload mode and makes a prediction decision for time F.*

The controller design is depicted in Fig. 4.12. CPU utilization vectors are calculated in *UtilizationMatrix* block, while running queue state of each appliance is stored in *VitrualQueueblock* every time *StateInvestigatorblock* decides that the controller must take a decision. U_{PR} vector is calculated in *BlockA* and port allocation is done in *BlockB*. Virtual queue will be explained in the following section.

After the utilization vector U_{PR} is calculated, we examine how we will spread the load in each appliance per domain. Load is spread probabilistically according to an instantiation matrix created by a Port Allocation Heuristic Algorithm. Also, because the system is distributed, each router in the system can be connected to a different appliances set. For this purpose each router holds a software agent, that locally adapts the controller instantiation matrix independently from other routers, based on the appliances set that it is connected and the domains that is servicing.

4.3.3 Controller Operation Algorithm

1. The system is continuously monitored for mode change or controller decision period expired events. When such events occur, the controller detects the mode of operation (underload/overload) and collects runtime statistics from the appliance tier.

2. The utilization vectors U_j^u (U_j^o if overload) are calculated for all the domains. U_j^u is the CPU utilization domain j received until that moment for underload operation.

3. Let q_{ij} denote the number of requests in the FIFO queue of appliance i for service domain j. In this step, we calculate the number of all the domain requests that are queued in all the appliance queues: $\sum_{i=0}\sum_{j=0} q_{ij}$ and the number of requests for service domain j in the virtual queue: $\sum_{i=0} q_{ij}$.

4. Then the desired utilization for domain j is calculated according to the following:

$$U^u_{PRj} = \frac{T_e}{t_p} * P^u_j - \frac{T_e - t_p}{t_p} * \left(U^u_j + U^u_{qj}\right) \tag{4.8}$$

5. The output U_{PR} vector is used to produce an instantiation matrix with all appliance/domain pairs. In this step the controller “translates” the percentages to ports that routers must open for every appliance/domain pair.

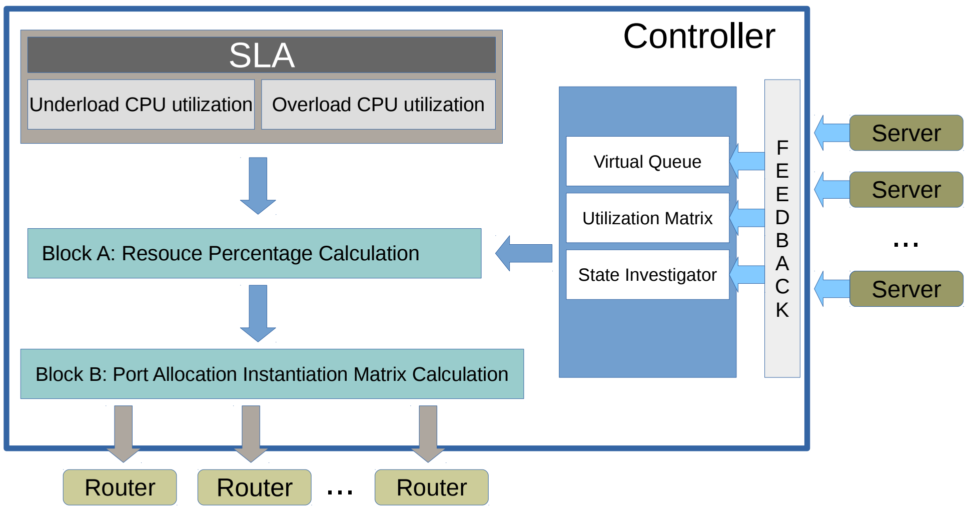


Figure 4.12: Controller Design

Why and How we Use the Queue term

In eq.(4.8), the controller's prediction of the utilization is not based solely on the utilization domains had at the moment of decision t_d. The term U^u_{qj} is an estimate of and accounts for utilization a domain receives due to already queued requests.

$$U^u_{qj} = a * \left[\max \left(b, \frac{\sum\limits_{i=0} q_{ij}}{\sum\limits_{i=0} \sum\limits_{j=0} q_{ij}} \right) \right] * U^u_j \tag{4.9}$$

In the above formula, a and b are two variables that are evaluated by extensive simulations. In this work we present results for $a = 1$ and $b = 0.5$, while in future work we will present evaluation of a and b based on more complex and formal modelization.

The intuition behind this estimation term can be explained as follows: suppose that we have a large number of requests in the cluster queues and all requests belong to domain 1; suppose further that at the moment of decision we are "in" target for this domain then we must intuitively use this "future" cpu cycles usage as already taken to perform our prediction calculations. U^u_{qj} doesn't depend on average service time measurements, but only on straightforward queue state measurements. In Eq. 4.9, the first term is used to control how much important we consider the queue state at decision time, as an additional percentage to utilization U^u_j (or U^o_j).

How we calculate CPU utilization in the cluster

In Eq. 4.8, P^u_j is the target underload percentage for service domain j; in order to calculate the CPU utilization every domain got until the time of decision we use the formula:

$$U^u_j(t_d) = a * \left(U^u_{1j} + U^u_{2j} + \ldots + U^u_{ij} \right) / N + (1-a) * U^u_j(t_{d-1}) \tag{4.10}$$

Table 4.1: Instantiation Matrix

	domain 1	domain 2	...	domain M
App 1	p_{11}	p_{12}	...	p_{1M}
App 2	p_{21}	p_{22}	...	p_{2M}
...	...	...	...	...
App N	p_{N1}	p_{N2}	...	p_{NM}

In Eq. 4.10, $U_j^u(t_d)$ is the utilization that domain j achieved during underload periods (until time t_d); similarly, $U_j^o(t_d)$ is the utilization achieved during overload periods. t_{d-1} is the time the previous decision was made and variable a is the percentage of the total time the system worked for this mode. For example for underload: $a = \frac{\text{last underload period}}{\text{total underload period}}$. Finally, $i = 1, \ldots, N$, where N is the total number of appliances in the cluster.

Instantiation matrix creation

Instantiation matrix is created by translating the utilization vector U_{PR}, to number of ports that must be open per appliance and per domain. For each domain the following calculation is performed: $p_j = R * U_{PRj}$ where p_j is the number of ports the router will open for domain j. The creation of the instantiation matrix is based on an iterative heuristic, where we try to hold the same allocation as possible as in the previous step, in order to have the minimum port open/close operations. Also heuristic tries to have same number of open ports in all the appliances in order to eliminate idle state.

Agents operation

After the controller computes a global instantiation matrix for all service domain-appliances pairs, all local agents in each router are updated with this information. These agents manipulate the "global" matrix locally and independently from other routers. Then this agent in each router is used to perform two operations, Scheduling and Routing.

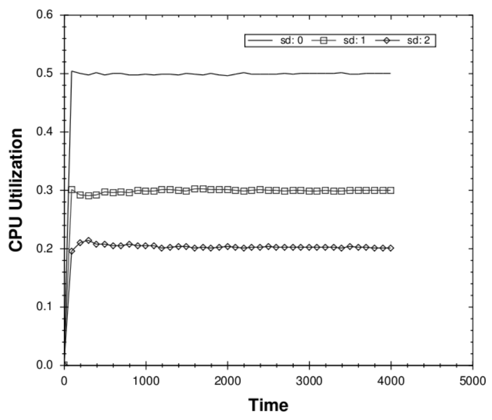
Scheduling Algorithm used: The router will probabilistically select domain j to schedule traffic to an appliance in the cluster with probability $P_{ij}^s = \frac{\sum_i p_{ij}}{R}$ where R is the total number of ports the router can handle and $\sum_i p_{ij}$ is the number of the ports allocated for domain j.

Routing Algorithm used: A request will be sent to appliance i with probability P_{ij} calculated as: $P_{ij}^s = \frac{\sum_j p_{ij}}{R}$. In doing so, we also provide a form of load balancing because traffic is probabilistically distributed among the appliances.

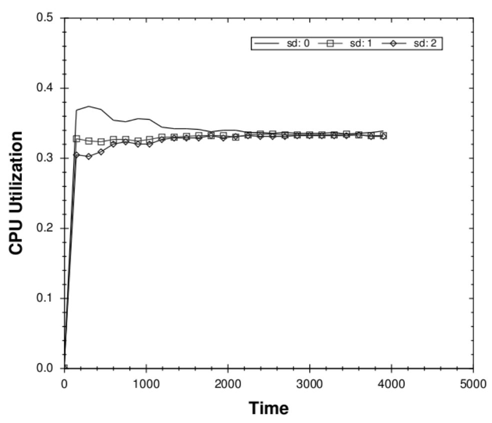
4.3.4 Simulations

We investigate through simulations the following:

1. (**Q1**) whether our proposed mechanism meets the CPU utilization goals in both underload and overload occasions.



(a) $P^u = \{50\%, 30\%, 20\%\}$



(b) $P^o = \{33\%, 33\%, 33\%\}$

Figure 4.13: Prediction Based Performance

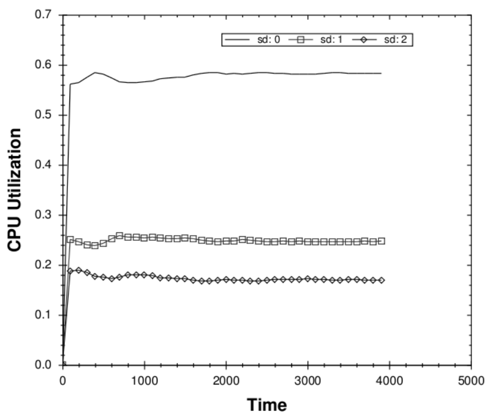
2. **Q2** we compare it with the static port allocation approach.
3. **Q3** we present how controller parameters such as decision or prediction period etc affect our controller performance.

For our simulation purposes, we built a custom, discrete-event simulator in C# language. For the sake of simplicity, incoming traffic to service domain i is modeled as a Poisson process with arrival rate $\hat{I}_i$. In order to avoid side effects as resource starvation, in all simulations presented, we use a minimum number of ports for every domain, no matter what the instantiation matrix is in each cycle of measurements. In our simulations, this percentage is set equal to 10%. The rest of the ports are distributed according to the controller operation described above. Also we consider that there is always enough traffic for each domain and that the SLA defined is feasible, meaning that the traffic for all domains is enough to meet the target CPU utilization.

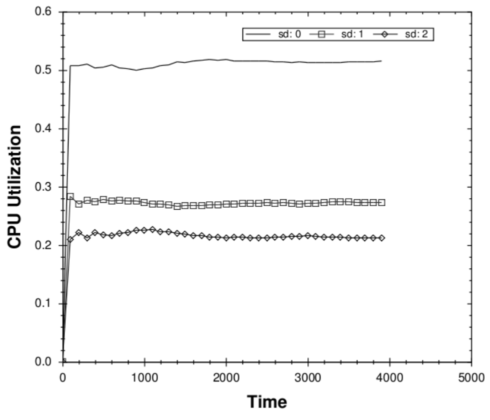
One switch is connected to 3 appliances that form the cluster we investigate. The traffic arrived in router comes from 3 different domains while the SLA defines the underload CPU utilization goal as $P^u = \{50\%, 30\%, 20\%\}$ and overload CPU utilization goal as $P^o = \{33\%, 33\%, 33\%\}$. In order to properly evaluate the mechanism in a stressful environment, we decided to choose different service rates and different arrival rates for each domain while the instantiation matrix is initialized with equal number of ports for all service classes and all equally shared for all the appliances. The total number of ports is set to 1000 while the prediction period in this set of simulations is equal to the controller decision period T_d.

In Figs. 4.13(a)-4.14(b), we provide an answer to questions **Q1** and **Q2**. The static allocation we simulated is different for the underload and overload modes. This means, for example, that, when the system is in underload mode, $1000 * 0.5 = 500$ ports are open for SD1 and $1000 * 0.333 = 333$ ports are open when the system is in overload mode.

As we can see in Figs. 4.13(a) and 4.13(b), the SLA target is clearly met while the system is in underload and in overload mode. Also in comparison to the static allocation shown in Figures 4.14(a)

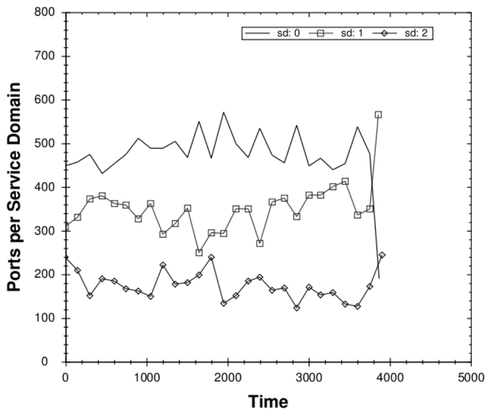


(a) Static allocation: $P^u = \{50\%, 30\%, 20\%\}$

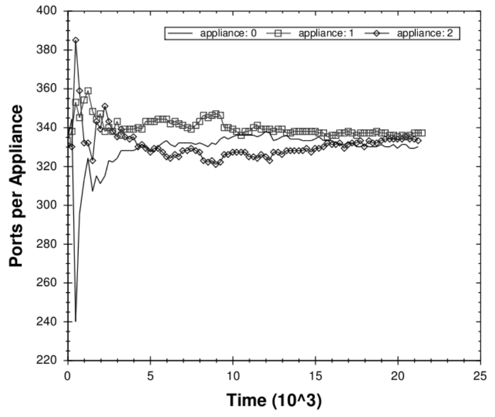


(b) Static allocation: $P^o = \{33\%, 33\%, 33\%\}$

Figure 4.14: Static Allocation Performance



(a) Ports per Service Domain allocation

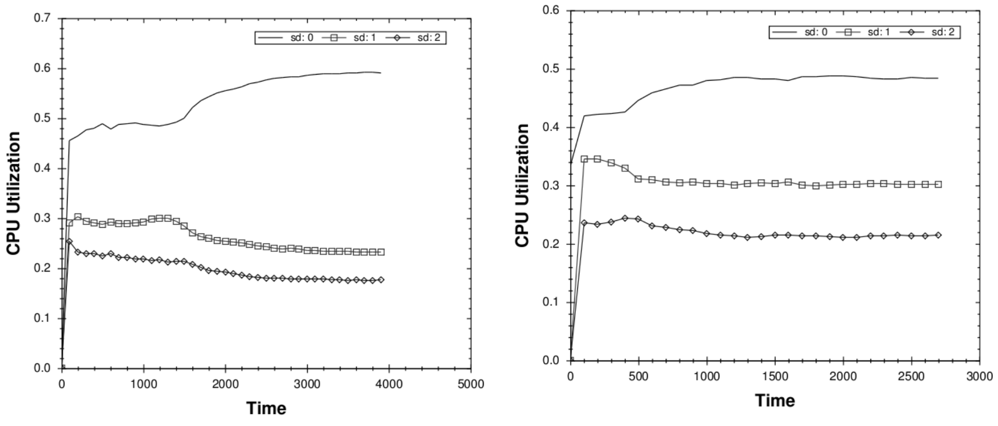


(b) Ports per appliance allocation

Figure 4.15: Ports Allocation

and 4.14(b), our mechanism clearly performs better. In fact our mechanism is even sensitive and adaptable in arrival rates that change in time while the static allocation solution seems to work better only in one occasion, where arrival rates are equal and service rates are equal for all classes of traffic. Static allocation fails to keep up with the fact that SLA defined for overload conditions is different than in underload, it takes no history of CPU utilization under consideration and its performance is highly dependaple on the arrival and service rates of the domains.

Two very interesting observations can be deduced by Fig.4.15(a) and Fig.4.15(b) where we can see the port allocation done in the instantiation matrix per domain and per appliance. As we can see the number of open ports for each domain changes each time the controller must take a decision. Because of the 10% portion of ports that all classes share, we can see that there is always a minimum of ports open to deliver requests for every domain. The iterative heuristic used for the Instantiation Matrix creation tries to share equal number of requests to each appliance and this load balancing behavior can be seen in Fig.4.15(b).



(a) Controller: $P^u = \{50\%, 30\%, 20\%\}$ (b) Controller: $P^u = \{50\%, 30\%, 20\%\}$

Figure 4.16: Prediction period and multiple routers

Figure 4.16(a) is used to answer question **Q3**, and in particular to show how the prediction period affects the controller performance. Our simulations showed that controller meets the goal when predictions are made for medium to short prediction periods while by reducing the decision period by a factor of 4 the controller converges faster to the desired goal.

In Fig. 4.16(b), we study a distributed environment where two routers are connected to two different sets of three appliances each, all servicing requests for three domains. Again, different arrival and service rates where selected for every domain. Here underload mode is presented but goal is reached also in overload conditions.

4.4 Chapter Conclusions

In this Chapter, we proposed stochastic enqueueing and predictions techniques for guaranteed service delivery. In the stochastic enqueueuing approach, we presented an approach to provide guaranteed service delivery to each customer class, by selecting the queue to store incoming requests. This scheme was evaluated in two application scenarios. In server systems, where guarantees are provided on CPU usage, and in 802.11 Access Points, where the stochastic enqueueing technique is used to provide guarantee throughput to individual customer classes. Future plans include study of the algorithm behavior in an environment with multiple Access Points (or multiple servers) and its performance under distributed and centralized control. The proposed scheme is not able to guarantee throughput optimality or QoS, so extensions are planned on these directions also. Delay bounds investigation and better performance regarding jitter will also be part of our future research. In the predictions based approach, in contrast to existing work where estimation and prediction techniques are used to estimate average values, we adjust scheduling probabilities based on system dynamics and a prediction for the future system evolution. For both the enqueueing and prediction techniques no steady state analysis is provided, nevertheless through extensive simulations, we presented that are acceptable approximations to the differentiation objectives.

Chapter 5

Applications of Service Differentiation in Heterogeneous Virtual Networks

Cloud computing technologies offer the necessary infrastructures to rapidly deploy large distributed systems and applications. Together with the recent advances in the way wireless access technologies evolve to provide ubiquitous wireless access, a technological breakthrough is on the go. The potential for collaboration between cloud/virtualization technologies and ubiquitous wireless access networks is enormous and this coupling constitutes a true paradigm shift, over which services can be build. The services we focus in this last part of this thesis are content distribution services, provided over a cloud system that exploits the virtualized multi-domain, multi-provider SDN architecture defined in the first Chapter.

The model of resource sharing in data-center operations, combined with the successful model of Mobile Virtual Network Operators (MVNOs)[16] can be exploited in end-to-end fashion and so extend the business models of the involved stakeholders. An efficient virtualization system could hide all the complexities of the underlying infrastructures and layers and leave the MOVNO focus on the services it provides. For example, a Content Delivery Network (CDN) provider could deploy its network over multi-domain, MOVNO environments [3].

The idea is that a content distribution provider owns content that users from all the virtual operators can access, while the provider can establish different business relationships with a) every physical storage provider regarding the placement cost of content and b) the operators. The problem under consideration in this Chapter, is to weigh the trade-off between speed of content access and increased user QoE, that we translate in higher revenue for the CDN provider. This comes at the cost of content placement in every physical domain, in a way that will lead in profit maximization for the provider. We focus on the analysis of the VNets/CDN scenario, since it is more generic and of high importance to the building of end-to-end virtual networks. The concept of cloud CDN providers [120] has emerged. In addition, inter-domain CDNs, now rely on the interaction between ISPs and CDNs, while they use cache management schemes over converged multi-domain environments [121] and [122].

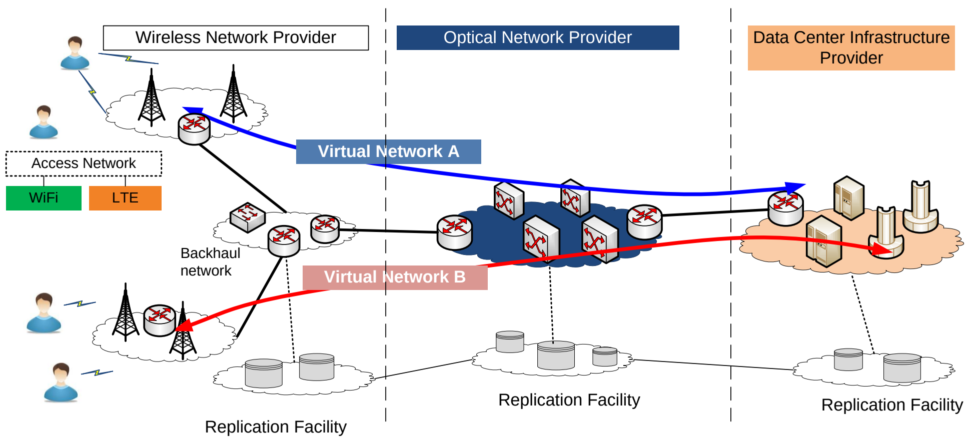


Figure 5.1: System Model

5.1 Introduction

The services we focus in this chapter are CDN services provided over multiple provider networks, virtual or not, where their access networks are deployed in wireless heterogeneous networks (HetNets). We examine how such converged virtual infrastructures, can be used to offer cloud based replication services (CDN-like) and how known replication policies can be exploited by both the physical infrastructure provider and the virtual network operators. Recent works, e.g. [123], [121],[122] and [124], show that a closer collaboration between CDN providers and ISPs, will have a proliferative positive effect on the systems operation and end-user performance. Both can jointly take advantage of the already deployed distributed multi-domain infrastructures and also benefit from the advancements in virtualization technology.

In the model that we consider, every domain of the end-to-end path (wireless, optical, datacenter) is virtualized by means of resource virtualization/isolation and in addition is able to host replication facilities, enabling this way replication actions. The replication facilities are accessed by users which belong in virtual networks (VNets), that use different virtual communication paths with different capacity/cost/network characteristics. Any user is logically associated to a VNet (owned by a Virtual Network Operator), but physically served by a number of wireless access domains. We nurture the concept of different costs per object request per virtual operators and per domain. Our objective is to minimize the end-to-end operation cost of the system and maximize the provider's profit, by exploiting the caching capabilities of the intermediate domains between the users and the datacenter.

The main motivation and questioning behind this work is the following: *if all the various segments/domains of an end-to-end architecture are able to perform replication actions and all the mobile end-users request objects from a common pool, but users belong to different virtual networks (VNets) that have different SLAs with the physical infrastructure provider, which is the best replication policy, so that the utilization of the content provider is maximized?*.

Our contributions are the following. Firstly, we develop a mathematical framework for efficient content placement in multi-domain environments. The model takes into account the probability distribution of users belonging to one access network or another, while the network providers (operators) that the users are associated with, have different business relationships with the physical providers and the CDN providers; these relationships affect the cost of content retrieval. The proposed model also considers the content placement cost in every domain, besides physical storage size limitations. Then, a greedy centralized approach and a distributed content placement/replacement scheme (low overhead and easily implementable) are proposed and evaluated through extensive simulations. Note that besides virtual end-to-end networking, the concept of cloud CDN providers [120] has also emerged.

This Chapter is organized as follows. In Section 5.2, we formulate the problem of content placement in multi-domain environments, whereas in Section 5.3 we present the content placement policies. In Section 5.4 we evaluate through simulations the proposed policies, while we conclude the paper and give pointers for future work in Section 5.5.

5.2 System model and Problem Statement

Let $\mathcal{K} = \{1,2,\cdots,K\}$ denote the set of all the available domains (e.g., optical A, optical B, WiFi, etc) and $\mathcal{L} = \{1,2,\cdots,L\}$ denote the set of all the access domains (e.g., LTE A, LTE B, WiMAX, WiFi A, etc), where $\mathcal{L} \subseteq \mathcal{K}$. Also let $\mathcal{V} = \{1,2,\cdots,V\}$ denote the set of all end-to-end virtual networks. We assume that every mobile user is associated with a single VNet. We also assume that a single CDN provider offers content services with $\mathcal{M} = \{1,2,\cdots,M\}$ objects. In our model, all the objects are accessible by all users belonging in all VNets; however, the utility for an object i enjoyed by a CDN user belonging in VNet j is different across VNets. This is reasonable in our end-to-end virtualized model, since the physical providers and their business relationship with the CDN provider may be also affected by the business relationship between them and the VNet operator. We use $u_{i,j}$ to denote the willingness to pay (in cost units) of a user belonging in VNet j for accessing object i.

In the access domain, a lot of research has been done regarding the optimal access network selection [125] and rate distribution in HetNets [126], while various mobility models exhibit the characteristics of temporal dependency, spatial dependency and geographic constraint mobility [127]. Nevertheless, in this work in order to handle user mobility and also be aligned with the HetNet concept, we are only interested in the steady state probability of a user using a number of access networks. We adopt a simple Markov model for any user, where states represent the access networks that a user/subscriber is physically served from. These steady state probabilities (of using one physical access network or another) are then used, to “split” the user’s total request traffic to the various access networks it enables.

Let $r_{i,j}$ denote the request rate for object i from all the users associated with VNet j. If we let π_j^l denote the steady state probability of any user that belongs to VNet j to be served by access network l, then the request rate distribution is equal to $r_{i,j}^l = \pi_j^l \cdot r_{i,j}$, $\forall l \in \mathcal{L}$. Also, we define $d_i(k,l)$ as the distance between the closest domain $k \in K$ where object i is placed and the access domain $l \in \mathcal{L}$ where the request originates.

We also use the following notation: c_i^k is the cost of placing object i in domain k (c_i^k also includes

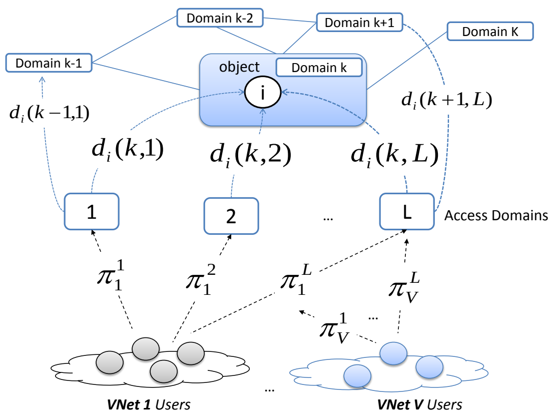


Figure 5.2: System Model

the transfer cost that the CDN provider pays to the physical network provider(s) to transfer object i in domain k), p_i^k is the probability of finding object i in domain k (and is our control variable as we show later), s_i is the size of object i and S^k is the storage capacity of domain $k \in \mathcal{K}$ in bits accordingly. Fig. 5.2 depicts the system model under consideration and Table 5.1 provides a notation summary.

5.2.1 Problem Statement

We define the utility $U_j(l)$ enjoyed by the CDN provider when the objects are accessed by users of VNet j residing at the $l \in \mathcal{L}$ access network as

$$U_j(l) = \sum_{i=1}^{|\mathcal{M}|} u_{i,j} \cdot r_{i,j}^l \cdot (1 - f(D_i(l)) + \Delta_i^*) \tag{5.1}$$

where $D_i(l)$ is the minimum distance between the domain k where object i is stored, and the requester when the requester is served by access network l. We define this distance as

$$D_i(l) = \min_{k:p_i^k=1} d_i(k,l) \tag{5.2}$$

where $d_i(k,l)$ is the distance expressed by hop counts.

In Eq.(5.1) function $f : R_+ \to [0,1]$ is used to normalize the distance values in a range between $[0,1]$. The intuition behind Eq.(5.1) and Eq.(5.2) is that whenever a user retrieves content from an access domain and the objects retrieved are also stored in this domain then $f(D_i(l)) = 0$. In this case, the maximum net benefit (i.e., utility minus cost) occurs for the CDN (content is retrieved as fast as possible and thus user willingness to pay is maximized); thus, CDN provider can make higher profit and increase demand for CDN services. In the case where the requested content can be found only in the data center, then $f(D_i(l)) = 1$. In this case, we use Δ_i^* to describe that a minimum gain (satisfaction) is achieved even when the object is found in the data center.

The probability p_i^k of finding object i in domain k is defined as

$$p_i^k = \begin{cases} 1, & \text{if } k = 1 \text{ (the datacenter)}, \\ \in \{0,1\}, & \text{otherwise.} \end{cases} \tag{5.3}$$

Table 5.1: Notation Summary

Not.	Description
$\mathcal{K}$	set of domains (optical, WiFi etc)
$\mathcal{L}$	set of access domains (e.g LTE,WiFi), $\mathcal{L} \subseteq \mathcal{K}$
$\mathcal{V}$	set of Virtual Network Operators (VNO)
$\mathcal{M}$	set of objects
$u_{i,j}$	willingness to pay of a user in VNet j for object i
$r_{i,j}$	request rate for object i by users associated to VNO j
π_j^l	steady state prob. of a user in VNet j to be served by access network l
$d_i(k,l)$	distance between domain k where object i is placed and access domain l
c_i^k	cost of placing object i in domain k
p_i^k	probability of finding object i in domain k
s_i	size of object i
S^k	storage size of domain k
$U_j(l)$	Utility $U_j(l)$ the CDN provider enjoys from VNet j when the objects are accessed by access network l

With the above definition we assume that domain $k = 1$ is the data center and that all objects are stored in the data center, while is up to the used cache management algorithm to decide in which other domain(s) to replicate the content (object i in this case). Note that $p_i^k = 0$ means that the object i is not available in domain k, while $p_i^k = 1$ means that the object i is available in domain k. These values are used to define the binary matrix $P = [p_i^k]$ of size $K \times M$, that is the control variable to the following optimization problem

$$\underset{P}{\text{maximize}} \sum_{l=1}^{|\mathcal{L}|} \sum_{j=1}^{|\mathcal{V}|} U_j(l) - \sum_{i=1}^{|\mathcal{M}|} \sum_{k=1}^{|\mathcal{K}|} p_i^k \cdot c_i^k \tag{4}$$

$$\text{subject to } \sum_{i=1}^{|\mathcal{M}|} p_i^k \cdot s_i \leq S^k, \forall k \in \mathcal{K} \tag{5.4a}$$

$$\sum_{k=2}^{|\mathcal{K}|} p_i^k \leq |\mathcal{K}| - 1, \forall i \in \mathcal{M} \tag{5.4b}$$

where c_i^k is the cost of placing object i in domain k. Note that we have also included the transfer cost in c_i^k. The physical interpretation of this inclusion is that in order to cache an object closer to the requester and increase its QoE and the CDN's profit, a higher transfer cost based on the business relationships of the CDN provider with the network provider or the ISP would be observed. This inclusion makes the model even more generic, since complex relationships can also be defined. The first restriction Eq.(5.4a) is set in order to meet the capacity constraints in every domain, whereas the second restriction Eq.(5.4b) means that any object i can be stored in up to $|\mathcal{K}| - 1$ domains, besides the data center where it is already stored. Eq.(5.4) provides the *maximum net benefit* of the CDN provider.

5.3 Content Placement Policies

Effective implementation of the vision for inter-domain cloud-based CDNs requires the formulation of a robust methodology regarding the coordination between multiple providers (e.g. access, optical, ISPs) and the integration of multiple cloud and virtualization technologies. Inter-domain CDNs rely on the interaction between ISPs and CDNs and cache management schemes over this converged environment are presented in [121] and [122], while replica-selection decisions coordination is presented in [128] in a distributed environment, focusing in cloud services. Also, today caches use dedicated hardware on a per-CDN provider and per-operator basis [129]. In the new virtualized environment, by applying the NFV paradigm in order to utilize and deploy virtualized caches, the underlying hardware resources could be consolidated and shared among multiple CDN providers, improving resources usage [130].

An autonomic cache management framework for future Internet was presented in [131], while a work on cache management over converged end-to-end virtual networks was presented in [132]. In our modeling, in contrast to [131]-[132] we consider different cost per end-to-end VNet and in addition, we consider user mobility, multiple access domains, while we take into account the placement cost, besides physical storage limitations. Placement algorithms that use workload information, such as distance from the storage points and request rates, to make the placement decision are investigated in [133].

Even at steady state (e.g., static object access rates, fixed geographical distribution of users, fixed storage costs, etc.) optimal placement of the objects at the caches of the various domains resembles the multiple knapsack problem, which is NP-complete. Hence, we provide two approximate solutions to the problem of object placement.

Centralized Approach - Greedy

In [134], a greedy approximation algorithm has been proposed based on a cost metric related to the distance of the object storage points to the end-users and the object request rates. According to [134], the greedy algorithm has a median performance of $(1.1-1.5)\cdot OPT$ and a worst case of $4\cdot OPT$.

We adapt the greedy approximation algorithm of [134] to our problem as follows. The greedy algorithm works in rounds/iterations. At each round, for each object at the data center, the net benefit gain of its replica placement at every feasible domain is calculated, given that the rest of the objects are already cached at each domain. The object whose replication at a domain gives the highest net benefit gain is selected to be replicated at that domain. The process is repeated for all objects until all the available storage capacity of every possible domain is full. This algorithm requires $\min\{M, \sum_{k=2}^{K} S^k/\bar{s}\}$ iterations, where $\bar{s}$ is the mean object size.

The greedy approximation algorithm is a centralized and static approach that should be re-executed for new objects, for objects whose request rates are significantly modified or whenever other significant problem parameters are modified (e.g. cache storage/placement costs).

Decentralized Approach - Holistic

In this approach, we assume a cache manager at each domain (other than the data center) that acts as autonomous agent and takes decisions, so as to host the objects that maximize the overall net benefit. Periodically, the cache manager of a domain may decide to fetch new objects from the data center or remove cached objects according to the following process:

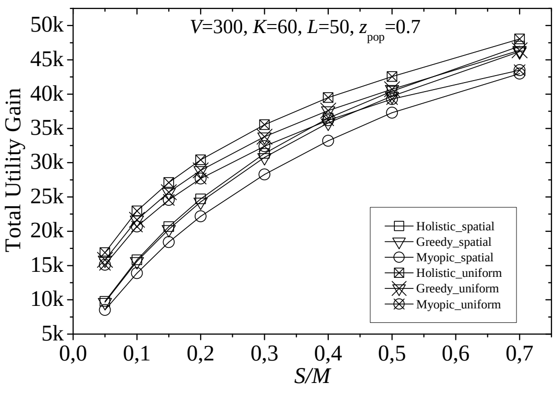
1. Calculate the net benefit decrease arising from the removal of an object stored at the domain. Sort objects in ascending order of the net benefit decrease in list D.
2. Calculate the net benefit increase by caching a new object at the domain. Sort objects in descending order of the net benefit increase in list I.
3. Select object o^* with size s_{o*} at the top of list I, whose insertion results to the maximum net benefit increase.
4. Starting from the beginning of list D, select d objects, so that their total size is greater or equal than the size of object o^*, i.e., $\sum_{i=1}^{d} s_i \geq s_{o*}$.
5. If the total net benefit decrease arising from the removal of d objects is lower than the net benefit increase by the insertion of the new object o^*, then replace the d objects with object o^*.
6. If there is some extra storage space left at the domain by the removal of the d objects, i.e., $\sum_{i=1}^{d} s_i - s_{o*} = e^* > 0$, then go through the list I and fetch every new object that fits and remove its size from the extra space e^*, until no new object fits any more or the extra space is exhausted.
7. Repeat the above steps until no further object replacements can be made among domains and increase the overall net benefit.

Only one domain (manager at each domain) is allowed to perform object replacements at each iteration by means of a distributed consensus algorithm, i.e., Paxos [135], until a steady state is reached where no further object replacements occur.

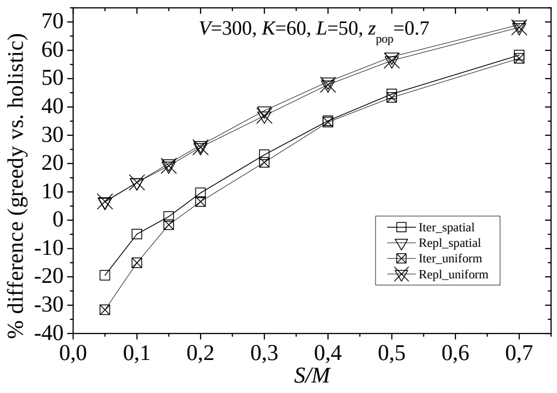
5.4 Performance Evaluation

In this section, we evaluate through simulations the performance of the two cache management algorithms (*Greedy* and *Holistic*) and we compare them against a *Myopic* algorithm. In the *Myopic* algorithm each domain caches objects with the objective to maximize the overall net benefit based on the observed request pattern, without having any knowledge of the caching decisions of the other domains, i.e. an intermediate domain does not know the caching strategy of the access domains that are connected to it.

For the performance evaluation we assume that all domains have the same caching capacity $S^k = S, \forall k \in \mathcal{K}$. We also consider the scenario of $|\mathcal{M}| = 10^6$ different unit sized objects, where the request rate for each object from each VNet $j \in \mathcal{V}$ is determined by its popularity. Here we approximate the popularity of the objects by a Zipf law of exponent z_{pop} (file popularity in the Internet follows Zipf distribution [136]-[137]). The request rate of each object at each VNet varies from $0-20$ reqs/sec according to its popularity and ranking.



(a) Total utility gain

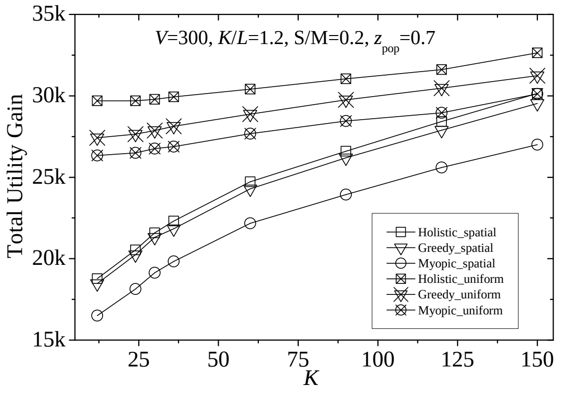


(b) Iterations and object exchanges relative gain

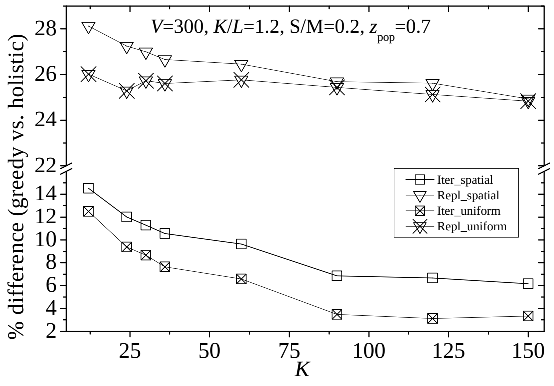
Figure 5.3: The performance of the cache management algorithms vs. the fraction (S/M) of the objects that can be stored in each one of the domains.

For comparison reasons we depict the performance of the proposed algorithms both with the spatial locality workload (noted as *spatial* in the figures), as well as when the VNets follow the same popularity distribution and the same object ranking (noted as *uniform* in the figures). The reason is that the popularity of each object may differ between different virtual networks, a phenomenon that is referred to as locality of interest (*spatial locality* in [138]). In our experiments, the workload is tuned from a localized subscription model, where at each virtual network (VNet) the popularity of the requests follow the same Zipf law distribution of exponent z_{pop}. Nevertheless, the ranking of the objects within this distribution is different among the virtual networks. This means that an object $i \in \mathcal{M}$ that is the most popular object in some VNet might not be the most popular in a different VNet, where another object, may be the most popular. Thus in our evaluation model all objects follow the same z_{pop} popularity distribution in the various VNets, but with different ranking.

In our system model we use a generic mathematical formulation where the request distribution for every VNet depends on the steady state probability of users/subscribers using a specific access domain. Due to page size limitations, we examine the performance of the proposed schemes, in the case where a user is served by a single access domain (so depending on the VNet j, $\pi_j^l = 1$ for a single l and zero elsewhere). More specifically, in order to assign the VNets to the access domains



(a) Total utility gain



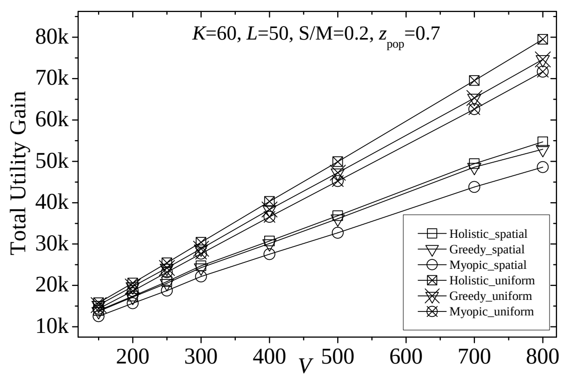
(b) Iterations and object exchanges relative gain

Figure 5.4: The performance of the cache management algorithms vs. the number of domains $\mathcal{K}$.

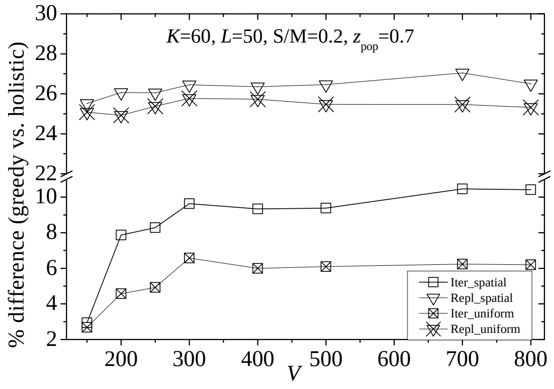
we assume a square area of 300 distance units, where the access domains are uniformly deployed. Each access domain $l \in \mathcal{L}$ covers a circle area of 50 distance units. This means that every VNet that is deployed within this area can be assigned to access domain l. Each VNet j can communicate with a subset of access domains (at least one), and randomly chooses one of them to be assigned. We also assume that each access domain $l \in \mathcal{L}$ can use a random number of other domains to connect with the data center. Finally, we assume that the utility $u_{i,j}$ that a user of VNet j enjoys for retrieving object i from domain k, as well as the the cost c_i^k of caching object i in domain k varies from $0-10$ cost units.

For each proposed algorithm the following performance metrics are used to describe the algorithm's performance: 1) the *Total Utility Gain* at the stationary point and 2) the *percentage difference* of the algorithms regarding
a) the number of iterations $= \frac{(\text{Iter-grd}-\text{Iter-hol})}{\text{Iter-grd}}$
b) the number of replacements $= \frac{(\text{Repl-grd}-\text{Repl-hol})}{\text{Repl-grd}}$.

The number of iterations is indicative of the difference of the algorithms regarding their running time. Regarding the holistic algorithm, the number of object replacements is the number of object fetches (from the data center) that have to be performed once the algorithm has converged, i.e. how many objects have to be replaced in the caches compared to the initial cache assignment, whereas the



(a) Total utility gain

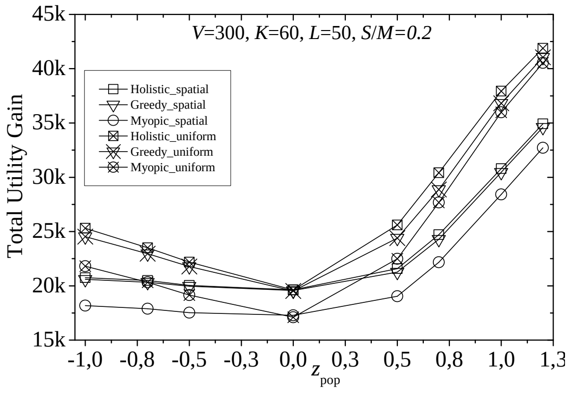


(b) Iterations and object exchanges relative gain

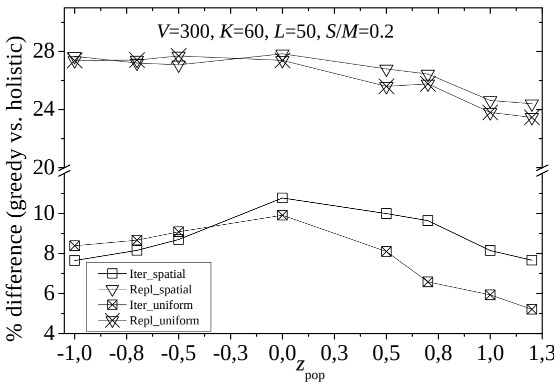
Figure 5.5: The performance of the cache management algorithms vs. the number of virtual networks $\mathcal{V}$.

greedy and the myopic algorithms always start from an empty cache and have to fetch every object from the data center. Since the holistic algorithm assumes an initial cache assignment in the caches of the domains, each point of the following figures is the mean value out of 100 executions starting from different initial cache assignments.

Impact of domain's caching capacity: In Fig. 5.3 we depict the impact of the cache capacity, expressed as the fraction of the objects that can be stored at each domain $k \in \mathcal{K}$, on the performance of the examined algorithms. Regarding the Total Utility Gain we observe that the the holistic algorithm performs $\approx 2\%$ better than the greedy one when the workload with the spatial locality is used, and $\approx 5\%$ better when uniform popularity is assumed. The holistic algorithm performs also better than the myopic algorithm regarding the utility gain $8\% - 15\%$. From Fig. 5.3 we also observe a linear increase regarding their difference in the number of iterations and object fetches. As we relax the storage capacity constraint and allow more objects to fit in the cache of each domain, the holistic algorithm only needs to make small adjustments in the caches to maximize the utility gain, whereas the greedy algorithm every time starts with an empty cache and its complexity is strongly coupled with the size of the caches. In more details, in the holistic algorithm the number of object replacements/fetches per domain decreases almost linearly as the capacity of the caches increase, since the



(a) Total utility gain



(b) Iterations and object exchanges relative gain

Figure 5.6: The performance of the cache management algorithms vs. the popularity exponent z_{pop}.

availability of more cache slots enables more objects to be stored and hence less replacements are required to reach the selected assignment. Only for very small sizes of the caching capacity of each domain the greedy algorithm requires less iterations than the holistic, which implies that terminates faster, but at any case the holistic requires always less object fetches (less traffic in the system).

Impact of the number of domains: In Fig. 5.4 we depict the impact of the number of the domains $\mathcal{K}$, on the performance of the examined algorithms. We assume that the access domains $\mathcal{L}$ are equal to $|\mathcal{L}| = |\mathcal{K}|/1.2$. Regarding the utility gain we observe an almost linear increase as we increase the number of domains, since more domains means that each VNet has more choices of access domains to be assigned to, which further implies that the total load could be distributed more balanced between the access domains. Also, more domains means more storage capacity between the VNets and the data center. As in the previous figure the holistic algorithm requires $\approx 25\%$ less object fetches than the greedy algorithm and $3\% - 15\%$ less iterations regardless the used workload setup (spatial or uniform).

Impact of the number of VNets: In Fig. 5.5 we depict the impact of the number of VNets V in the performance of the examined algorithms. We notice that the total utility gain metric increases linearly with the number of VNets. This means that the algorithms are not affected by the number of the VNets in the system and they manage to accommodate the extra load within the domains, without having to request more objects from the data center. In more details, the holistic algorithm performs $1.5\% - 5\%$ better than the greedy algorithm depending on the used workload setup, and requires almost 10% less iterations and $\approx 28\%$ less object fetches from the data center. This further implies that even if the usage of the holistic algorithm only provides marginal differences in the utility gain compared to the greedy algorithm, its execution converges faster and produces less overhead cost/traffic (object fetches).

Impact of the Zipf's exponents value: In Fig. 5.6 we investigate the performance of the algorithms as the popularity (exponent z_{pop}) of the request pattern at each VNet change. As with the previous figures we observe that the newly proposed holistic algorithm performs slightly better than the greedy, but requires almost 10% less iterations and approximately 25% less object fetches at the stationary point.

5.5 Chapter Conclusions

Cloud services related to content distribution are in the core of today's research, due to the explosion of the mobile usage of Internet and multimedia services. Although it is well known that cloud computing technologies will play a significant role in content delivery, it is less understood how cloud service provisioning will evolve on a global scale in the near future.

In this work we get into the insights of content replication strategies and capture the effects of using them in an converged wireless-optical-datacenter virtual environment. Particularly, we compared two different cache management algorithms with regards to their performance, complexity and convergence time. Our numerical results provide evidence that well known distributed approaches give significant performance benefits and reduce the time to convergence, when compared to centralized off-line policies. Our imminent future plans is to implement the proposed end-to-end cloud based content replication framework, over the facilities (wireless, optical) of the CONTENT solution, as well as to investigate new cache management algorithms that will also take into consideration topological constraints of the intermediate domains. We focused on a single CDN provider (virtual or not) and the content placement problem (over physical or virtual infrastructures), since the CDN content placement and provisioning is important to be understood and optimally controlled. Investigation of scenarios with multiple CDN providers are left for future work.

Chapter 6

Conclusions and Future Work

The high level goal for every virtual network operator is to carry mission-critical and non-mission-critical traffic of its subscribers from the access network up to the data center and visa versa. Our goal in this thesis was to add a δ, in evolving the SDN concepts, especially in the wireless domain of multi-domain architectures and provide scientific solutions to the emerging problem of dynamic provisioning, network management and control in cloud based systems. New mechanisms, while also leverage ways to apply known QoS and service differentiation techniques that are able to differentiate traffic between operators, in the new virtualized environment of operation. We hope that hopefully we reached both these goals.

We truly believe that the SDN paradigm aims in radically rethinking the end-to-end virtualization concepts, by removing the boundaries set when studying per-domain communications. At this stage it has the necessary industry support for unleashing the hidden potential of cloud technologies, in order to provide a holistic network view. Note that while Software Defined Networks (SDN) and Network Function Virtualization (NFV) technologies and methodologies, will be in a constant state of mutation the following years. Towards 5G communications, a multifaceted impact in the way that mobile virtual networks are actually build and operate and the way cloud services are provided is expected.

6.1 Summary of the Contributions

In this thesis we addressed the problem of differentiating services between classes of customers in virtual end-to-end environments, by dissecting the high level challenge into the following research problems:

1. Design of an end-to-end, multi-domain SDN architecture, that spans from the wireless access network up to the virtualized data-center. This end-to-end architecture served as the ground-floor over which the service differentiation problems were investigated.

2. Dynamic Provisioning in cloud based environments. We contributed with the development of dynamic scheduling algorithms used for guaranteed service differentiation and secondly with

modeling of applications, where multiple virtual operators utilize shared physical infrastructures, with the objective to maximize the service provider profit.

6.1.1 Multi-Domain SDN Architectures

Regarding the design of an end-to-end SDN architecture, we assisted in the design of an open architecture that is able to provide multi-domain virtualization, facilitating not only network virtualization but also network management and control. Our work was made in the context of the CONTENT SDN solution, that required to exemplify all the fictions the research community encounters and debunks regarding multi-domain SDN control. The design of this novel SDN architecture, is aligned with the Open Networking Foundation ONF guidelines, while the research work presented in this thesis is related to the wireless access domain of the architecture and its integration with the optical metro domain in both the Control and Data Planes of the architecture.

Cloud technology driven changes are inevitable and we believe that multifaceted analysis is required in order to come up with a framework that is robust, agile and scalable while is able to provide multi-domain virtualization by following the SDN paradigm. Our goal in this thesis was to rely on a convergent multi-domain SDN architecture that is able to provide differentiated services. The proposed solution serves not only as the necessary, but also as the ideal ground-floor to apply new policies that we devised that are able to offer guaranteed service delivery to different virtual traffic flows.

6.1.2 Dynamic Resource Provisioning in cloud based environments

We developed and analyzed new dynamic scheduling algorithms, that are able to satisfy specific differentiation objectives between competing customer classes. Different customer classes utilize network and processing resources (virtual or physical) that span end-to-end from the wireless access up to the data-center. Due to multi-tenancy effects and the presence of time varying workload conditions, the application of stochastic control theory is required in order to rigorously analyze the performance of policies that are able to achieve differentiation objectives. Considering the modeling of applications, where multiple virtual operators utilize shared physical infrastructures, we addressed the emerging content replication problem a content provider needs to consider, when deploying its network over virtual, multi-domain, heterogeneous environments. In our proposed solution, the benefit that the provider enjoys may be different per network operator for the same request, while our model takes into account the replication cost to every domain, as well as the user mobility, besides physical storage limitations.

More specifically, we proposed and analyzed a class Π of negative-drift Dynamic Weighted Round Robin (DWRR) policies that can be used to satisfy specific differentiation objectives. A general mathematical framework is developed that can be applied to a broad family of scheduling problems where differentiation of resources must be provided, like in OS thread/process scheduling or multi-processing systems. We presented a theoretical analysis regarding saturated arrival conditions, using stochastic analysis, while we investigated various properties like speed of convergence. We also extended this mathematical framework to include the stochastic arrivals case and we present proof of convergence for the non-work-conserving mode of operation, feasibility space analysis and dependence on the service redistribution algorithm on final performance, for the general case.

An example SLA that the DWRR classs of policies can be used to satisfy, could be cast along the following lines: "*without any knowledge of statistics regarding the arrival and the service process in a cluster of servers, guarantee* 20% *of CPU power to requests of class A,* 30% *of CPU power to requests of class B and* 50% *of CPU power to requests of class C in the long run*.

The following theoretical results can be collectively deduced from our analysis of negative drift DWRR policies $\pi \in \Pi$, under unknown service time and arrival process statistics. Let $\tilde{p}_i$ denote the maximum utilization service a domain can enjoy, assuming no competition, and p_i the resource share, percentage objective. Then:

1. In the case of saturated arrivals, $\forall \pi \in \Pi$ converges with probability 1 (w.p. 1) to the goal percentage, p_i.
2. In the case of stochastic arrivals, when operating in non-work conserving mode, steady state exists for any policy $\pi \in \Pi$. Any policy converges with probability 1 (w.p 1) to the minimum between the goal percentage and the maximum utilization service.
3. Assuming steady state, the feasibility space for any policy $\pi \in \Pi$, in the case of stochastic arrivals and all modes of operation, can be clearly defined.
4. In the case of stochastic arrivals, under work conserving mode of operation:
   a. any policy $\pi \in \Pi : \tilde{p}_i \leq p_i$ convergences to $\tilde{p}_i$.
   b. for any policy $\pi \in \Pi : \tilde{p}_i > p_i$ convergence point depends on the service redistribution algorithm.

Besides the development and analysis of negative-drift DWRR policies, we also developed new stochastic en-queueing techniques and prediction algorithms that are able to provide differentiating services between competing customer classes. For both the en-queueing and prediction techniques no steady state analysis was provided, nevertheless through extensive simulations we present that are acceptable approximations to the differentiation objectives. These techniques are related with the contributions. In the stochastic en-queueuing approach, we present an approach to provide guaranteed service delivery to each customer class by selecting the queue to store incoming requests. This scheme is evaluated in two application scenarios. In server systems, where guarantees are provided on CPU usage, and in 802.11 Access Points, where the stochastic en-queueing technique is proposed to provide guarantee throughput to individual customer classes. In the predictions based approach, in contrast to existing work where estimation and prediction techniques are used to estimate average values, we adjust scheduling probabilities based on system dynamics.

Regarding the contribution of this thesis related to applications over a convergent multi-domain system, a mathematical framework for efficient content placement in multi-domain environments is proposed. The services we focus are content distribution services provided over a cloud system that exploits the virtualized multi-domain, multi-provider CONTENT architecture. The model takes into account the probability distribution of users belonging to one access network or another, while the network providers that the users are associated with, have different business relationships with the physical providers and the content distribution providers. The proposed model also considers the content placement cost in every domain, besides physical storage size limitations while the objective is to maximize the benefit that the provide enjoys.

6.2 Future Work

The core work presented in this thesis can be extended in many ways, such as optimizing different objectives to serve different QoS metrics and SLAs among the network and the service providers. In distributed computing cloud-based environments, reaching a high level goal is a very complicated procedure, because a very large number of intermediates, must cooperate in order to provide end-to- end solutions. Network conditions and configurations, operating systems used, applications robustness, server used in complicated operations, access networks operations etc. may experience unpredictable failures or performance. In this context, it is very optimistic for an administrator to provide continuous guaranteed support, without the help of vertical or horizontal scalability in the network tier, in the access tier or the processing tier and without using advanced mathematical tools, regarding QoS analysis and service differentiation analysis.

Thus a promising direction for future work, that is applicable to cloud environments, would be along the lines of such end-to-end problems: the control policies in this scenario must take into account interactions between multiple tiers. Furthermore, investigation of more complex SLAs that take into the account the relationship between various performance metrics and the dependencies between services, are left for future work. The same goes for the study of metrics like completion time to capture the effects arised in the case of session based workloads. Further research must be also made in order to meet implementation constraints in highly dynamic environments. Our future plans also include comparison between DWRR and alternative schedulers in hypervisor systems (e.g Xen); implementation of DWRR in web server operations, I/O handling enhancements and performance investigation under preemptive operation.

Our future research includes the investigation on service differentiation and QoS problems in virtual wireless, SDN/NFV based networks. Ongoing work includes scenarios where besides guaranteed service delivery by means of service differentiation, at the same time mission-critical applications receive higher priority and QoS is applied. This is very important consideration especially in the wireless domain.

In the application layer of the envisioned architecture, by means of service delivery, in this thesis we addressed the emerging content replication problem a content provider needs to consider, when virtual network operators utilize both the content services and end-to-end virtualized infrastructures. More complex business relationships between the various players (physical infrastructure providers, content providers and virtual network operators) in more complex network architectures, with multiple content providers, will be investigated in the future. Moreover, content migration cost considerations and thorough investigation of the utility function definition to reflect real market conditions, will also be part of our future research, as well as cases of content delivery failure that can be caused by the convergence of multiple domains.

Bibliography

[1] SDN architecture. *Technical Reference, Open Networking Foundation*, June 2014.

[2] CONTENT Project. `http://content-fp7.eu/`.

[3] K. Katsalis, T. Korakis, G. Landi, G. Bernini, B.R. Rofoee, Shuping Peng, M. Anastasopoulos, A. Tzanakaki, D. Christofi, M. Georgiades, R. Larsen, J. Ferrer Riera, E. Escalona, and J.A. Garcia-Espin. Content project: Considerations towards a cloud-based internetworking paradigm. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7, Nov 2013.

[4] Anna Tzanakaki, Markos P Anastasopoulos, Georgios S Zervas, Bijan Rahimzadeh Rofoee, Reza Nejabati, and Dimitra Simeonidou. Virtualization of heterogeneous wireless-optical network and it infrastructures in support of cloud and mobile cloud services. *Communications Magazine, IEEE*, 51(8), 2013.

[5] K. Katsalis, K. Choumas, T. Korakis, M. Anastasopoulos, A. Tzanakaki, J.F. Riera, and G. Landi. Wireless network virtualization: The content project approach. In *19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), IEEE*, pages 90–94, Dec 2014.

[6] A. Tzanakaki, M.P. Anastasopoulos, S. Peng, B. Rofoee, Y. Yan, D. Simeonidou, G. Landi, G. Bernini, N. Ciulli, J.F. Riera, E. Escalona, J.A. Garcia-Espin, K. Katsalis, and T. Korakis. A converged network architecture for energy efficient mobile cloud computing. In *International Conference on Optical Network Design and Modeling*, pages 120–125, May 2014.

[7] Openstack. `http://www.openstack.org`.

[8] K. Katsalis, G.S. Paschos, L. Tassiulas, and Y. Viniotis. Dynamic cpu scheduling for qos provisioning. 2013.

[9] K. Katsalis, G.S. Paschos, L. Tassiulas, and Y. Viniotis. Cpu provisioning algorithms for service differentiation in cloud-based environments. In *Transactions on Network and Service Management (TNSM), IEEE*, 2014.

[10] K. Katsalis, C. Liaskos, A. Choustis, L. Tassiulas, and Y. Viniotis. Dynamic weighted round robin for service guarantees in server systems. In *Transactions on Computers (TOC), IEEE (under review)*, 2015.

[11] K. Katsalis, G.S. Paschos, L. Tassiulas, and Y. Viniotis. Service differentiation in multitier data centers. In *International Conference on Communications (ICC), IEEE*, pages 2567–2572, 2013.

[12] K. Katsalis, K. Choumas, T. Korakis, and L. Tassiulas. Virtual 802.11 wireless networks with guaranteed throughout sharing. In *IEEE/ISCC,The Twentieth IEEE Symposium on Computers and Communications*, 2015.

[13] K. Katsalis, L. Tassiulas, and Y. Viniotis. Distributed Resource Allocation Mechanism for SOA Service Level Agreements. *NTMS, IFIP/IEEE*, pages 1–6, 2011.

[14] K. Katsalis, V. Sourlas, T. Korakis, and L. Tassiulas. A cloud-based content replication framework over multi-domain environments. In *International Conference on Communications (ICC), IEEE*, pages 2926–2931, June 2014.

[15] K. Katsalis, V. Sourlas, T. Papaioannou, T. Korakis, and L. Tassiulas. Content placement in heterogeneous end-to-end virtual networks. In *The 30th ACM/SIGAPP Symposium On Applied Computing, NET, ACM*, 2015.

[16] Dimitris Varoutas, Dimitris Katsianis, Th Sphicopoulos, Kjell Stordahl, and Ilari Welling. On the economics of 3g mobile virtual network operators (mvnos). *Wireless Personal Communications*, 36(2):129–142, 2006.

[17] Dong-Hee Shin. Mvno services: Policy implications for promoting mvno diffusion. *Telecommunications Policy*, 34(10):616–632, 2010.

[18] Mosharaf Chowdhury, Fady Samuel, and Raouf Boutaba. Polyvine: policy-based virtual network embedding across multiple domains. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, pages 49–56. ACM, 2010.

[19] Jian Chu and Chin-Tau Lea. New architecture and algorithms for fast construction of hose-model vpns. *Networking, IEEE/ACM Transactions on*, 16(3):670–679, June 2008.

[20] M.M. Hasan, H. Amarasinghe, and A Karmouch. Network virtualization: Dealing with multiple infrastructure providers. In *Communications (ICC), 2012 IEEE International Conference on*, pages 5890–5895, June 2012.

[21] NM Mosharaf Kabir Chowdhury and Raouf Boutaba. Network virtualization: state of the art and research challenges. *Communications Magazine, IEEE*, 47(7):20–26, 2009.

[22] D.A. Schupke. Multilayer and multidomain resilience in optical networks. *Proceedings of the IEEE*, 100(5):1140–1148, May 2012.

[23] A. Tzanakaki, M. Anastasopoulos, K. Georgakilas, J. Buysse, M. De Leenheer, C. Develder, Shuping Peng, R. Nejabati, E. Escalona, D. Simeonidou, N. Ciulli, G. Landi, M. Brogle, A. Manfredi, E. Lopez, J.F. Riera, J.A. Garcia-Espin, P. Donadio, G. Parladori, and J. Jimenez. Energy efficiency in integrated it and optical network infrastructures: The geysers approach. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 343–348, April 2011.

[24] A. Pages, J. Perello, S. Spadaro, J.A. Garcia-Espin, J.F. Riera, and S. Figuerola. Optimal allocation of virtual optical networks for the future internet. In *Optical Network Design and Modeling (ONDM), 2012 16th International Conference on*, pages 1–6, April 2012.

[25] Georgios S. Zervas, Joan Triay, Norberto Amaya, Yixuan Qin, Cristina Cervelló-Pastor, and Dimitra Simeonidou. Time shared optical network (tson): a novel metro architecture for flexible multi-granular services. *Opt. Express*, 19(26):B509–B514, Dec 2011.

[26] David Dietrich, Amr Rizk, and Panagiotis Papadimitriou. Autoembed: Automated multi-provider virtual network embedding. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 465–466. ACM, 2013.

[27] Timothy Wood, KK Ramakrishnan, Prashant Shenoy, and Jacobus Van der Merwe. Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines. In *ACM SIGPLAN Notices*, volume 46, pages 121–132. ACM, 2011.

[28] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM*, volume 9, pages 39–50, 2009.

[29] Theophilus Benson, Aditya Akella, Anees Shaikh, and Sambit Sahu. Cloudnaas: a cloud networking platform for enterprise applications. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 8. ACM, 2011.

[30] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards predictable datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 242–253. ACM, 2011.

[31] Sebastian Angel, Hitesh Ballani, Thomas Karagiannis, Greg OShea, and Eno Thereska. End-to-end performance isolation through virtual datacenters. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*, pages 233–248. USENIX Association, 2014.

[32] Katherine Guo, Shruti Sanadhya, and Thomas Woo. ViFi: virtualizing WLAN using commodity hardware. In *Proceedings of the 9th ACM workshop on Mobility in the evolving internet architecture*, pages 25–30. ACM, 2014.

[33] G. Bhanage, D. Vete, I Seskar, and D. Raychaudhuri. Splitap: Leveraging wireless network virtualization for flexible sharing of wlans. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–6, Dec 2010.

[34] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, and Teresa Vazao. Towards programmable enterprise wlans with odin. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 115–120. ACM, 2012.

[35] Contrail Architecture. *White paper, Juniper Networks*, 2013.

[36] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6. ACM, 2014.

[37] Opendaylight sdn controller. `http://opendaylight.org/`.

[38] Enabling End-to-End SDN. *White Paper, Huawei Technologies*, 2012.

[39] Dan Levin, Andreas Wundsam, Brandon Heller, Nikhil Handigol, and Anja Feldmann. Logically centralized?: state distribution trade-offs in software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 1–6. ACM, 2012.

[40] M. Bjorklund (Ed). YANG - a data modeling language for the network configuration protocol (NETCONF). *RFC 6020*, 2010.

[41] Daniel Bovet and Marco Cesati. *Understanding The Linux Kernel*. Oreilly & Associates Inc, 2005.

[42] D. Ongaro, A. Cox, and S. Rixner. Scheduling I/O in virtual machine monitors. *SIGPLAN/SIGOPS Proceedings, ACM*, pages 1–10, 2008.

[43] Erol Gelenbe and Isi Mitrani. *Analysis and Synthesis of Computer Systems: Texts)*. Imperial College Press, London, UK, UK, 2nd edition, 2010.

[44] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[45] V. Elek, G. Karlsson, and R. Ronngren. Admission control based on end-to-end measurements. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 623–630 vol.2, 2000.

[46] Alberto Leon-Garcia and Indra Widjaja. *Communication Networks: Fundamental Concepts and Key Architectures*. McGraw-Hill, Inc., New York, NY, USA, 2 edition, 2003.

[47] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *Trans. on Networking, IEEE/ACM*, 1(3):344–357, 1993.

[48] Yuming Jiang, Qinghe Yin, Yong Liu, and Shengming Jiang. Fundamental calculus on generalized stochastically bounded bursty traffic for communication networks. *Computer Networks*, 53(12):2011–2021, 2009.

[49] Leonard Kleinrock. Queueing systems, volume 1: theory. 1975.

[50] Arnold O Allen. *Probability, statistics, and queueing theory*. Academic Press, 2014.

[51] Panqanamala Ramana Kumar and Pravin Varaiya. *Stochastic systems: Estimation, identification, and adaptive control*. Prentice Hall Englewood Cliffs, NJ, 1986.

[52] K.J. Astrom. Theory and applications of adaptive controla survey. *Automatica*, 19(5):471 – 486, 1983.

[53] Shankar Sastry and Marc Bodson. *Adaptive control: stability, convergence and robustness*. Courier Corporation, 2011.

[54] Douglas J Leith and William E Leithead. Survey of gain-scheduling analysis and design. *International journal of control*, 73(11):1001–1025, 2000.

[55] Yixin Diao, Joseph L Hellerstein, and Sujay Parekh. Optimizing quality of service using fuzzy control. In *Management Technologies for E-Commerce and E-Business Applications*, pages 42–53. Springer, 2002.

[56] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *Trans. on Automatic Control, IEEE*, 37(12):1936–1948, 1992.

[57] Leonidas Georgiadis, Michael J Neely, and Leandros Tassiulas. *Resource allocation and cross layer control in wireless networks*. Now Publishers Inc, 2006.

[58] J. Smith, E. K P Chong, A.A. Maciejewski, and H.J. Siegel. Stochastic-Based Robust Dynamic Resource Allocation in a Heterogeneous Computing System. *Parallel Processing, ICPP* , pages 188–195, 2009.

[59] D. Guo and L.N. Bhuyan. A QoS aware multicore hash scheduler for network applications. *INFOCOM Proceedings, IEEE*, pages 1089–1097, 2011.

[60] T.F. Abdelzaher, J.A. Stankovic, Chenyang Lu, Ronghua Zhang, and Ying Lu. Feedback performance control in software services. *Control Systems, IEEE*, 23(3):74–90, 2003.

[61] J. Wei, Xiaobo Zhou, and Cheng-Zhong Xu. Robust processing rate allocation for proportional slowdown differentiation on internet servers. *Computers, IEEE Transactions on*, 54(8):964–977, 2005.

[62] T. Abdelzaher and K.G. Shin. End-host architecture for qos-adaptive communication. In *Real-Time Technology and Applications Symposium, 1998. Proceedings. Fourth IEEE*, pages 121–130, 1998.

[63] T.F. Abdelzaher and K.G. Shin. Qos provisioning with qcontracts in web and multimedia servers. In *Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE*, pages 44–53, 1999.

[64] A. Mehra, A. Shaikh, T. Abdelzaher, Zhiqun Wang, and K.G. Shin. Realizing services for guaranteed-qos communication on a microkernel operating system. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 460–469, 1998.

[65] A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. *SIGMETRICS, ACM*, 31(1):300–301, 2003.

[66] Chih ping Li and M.J. Neely. Delay and rate-optimal control in a multi-class priority queue with adjustable service rates. In *INFOCOM, 2012 Proceedings IEEE*, pages 2976–2980, 2012.

[67] P. P. Bhattacharya, L. Georgiadis, P. Tsoucas, and I. Viniotis. Adaptive lexicographic optimization in multi-class m/gi/1 queues. *Mathematics of Operations Research*, 18:705–740, 1993.

[68] Siva Theja Maguluri, R Srikant, and Lei Ying. Stochastic models of load balancing and scheduling in cloud computing clusters. In *INFOCOM, 2012 Proceedings IEEE*, pages 702–710. IEEE, 2012.

[69] Chih-Ping Li, G.S. Paschos, L. Tassiulas, and E. Modiano. Dynamic overload balancing in server farms. In *Networking Conference, 2014 IFIP*, pages 1–9, June 2014.

[70] Rahul Urgaonkar, Ulas C Kozat, Ken Igarashi, and Michael J Neely. Dynamic resource allocation and power management in virtualized data centers. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 479–486. IEEE, 2010.

[71] Matthew Gast. *802.11 wireless networks: the definitive guide*. " O'Reilly Media, Inc.", 2005.

[72] Stefania Sesia, Issam Toufik, and Matthew Baker. *LTE: the UMTS long term evolution*. Wiley Online Library, 2009.

[73] Yan Yan, Georgios Zervas, Yixuan Qin, Bijan R Rofoee, and Dimitra Simeonidou. High performance and flexible fpga-based time shared optical network (tson) metro node. *Optics express*, 21(5):5499–5504, 2013.

[74] Jingchu Liu, Tao Zhao, Sheng Zhou, Yu Cheng, and Zhisheng Niu. Concert: a cloud-based architecture for next-generation cellular systems. *Wireless Communications, IEEE*, 21(6):14–22, 2014.

[75] Chunming Qiao and Myungsik Yoo. Optical burst switching (obs)–a new paradigm for an optical internet^{1}. *Journal of high speed networks*, 8(1):69–84, 1999.

[76] NITOS Wireless Testbed. `http://nitlab.inf.uth.gr/NITlab/`.

[77] K. Pechlivanidou, K. Katsalis, I. Igoumenos, D. Katsaros, T. Korakis, and L. Tassiulas. Nitos testbed: A cloud based wireless experimentation facility. In *Teletraffic Congress (ITC), 2014 26th International*, pages 1–6, Sept 2014.

[78] FLEX Project. `www.flex-project.eu/`.

[79] Fed4FIRE Project. `http://www.fed4fire.eu/`.

[80] Sanjoy Paul and Srini Seshan. Virtualization and slicing of wireless networks. *GENI Design Document*, pages 06–17, 2006.

[81] J. Ferrer Riera, E. Escalona, J. Batalle, J.A Garcia-Espin, and S. Figuerola. Management of virtual infrastructures through opennaas. In *Smart Communications in Network Technologies (SaCoNeT), 2013 International Conference on*, volume 02, pages 1–5, June 2013.

[82] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Symposium proceedings on Communications architectures & protocols*, SIGCOMM '89, pages 1–12, New York, NY, USA, 1989. ACM.

[83] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *Networking, IEEE/ACM Transactions on*, 1(4):397–413, 1993.

[84] Srinivas Vegesna. *IP quality of service*. Cisco Press, 2001.

[85] Giuseppe Bianchi and Ilenia Tinnirello. Remarks on ieee 802.11 dcf performance analysis. *IEEE communications letters*, 9(8):765–767, 2005.

[86] Tak Lam, J.J. Ding, and J.-C. Liu. Xml document parsing: Operational and performance characteristics. *Computer*, 41(9):30–37, 2008.

[87] D. Lu, H. Sheng, and P. Dinda. Size-based scheduling policies with inaccurate scheduling information. In *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS)*, pages 31–38, 2004.

[88] VMware vSphere 4: The CPU Scheduler in VMware ESX4, 2009.

[89] Richard McDougall and Jennifer Anderson. Virtualization performance: perspectives and challenges ahead. *SIGOPS, ACM*, 44(4):40–56, December 2010.

[90] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat. Comparison of the three cpu schedulers in xen. *SIGMETRICS Perform. Eval. Rev.*, 35(2):42–51, 2007.

[91] Mike Tanner. *Practical Queueing Analysis*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1995.

[92] Donald L Burkholder. Explorations in martingale theory and its applications. In *École d'Été de Probabilités de Saint-Flour XIX1989*, pages 1–66. Springer, 1991.

[93] G.A Edgar and L. Sucheston. Martingales in limit and amarts. *Proceedings of the American Mathematical Society*, 67(2):315–320, 1977.

[94] Ali Ghodsi, Vyas Sekar, Matei Zaharia, and Ion Stoica. Multi-resource fair queueing for packet processing. *SIGCOMM, ACM*, 42(4):1–12, August 2012.

[95] M. Shreedhar and George Varghese. Efficient fair queueing using deficit round robin. *SIGCOMM Comput. Commun. Rev.*, 25(4):231–242, October 1995.

[96] N. Giroux, R. Liao, and M. Aissaoui. Fair queue servicing using dynamic weights (DWFQ), 2001. US Patent 6,317,416.

[97] Der-Chiang Li, Chihsen Wu, and Fengming M. Chang. Determination of the parameters in the dynamic weighted Round-Robin method for network load balancing. *Computers and Operations Research*, 32(8):2129–2145, 2005.

[98] T. Li, D. Baumberger, and S. Hahn. Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin. *SIGPLAN, ACM*, pages 65–74, 2009.

[99] J. B. Nagle. On packet switches with infinite storage. *Trans. on Communications, IEEE*, 35(4):435–438, 1987.

[100] A. Burns, G. Bernat, and I. Broster. A Probabilistic Framework for Schedulability Analysis. *Embedded Software, Lecture Notes in Computer Science, Springer*, 2855:1–15, 2003.

[101] Diwaker Gupta, Ludmila Cherkasova, Rob Gardner, and Amin Vahdat. Enforcing performance isolation across virtual machines in xen. In Maarten van Steen and Michi Henning, editors, *Middleware 2006*, volume 4290 of *Lecture Notes in Computer Science*, pages 342–362. Springer Berlin Heidelberg, 2006.

[102] A. Sharma, H. Adarkar, and S. Sengupta. Managing qos through prioritization in web services. *Web Information Systems Engineering Workshops, IEEE*, pages 140–148, 2003.

[103] C. Zhang, R.N. Chang, C. Perng, E. So, C. Tang, and T. Tao. Leveraging Service Composition Relationship to Improve CPU Demand Estimation in SOA Environments. *SCC, IEEE*, 1:317–324, 2008.

[104] A. Fischer, J.F. Botero, M. Till Beck, H. de Meer, and X. Hesselbach. Virtual network embedding: A survey. *Communications Surveys Tutorials, IEEE*, 15(4):1888–1906, Fourth 2013.

[105] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann. Vne-ac: Virtual network embedding algorithm based on ant colony metaheuristic. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–6, June 2011.

[106] M. M. N. Aldeer. Performance comparison of packet-level multiplexing algorithms with bursty traffic. *Journal of Engineering Science and Technology*, pages 46–52, 2010.

[107] A. Mucci. Limits for martingale-like sequences. *Pacific J. Math.*, 48(1):197–202, 1973.

[108] M.G. Markakis, E.H. Modiano, and J.N. Tsitsiklis. Max-weight scheduling in networks with heavy-tailed traffic. *INFOCOM Proceedings, IEEE*, pages 2318–2326, 2012.

[109] Naoki Osada. The early history of convergence acceleration methods. *Numerical Algorithms*, 60(2):205–221, 2012.

[110] Dror G Feitelson. Workload modeling for computer systems performance evaluation. 2014.

[111] Jianyong Zhang, Anand Sivasubramaniam, Alma Riska, Qian Wang, and Erik Riedel. An interposed 2-level i/o scheduling framework for performance virtualization. *SIGMETRICS Perform. Eval. Rev.*, 33(1):406–407, June 2005.

[112] P.E. McKenney. Stochastic fairness queueing. In *Proceedings of IEEE INFOCOM*, 1990.

[113] Feng Lu, G.M. Voelker, and A.C. Snoeren. Weighted fair queuing with differential dropping. In *INFOCOM, 2012 Proceedings IEEE*, pages 2981–2985, 2012.

[114] Clayton Shepard, Hang Yu, Narendra Anand, Erran Li, Thomas Marzetta, Richard Yang, and Lin Zhong. Argos: Practical many-antenna base stations. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 53–64. ACM, 2012.

[115] Ghannam Aljabari and Evren Eren. Virtualization of wireless LAN infrastructures. In *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2011 IEEE 6th International Conference on*, volume 2, pages 837–841. IEEE, 2011.

[116] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[117] G. Foschini and J. Salz. A Basic Dynamic Routing Problem and Diffusion. *Communications, IEEE Transactions on*, 26(3):320–327, 1978.

[118] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate Fairness Through Differential Dropping. In *Proceedings of ACM SIGCOMM*, 2003.

[119] R. Wolski, N. Spring, and J. Hayes. Predicting the cpu availability of time-shared unix systems on the computational grid. In *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on*, pages 105–112, 1999.

[120] Onapp cdn. `http://onapp.com/`.

[121] Kideok Cho, Hakyung Jung, Munyoung Lee, Diko Ko, T.T. Kwon, and Yanghee Choi. How can an isp merge with a cdn? *Communications Magazine, IEEE*, 49(10):156–162, 2011.

[122] Wenjie Jiang, Rui Zhang-Shen, Jennifer Rexford, and Mung Chiang. Cooperative content distribution and traffic engineering in an isp network. *SIGMETRICS Perform. Eval. Rev.*, 37(1):239–250, 2009.

[123] M. Claeys, D. Tuncer, J. Famaey, M. Charalambides, S. Latre, G. Pavlou, and F. De Turck. Proactive multi-tenant cache management for virtualized isp networks. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 82–90, Nov 2014.

[124] Benjamin Frank, Ingmar Poese, Yin Lin, Georgios Smaragdakis, Anja Feldmann, Bruce Maggs, Jannis Rake, Steve Uhlig, and Rick Weber. Pushing cdn-isp collaboration to the limit. *ACM SIGCOMM Computer Communication Review*, 43(3):34–44, 2013.

[125] Lusheng Wang and G.-S.G.S. Kuo. Mathematical modeling for network selection in heterogeneous wireless networks; a tutorial. *Communications Surveys Tutorials, IEEE*, 15(1):271–292, 2013.

[126] S. Singh, H.S. Dhillon, and J.G. Andrews. Offloading in heterogeneous networks: Modeling, analysis, and design insights. *Wireless Communications, IEEE Transactions on*, 12(5):2484–2497, May 2013.

[127] Fan Bai and Ahmed Helmy. A survey of mobility models. *Wireless Adhoc Networks. University of Southern California, USA*, 206, 2004.

[128] Patrick Wendell, Joe Wenjie Jiang, Michael J. Freedman, and Jennifer Rexford. Donar: decentralized server selection for cloud services. *SIGCOMM Comput. Commun. Rev.*, 41(4), 2010.

[129] Akamai cdn. `http://www.akamai.com`.

[130] M. Chiosi et al. Network functions virtualisation. an introduction, benefits, enablers, challenges & call for action. *SDN and OpenFlow World Congress, Darmstadt-Germany*, 2012.

[131] V. Sourlas, L. Gkatzikis, P. Flegkas, and L. Tassiulas. Distributed cache management in information-centric networks. *Network and Service Management, IEEE Transactions on*, 10(3):286–299, 2013.

[132] K. Katsalis, V. Sourlas, T. Korakis, and L. Tassiulas. Cloud-based content replication framework over multi-domain environments. In *International Conference on Communications (ICC),IEEE*, 2014.

[133] Lili Qiu, V.N. Padmanabhan, and G.M. Voelker. On the placement of web server replicas. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1587–1596, 2001.

[134] Jussi Kangasharju, James Roberts, and Keith W. Ross. Object replication strategies in content distribution networks. *Comput. Commun.*, 25(4):376–383, 2002.

[135] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16, 1998.

[136] L. Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: evidence and implications. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 126–134, 1999.

[137] Lada A. Adamic and Bernardo A. Huberman. Zipf's Law and the Internet. *Glottometrics*, 3:143–150, 2002.

[138] K.V. Katsaros, G. Xylomenos, and G.C. Polyzos. Globetraff: A traffic workload generator for the performance evaluation of future internet architectures. In *New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on*, pages 1–5, 2012.