Department of Computer Science
Series of Publications A
Report A-2020-9

Enabling Network Flexibility by Decomposing Network Functions

Matteo Pozza

Doctoral dissertation, to be presented for public examination with the permission of the Faculty of Science of the University of Helsinki in Auditorium D101, Physicum building, on October 22nd, 2020 at 12 o'clock noon.

University of Helsinki
Finland

Supervisors
Sasu Tarkoma, University of Helsinki, Finland
Ashwin Rao, University of Helsinki, Finland

Pre-examiners
Stefano Secci, Conservatoire National des Arts et Métiers, France
Mika Ylianttila, University of Oulu, Finland

Opponent
Serge Fdida, Sorbonne Université, France

Custos
Sasu Tarkoma, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/en/
Telephone: +358 2941 911

ISSN 1238-8645
ISBN 978-951-51-6644-9 (paperback)
ISBN 978-951-51-6645-6 (PDF)
Helsinki 2020
Unigrafia

Enabling Network Flexibility by Decomposing Network Functions

Matteo Pozza

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
matteo.pozza@helsinki.fi
https://www.cs.helsinki.fi/u/pozza/

PhD Thesis, Series of Publications A, Report A-2020-9
Helsinki, October 2020, 85+75 pages
ISSN 1238-8645
ISBN 978-951-51-6644-9 (paperback)
ISBN 978-951-51-6645-6 (PDF)

Abstract

Next-generation networks are expected to serve a wide range of use cases, each of which features a set of diverse and stringent requirements. For instance, video streaming and industrial automation are becoming more and more prominent in our society, but while the first use case requires high bandwidth, the second one mandates sub-millisecond latency. To accommodate these requirements, networks must be *flexible*, *i.e.,* they must provide cost-efficient ways of adapting to different requirements. For example, networks must be able to scale with the traffic load to support the bandwidth requirements of the video streaming use case. In response to the need for flexibility, the scientific community has proposed Software Defined Networking (SDN), Network Function Virtualization (NFV), and network slicing. SDN simplifies the management of networks by separating control plane and data plane, while NFV allows scaling the network functions with the traffic load. Network slicing provides the operators with virtual networks which can be tailored to meet the requirements of the use cases.

While these technologies pave the way towards network flexibility, the capability of networks to adapt to different use cases is still limited by several inefficiencies. For example, to improve the scalability of network functions, network operators use dedicated systems which manage the state of network functions by keeping it in a data store. These systems are designed to offer specific features, such as reliability or performance, which determine

the data store adopted and the Application Programming Interface (API) exposed to the network functions. Network operators need to change the data store depending on the features required by the use case served, but this operation involves refactoring the network functions, thus implying significant costs. Furthermore, network operators need to migrate the network functions, for example to minimize bandwidth usage during traffic peaks. Nevertheless, network slices convey the traffic coming from a multitude of sources through a small set of network functions, which are consequently resource-hungry and difficult to migrate, forcing the network operator to overprovision the network. Due to these inefficiencies, adapting the network to different use cases requires a significant increase in both Capital Expenditure (CapEx) and Operational Expenditure (OpEx), thus resulting in a showstopper for network operators.

Addressing these inefficiencies would lower the costs of adapting networks to different use cases, thus improving network flexibility. To this end, we propose to decompose the network functions into fine-grained network functions, each providing only a subset of the functionalities, or processing only a share of the traffic, thus obtaining network functions which are less resource-hungry, easier to migrate, and easier to upgrade. We examine three directions along which we can perform the decomposition. The first direction is leveraging the networking planes, such as control and data planes, for example separating the functionalities for packet processing from the ones for network management. The second direction is leveraging the sources and destinations of the traffic flowing through each network function and creating a dedicated network function for each source-destination pair. The third direction is decoupling the state management of the network functions from the data store by leveraging an API which is independent from the data store adopted. We show that each decomposition addresses a specific inefficiency. For example, decoupling the state management from the data store enables network operators to change the data store adopted without the need for refactoring the network functions.

Decomposing network functions also brings some drawbacks. For example, it can result in an increase of the number of network functions, thus making network management tasks, such as network reconfiguration, more challenging. We study two key drawbacks and we discuss the solutions we designed to contrast them. In this thesis, we show that decomposing network functions allows improving network flexibility, but it must be complemented with techniques to mitigate any negative side effect.

Computing Reviews (2012) Categories and Subject Descriptors:

Networks → Network components → Middle boxes / network appliances
Networks → Network types → Mobile networks
Networks → Network algorithms → Control path algorithms → Network design and planning algorithms
Networks → Network architectures → Programming interfaces

General Terms:
Ph.D. thesis, network function, 5G, mobile network

Additional Key Words and Phrases:
decomposition, decoupling, flexibility, requirements

Acknowledgements

When thinking about my doctoral studies, it is easy to realize that the completion of the path I started four years ago wouldn't have been possible without the help I received from a number of people. First of all, I would like to thank my supervisor, Sasu Tarkoma, and my mentors, Ashwin Rao and Julien Mineraud, who provided me with support and guidance throughout all my doctoral studies. I would also like to thank Hannu Flinck, Diego Lugones, and Patrick Nicholson from Nokia Bell Labs, with whom I had the pleasure to work in a number of projects. My supervisor, my mentors, as well as the colleagues from Nokia, all have in common the trust they decided to put in my capabilities even without knowing me well: I will always be profoundly grateful for the opportunities you have offered me. I would like to extend my thanks to my colleague Seppo Hätönen for the help and suggestions in a countless number of technical dilemmas I faced during the PhD, and to the people working at the Department of Computer Science: the support I received through your work allowed me to enjoy a smooth experience with my doctoral studies. My gratitude goes also to the organizations that have funded my research work, namely the Nokia Center for Advanced Research (NCAR), Business Finland, which has supported my studies through the TAKE-5 and 5G-FORCE projects, and Academy of Finland, which has also supported my studies through grant 319017. Finally, I would like to thank my parents, Armida and Luciano, my siblings, Francesco and Laura, my good friend Alberto, and my girlfriend Lauren, who have relentlessly listened to and supported me, convincing me to keep pushing forward even in the most difficult points of this journey.

Helsinki, October 2020
Matteo Pozza

List of Abbreviations

2G 2^{nd} Generation

3GPP 3^{rd} Generation Partnership Project

5G 5^{th} Generation

AI Artificial Intelligence

ANR Automatic Neighbor Relation

API Application Programming Interface

AtI Active to Idle

CapEx Capital Expenditure

CDF Cumulative Distribution Function

CPU Central Processing Unit

CRUD Create, Read, Update, Delete

CU Centralized Unit

DHT Distributed Hash Table

E2E End-to-End

eMBB enhanced Mobile BroadBand

eNB evolved Node B

EPC Evolved Packet Core

Gbps Gigabit per second

GPRS General Packet Radio Service

GTP GPRS Tunnelling Protocol

HD High Definition

HSS Home Subscriber Server

I/O Input/Output

IA Initial Attach

ILP Integer Linear Programming

IP Internet Protocol

IT Information Technology

ItA Idle to Active

KVS Key-Value Store

LTE Long Term Evolution

MME Mobile Management Entity

mMTC massive Machine-Type Communication

Mpps Million packets per second

NAT Network Address Translation

NF Network Function

NFV Network Function Virtualization

NIB Network-In-a-Box

OpEx Operational Expenditure

P-GW Packet Data Network Gateway

PCRF Policy and Charging Rules Function

PTT Push-To-Talk

QCI Quality Class Identifier

QoS Quality of Service

RAN Radio Access Network

S-GW Serving Gateway

S1H S1 Handover

SDN Software Defined Networking

SeNB Source evolved Node B

SFC Service Function Chain

SLO Service-Level Objective

SON Self-Organizing Network

SPEC Standard Performance Evaluation Corporation

SPR Subscriber Profile Repository

TCP Transmission Control Protocol

TeNB Target evolved Node B

UE User Equipment

UPF User Plane Function

URLLC Ultra-Reliable Low-Latency Communication

VM Virtual Machine

VNF Virtual Network Function

X2H X2 Handover

Contents

Chapter 1

Introduction

Our society is witnessing the proliferation of IT-based services which are changing an increasing number of aspects in our lives, ranging from work to leisure time. An example of these services is home entertainment through on-demand High Definition (HD) video streaming, whose number of users has increased exponentially in the last years [31, 95]. Another example is industrial automation, which can dramatically improve productivity and reduce the rate of human-made errors [65]. All these services are characterized by a different set of requirements which the IT infrastructure providing the services must meet. For example, video streaming requires a significant amount of bandwidth between the device playing the video and the server delivering it, while industrial automation requires accurate synchronization of the actuators, which must experience no delays in their communication [83]. As we can see, the most stringent requirements involve the network connecting the appliances providing the service. Networks are thus expected to be capable of meeting all these requirements, imposing non-trivial challenges in their design and maintenance.

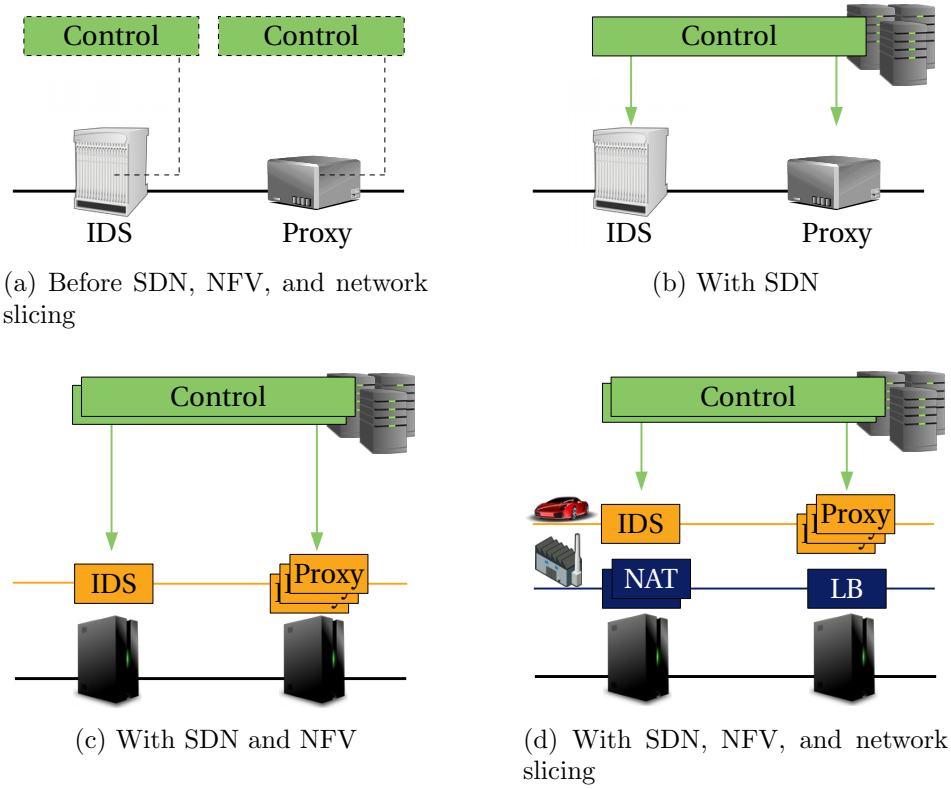
To accommodate this multitude of different requirements, networks must be *flexible*. For example, a network operator should have the possibility to customize the granularity of the network functions, *i.e.,* the set of operations they perform and the kind of traffic they are in charge of. In this way, latency-sensitive operations, *e.g.,* changing the packet header, can be decoupled from the non-critical operations, *e.g.,* reporting statistics, so the first ones are migrated closer to the users, while the second ones are performed in the cloud. More generally, *we consider a network to be flexible if a network operator can tune the granularity, the scale, and the location of the network functions as desired without incurring excessive costs in terms of capital or time.* As an example, a flexible network is capable of adapting to a varying traffic load, *e.g.,* scaling up and down the network

functions, and to meet stringent latency requirements, *e.g.,* migrating the network functions towards the edge of the network, without undergoing time-consuming and expensive upgrade procedures.

In the last years, the scientific community has proposed two main techniques to improve the flexibility of networks, namely Software Defined Networking (SDN) [70] and Network Function Virtualization (NFV) [39]. SDN reduces the granularity of the network functions by separating the data plane from the control plane and proposing to centralize the control plane, thus making networks more easy to configure. NFV improves the ability to scale the network functions in a cost-efficient manner by virtualizing network functions into software-only entities which can be run on general-purpose computing nodes. Lastly, the forthcoming 5^{th} Generation (5G) mobile technology improves network flexibility by leveraging network slicing, which prescribes having isolated groups of network functions for each service. In this way, network operators can tailor the scale and the location of the network functions of each network slice to meet the requirements of the service provided. Network slicing is expected to be the key enabler to meet the stringent requirements of the aforementioned services [36, 47, 76], the reason for which these services are known as the 5G use cases [18]. Figure 1.1 summarizes the benefits brought by each of the technology advancements.

Regardless of their purpose, network functions feature the same underlying architecture in Figure 1.2, which we use as a reference for the rest of the thesis. A Network Function (NF) is typically virtualized, *i.e.,* it runs on general-purpose computing nodes and it is instantiated through a variable number of NF instances, which are virtualized either as traditional Virtual Machines (VMs) or as containers. Henceforth, we use network function and Virtual Network Function (VNF) interchangeably.[1] Each network function consists of the packet processing logic and the state management. The packet processing logic comprises the functionalities provided by the network function, *i.e.,* the operations that are executed when packets flow through the network function. For example, a network function performing Network Address Translation (NAT), simply referred to as NAT, changes the source IP address and port of each packet going from the private network out to the Internet using an assigned (IP address, port) pair. The state management deals with the handling of state information. For example, a NAT stores the (IP address, port) pairs that have been assigned to the packets going from the private network to the Internet so that when a

[1] We use the term *legacy* network functions when we explicitly refer to non-virtualized network functions.



(a) Before SDN, NFV, and network slicing

(b) With SDN

(c) With SDN and NFV

(d) With SDN, NFV, and network slicing

Figure 1.1: Software-Defined Networking (SDN) simplifies the management of the network by separating the data plane from the control plane and centralizing the control plane. Network Function Virtualization (NFV) enables running the network functions on general-purpose computing nodes and scaling the network functions as needed. Network slicing enables tailoring different groups of network functions to meet the requirements of different use cases.

reply arrives it can be forwarded to the requester. The state management stores the state information in a data store, which can correspond to a simple in-memory data structure as well as to a more complex system, *e.g.*, a Distributed Hash Table (DHT).

The adoption of NFV allows i) migrating network functions across the computing nodes in the network, as well as ii) running several network functions in the same computing node, an operation also called *coalescing*, or *co-location*, of the network functions. Note that we consider migrating a network function as migrating all its NF instances, unless we explicitly

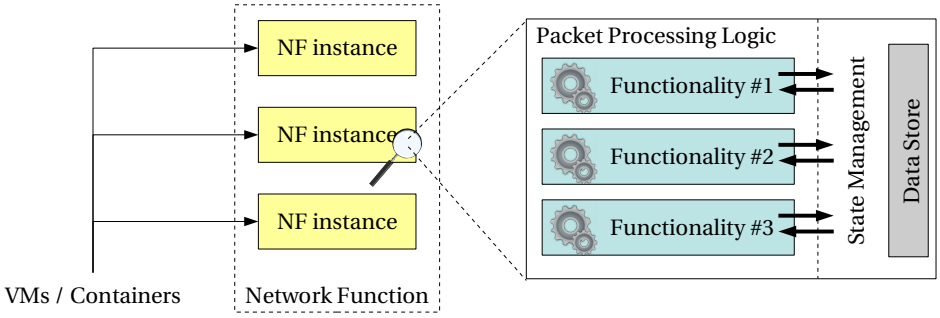


Figure 1.2: A Network Function (NF) comprises a variable number of NF instances, each of which is virtualized, *e.g.*, as a Virtual Machine (VM) or a container. Each network function consists of the packet processing logic and the state management. The packet processing logic comprises the functionalities of the network function, *e.g.*, modifying fields in the packet header in case of Network Address Translation (NAT). The state management handles state information, *e.g.*, packet counters, by storing it in a data store.

mention that we consider a different type of migration, *e.g.*, migrating each NF instance individually. Note also that if i) two network functions send traffic to each other, and ii) the network functions are co-located in the same computing node, then we assume the traffic exchanged between the network functions does not leave the computing node and thus it does not consume bandwidth of the links in the network.

1.1 Motivation

Despite the flexibility provided by current technologies, network operators are still facing major issues in their networks for which they incur high costs. For example, due to the rise in the number of mobile subscriptions and the increasing usage of messaging applications in the last years, Long Term Evolution (LTE) networks incur signaling overheads [35]. Network operators still adopt legacy network functions, which i) provide functionalities of different networking planes, *e.g.*, both control and data planes, and ii) offer no options for customizing the functionalities provided, *e.g.*, separating the functionalities of the control plane from the functionalities of the data plane. As a result, these legacy network functions require specialized platforms and they cannot be easily scaled, hence the signaling overheads, which cause degraded performance and consequently high costs for network

operators [23]. Another example is the limited options that network operators have in organizing the traffic going through a set of network functions. When considering a network slice, the network functions must process the whole traffic flowing between several base stations and several gateways, which makes network functions resource-hungry and difficult to migrate. Network operators have difficulties in migrating these network functions, *e.g.,* moving them towards the edge of the network to meet latency requirements, ultimately leading to resource wastage in the network and revenue loss. Finally, the state management of network functions is currently tightly coupled with the data store adopted for storing the state information. In this context, a network operator cannot upgrade the data store required for a specific use case without refactoring the network functions, which is a time-consuming and error-prone operation [43, 61].

These examples show that current networks are still inflexible because their ability to adapt to different requirements involves high costs. Achieving network flexibility requires addressing the described problems to make the adaptation to different requirements more cost-efficient. Note that, due to the upcoming advent of 5G technology, the ability to meet stringent and distinct requirements is expected from the new generation of mobile networks. For this reason, in this thesis we consider mainly mobile networks and network slicing, but the concepts we illustrate can be applied to any kind of network.

1.2 Problem Statement and Methodology

In this thesis, we focus on three sources of inflexibility, namely signaling overheads, resource wastage, and data store lock-in. In the following, we describe how each of the three sources affects network flexibility.

Signaling overheads. The legacy network functions composing mobile networks incur signaling overheads when the number of active users increases, and scaling these legacy network functions is often expensive [82]. The problem stems from the fact that these legacy network functions provide a multitude of functionalities that belong to different networking planes. For example, the Serving Gateway (S-GW) and the Packet Data Network Gateway (P-GW) of LTE networks provides both control-plane functionalities, *e.g.,* assignment and maintenance of the identifiers of GPRS Tunnelling Protocol (GTP) tunnels, and data-plane functionalities, *e.g.,* enforcement of Quality of Service (QoS) rules and forwarding of the packets [12]. A sudden spike in requests for one functionality requires either to drop requests

when hitting the processing limit, or to scale the entire network function, including under-loaded functionalities [56].

Resource wastage. Each network slice acts as a bridge for the traffic between several ingress points, *i.e.,* base stations [42], and several egress points, *i.e.,* gateways to the Internet [50, 51, 86]. The uplink traffic is conveyed from the ingress points to a limited group of network functions, which process the traffic and send it out towards the egress points, while the downlink traffic follows the same path but in the opposite direction [16]. This organization featuring many ingress nodes, a few network functions, and many egress nodes, results in network functions which require a high amount of resources, *e.g.,* CPU and memory, to process the entire traffic. It is thus challenging to migrate the network functions in the network, *e.g.,* to make room for an additional network slice, to move the network functions towards the edge for meeting latency requirements, or to save bandwidth by coalescing the network functions in the same computing node. This limited mobility of the network functions leads to resource wastage, which ultimately forces the network operator to deny the hosting of additional network slices in the network. In this context, overprovisioning the network is the only way a network operator has to accommodate new requests for hosting network slices.

Data store lock-in. Managing network functions' state, *e.g.,* packet counters, is hard because the state is potentially accessed and modified by many NF instances simultaneously [43]. For this reason, network operators make use of specific state management systems, which take care of tasks such as ensuring the consistency of the state across the NF instances. To store the state of network functions, these systems internally use a data store, which is chosen depending on the requirements the system is designed to meet, *e.g.,* reliability [57] or performance [101]. Moreover, these systems expose an API that network functions use to access and modify the state. The API used by the network functions is highly influenced by the data store used by the state management system, which results in having the network functions tightly coupled with the data store. Nevertheless, network operators need to change the data store adopted when the requirements of the use cases are different, but currently this operation is not possible without refactoring the network functions for the new data store, and thus incurring significant costs.

Making networks more flexible requires developing new solutions to address the problems we described. More specifically, we argue that the way to

master these problems is leveraging fine-grained network functions, which we obtain by decomposing each of the network functions, *i.e.,* distributing its functionalities or its traffic among a number of network functions. The intuition behind is that fine-grained network functions are less resource-demanding [59], easier to orchestrate [92], and easier to upgrade [21], thus providing an additional tool to the network operator for tailoring the network to meet the requirements of the use cases. Before exploring the validity of our intuition, the first step to solve the described problems is performing a literature review to get a complete picture about the state of the art and to make sure a solution has not been found yet. We formulate our first research question.

RQ1. What are the state-of-the-art techniques to improve network flexibility?

To answer the question, we survey both academic and industrial literature, and we examine proposals and prototypes which are explicitly designed to provide flexibility. If the scientific literature already provides cost-effective ways of achieving network flexibility, then we can consider applying these techniques to solve the aforementioned problems. Otherwise, we can conclude that no solution is available yet, and we thus need to design, implement, and evaluate new solutions for the described problems. In this case, we need to make sure that the assumptions and data we use to represent a mobile network and the requirements of the 5G use cases are realistic, leading us to the second research question.

RQ2. How can we faithfully represent a mobile network and the requirements of the 5G use cases?

Given the scarcity of real-world deployment of 5G networks, we collect information from technical documentation, measurements on testbeds, and we confirm our data with experts in the field.

The answers to RQ1 and RQ2 provide the tools needed for the second phase of our research activity, *i.e.,* designing solutions for the aforementioned problems to improve the flexibility of the network. As previously mentioned, we examine ways of decomposing the network functions into fine-grained network functions; more specifically, we examine a different way of performing network function decomposition for each of the problems considered. First, we consider the problem of the signaling overheads affecting LTE networks. We propose to decompose the network functions using the networking planes, thus creating network functions providing a

smaller number of functionalities, all having the same requirements, *e.g.*, CPU-bounded or I/O-bounded.

RQ3. Can we efficiently mitigate signaling overheads by decomposing the network functions using the networking planes?

Decomposing network functions may trigger the need for additional signals because the newly made fine-grained network functions communicate with each other as well, thus apparently worsening the problem. Nevertheless, once network functions are decomposed using the networking planes, we can assess the benefits of coalescing the network functions that communicate the most in the same computing node, thus preventing the signals from flowing on the network links. We thus design a network architecture made of network functions which we decomposed using the networking planes, and we experiment with different ways of instantiating the obtained network functions to reduce the signaling overhead.

To solve the problem of resource wastage in network slicing, we consider the following intuition. If we decompose each network function into network functions which are tailored for a specific pair of ingress and egress nodes, *i.e.*, processing the traffic flowing between the two nodes only, then the demand for resources of each network function is smaller. As a consequence, the obtained network functions are easier to migrate and easier to fit in the computing nodes of the network, offering more options for optimizing the usage of network resources, *e.g.*, coalescing the network functions to save bandwidth.

RQ4. Can we optimize the usage of network resources by decomposing the network functions using sources and destinations of the traffic?

We design and implement an Integer Linear Programming (ILP) model to instantiate network slices in the network while saving as much bandwidth as possible. We then compared the amount of bandwidth saved when adopting normal network slicing and when decomposing the network functions of each network slice instead.

Lastly, we examined the problem of the data store lock-in that network operators incur in the management of the state in network functions.

RQ5. Can we adapt the network functions to different use cases by changing the data store used for managing the state in an efficient manner?

We propose to decouple the state management of the network functions from the data store so that i) the packet processing logic is independent from the data store used for state management, and ii) substituting the data store requires minimal efforts, *e.g.,* changing a configuration file. We design an API that the packet processing logic of a network function can use to access and modify the state. The API is complemented with a series of data store drivers, which take care of translating the API calls into data-store-specific query language, thus keeping the API transparent to the data store actually used for managing the state. The decoupling should also not compromise the ability of network functions to scale. We decompose two network functions using the proposed approach and we test their ability to scale.

Decomposing network functions seems a promising approach to solve the aforementioned problems, but it also brings some intuitive drawbacks that must be addressed. First, decomposing a network function originally designed to work as a single system often implies a loss in performance, which can be unacceptable in certain contexts [64, 101]. For example, when decoupling the state management in network functions from the data store adopted, we need to ensure that the resulting performances are as close as possible to the performance before the decomposition.

RQ6. Can we decouple the state management in network functions from the data store adopted and preserve line-rate performance?

To answer this research question, we re-design the network functions to include performance optimizations which allow approaching line-rate packet processing[2].

Another intuitive drawback of network function decomposition is that it can result in an increase of the number of network functions in the network, which makes more cumbersome certain network management tasks. For example, reconfiguring a network involves migrating the instances of its network functions to obtain a more desirable configuration, *e.g.,* minimizing bandwidth wastage or maximizing spare computing resources. The increase in the number of network functions due to network function decomposition, combined with the stringent requirements of 5G use cases, makes network reconfiguration more cumbersome, thus triggering the following research question.

[2]In this thesis, we consider line-rate packet processing as the packet processing rate recorded by network functions using state-of-the-art state management systems, which generally corresponds to values around 10 Million packets per second (Mpps).

RQ7. How can we reconfigure networks with a high number of network functions?

We design an algorithm that scales with the resources available on the computing platform, and we evaluate its ability to find solutions to the problem when compared with traditional reconfiguration algorithms.

In this thesis, we aim to provide an answer to the seven research questions we formulated. Table 1.1 summarizes the research questions and the methodology adopted to answer them.

1.3 Thesis Contributions

The contributions of this thesis can be summarized as follows:

- We provide a detailed analysis of the state-of-the-art techniques to provide flexibility in networks. More specifically, we focus on examining and classifying Networks-In-a-Box (NIBs), which are networked systems explicitly designed to provide connectivity in a range of different scenarios, such as the aftermath of a natural disaster or overseas flights. We also collect information from a range of trustworthy sources, such as scientific and technical documentation, to provide an input dataset for studying problems in mobile networks. The dataset includes inputs for the substrate network, *e.g.,* computing nodes and links, for the virtual network, *e.g.,* network functions and virtual links, and for the requirements of the use cases, for example in terms of bandwidth and latency.

- We detail techniques to decompose the network functions that can be used to address three key problems stemming from the inflexibility of current networks, namely signaling overheads, resource wastage, and inability to change the data store used for state management of network functions. We provide evidence that the proposed techniques are indeed effective in solving the described problems by comparing the proposed techniques with the state of the art. In particular, we provide i) an ILP model that exemplifies the benefits of decomposing the network functions considering the sources and destinations of traffic, and ii) a system comprising an API and a set of data store drivers that allows decoupling the state management of network functions from the data store used for storing state information.

- We provide a set of tools to mitigate the drawbacks of decomposing network functions, such as drops in performance and managing a po-

Ch.	Research Questions	Methodology	Publ.
Background	**RQ1**: What are the state-of-the-art techniques to improve network flexibility?	Survey scientific and industrial literature.	P2
	RQ2: How can we faithfully represent a mobile network and the requirements of the 5G use cases?	Gather information from scientific documentation, interviews with experts, and measurements on testbeds.	P3, P4
Decomposing Network Functions	**RQ3**: Can we efficiently mitigate signaling overheads by decomposing the network functions using the networking planes?	Analyze the amount of signals saved when adopting decomposed network functions.	P1
	RQ4: Can we optimize the usage of network resources by decomposing the network functions using sources and destinations of the traffic?	Design and implement an ILP model, and evaluate the amount of resources saved.	P3
	RQ5: Can we adapt the network functions to different use cases by changing the data store used for managing the state in an efficient manner?	Design a scalable, data-store-independent system for accessing and modifying the network function state.	M1
Addressing Limitations	**RQ6**: Can we decouple the state management in network functions from the data store adopted and preserve line-rate performance?	Design, implement, and evaluate performance optimizations for the network functions.	M1
	RQ7: How can we reconfigure networks with a high number of network functions?	Design, implement, and evaluate a scalable algorithm for network reconfiguration.	P4

Table 1.1: The research questions addressed in the thesis. For each research question, we indicate the chapter (Ch.) in which it is discussed, the methodology adopted for formulating an answer, and the publication(s) reporting the details of the work.

tentially higher number of network functions in the network. More specifically, we provide i) a range of performance optimizations for network functions to enable line-rate packet processing when the state management is decoupled from the data store adopted, and ii) a scalable algorithm for reconfiguring networks involving a high number of network functions.

The contributions are described in four publications and one manuscript, which is currently under submission. In the following, we report the details of the publications and the manuscript, highlighting the specific contributions of the author as well. The publications and the manuscript are appended to the thesis.

Publication 1 (P1): Matteo Pozza, Ashwin Rao, Armir Bujari, Hannu Flinck, Claudio Enrico Palazzi, and Sasu Tarkoma, "A Refactoring Approach for Optimizing Mobile Networks," In IEEE International Conference on Communications, ICC 2017, Paris, France, May 21-25, 2017, pages 1-6. IEEE, 2017.
Contribution: *The original idea of the work was conceived during a discussion with Ashwin Rao and Sasu Tarkoma. The author developed the idea by studying the original LTE architecture and designing the refactored mobile architecture and the signals required between the modules. The author also performed the analysis and obtained the results reported in the paper. The author and Ashwin Rao wrote the paper, while the other authors provided feedback throughout all the phases of the work.*

Publication 2 (P2): Matteo Pozza, Ashwin Rao, Hannu Flinck, and Sasu Tarkoma, "Network-In-a-Box: A Survey About On-demand Flexible Networks," IEEE Communications Surveys and Tutorials, 20(3):2407-2428, 2018.
Contribution: *All listed authors contributed to the conception of the idea. The author performed the literature review by collecting and analyzing the scientific and technical documentation. The author also wrote the majority of the paper. The other authors provided feedback on the development of the work and on the paper writing.*

Publication 3 (P3): Matteo Pozza, Akanksha Patel, Ashwin Rao, Hannu Flinck, and Sasu Tarkoma, "Composing 5G Network Slices by Co-locating VNFs in μslices," In 2019 IFIP Networking Conference, Networking 2019, Warsaw, Poland, May 20-22, 2019, pages 1-9. IEEE, 2019.

Contribution: *The idea was conceived by Akanksha Patel and Ashwin Rao. The author performed the literature review of the state of the art, while Akanksha Patel designed a preliminary version of the ILP model. The author improved, implemented, and evaluated the ILP model. The author and Ashwin Rao wrote the paper, while the other authors provided feedback in all the phases of the work.*

Publication 4 (P4): Matteo Pozza, Patrick Kevin Nicholson, Diego Lugones, Ashwin Rao, Hannu Flinck, and Sasu Tarkoma, "On Reconfiguring 5G Network Slices," IEEE Journal on Selected Areas in Communications, 38(7):1542-1554, 2020.
Contribution: *The author performed the literature review of the state of the art, while the idea was conceived thanks to the participation of all the authors. The author, Diego Lugones, and Patrick Kevin Nicholson together developed the core of the idea and the algorithm. The author implemented and evaluated the algorithm. The author also wrote the majority of the paper, while all other authors provided feedback in all the phases of the work.*

Manuscript 1 (M1): Matteo Pozza, Ashwin Rao, Diego Lugones, and Sasu Tarkoma, "FlexState: Enabling Innovation in Network Function State Management," Manuscript available at: `https://arxiv.org/abs/2003.10869`
Contribution: *The author performed the literature review of the state of the art, while the idea was conceived thanks to the participation of all the authors. The author and Ashwin Rao developed the idea, integrating the feedback from Diego Lugones in the process. The author created the testbed for the evaluation, and implemented and evaluated the system for network functions and the designed performance optimizations. The author wrote the majority of the paper, while the other authors provided feedback in all the phases of the work.*

1.4 Thesis Structure

The rest of the thesis is organized as follows. Chapter 2 illustrates the results of our literature review about the state-of-the-art techniques for providing network flexibility. The chapter highlights the limitations of these techniques, shedding light on potential avenues for improving network flexibility. The chapter also illustrates the information we gathered to represent

a mobile network and its use cases in our experiments. Chapter 3 develops the core of the thesis, *i.e.*, leveraging network function decomposition to improve flexibility in networks. We illustrate the severe inefficiencies affecting networks and the three directions we explore to decompose network functions. In Chapter 4 we discuss the double-edged sword of network function decomposition: the benefits brought by the decomposition are counterbalanced by drawbacks, which either introduce new problems or worsen pre-existing problems. In this context, we present two key problems stemming from network function decomposition and the solutions we designed to mitigate such problems. Finally, in Chapter 5, we summarize the outcomes of the thesis, its contributions to the state of the art, and we discuss directions for future work on the topic.

Chapter 2

Background

In this chapter, we describe two preliminary studies we have conducted in preparation for the main topic of the thesis, *i.e.*, improving network flexibility by decomposing network functions. First, we performed a literature review to examine the state of the art in terms of techniques to provide network flexibility. After we identified that the state-of-the-art techniques we analyzed were insufficient to meet the requirements of 5G use cases, we gathered information from scientific documentation, interviews with experts, and measurements in our testbed to obtain two datasets to represent realistic mobile networks in our experiments.

2.1 State of the Art in Network Flexibility

We performed a survey that was aimed at finding state-of-the-art networked systems that are designed to provide flexibility. The ultimate goal of the survey is to assess if such systems are enough to meet the stringent and diverse requirements of the 5G use cases. To identify these systems, we first need to identify which the use cases are that are known for triggering a need for flexibility in the network. We identified the following use cases.

After-Disaster Scenario. One of the main use cases for a flexible network is in the aftermath of events such as an earthquakes, tsunamis, or terrorist attacks. These events tend to cause significant damages to the telecommunication infrastructure, and can often result in the network being unusable to the survivors and the rescue teams [8, 55]. Since repairing the pre-existing network typically requires excessive time [72], there is a need for a networked system that complements its working parts, or substitutes it completely. Therefore, the system must be able to selectively

activate, deactivate, and migrate its network functions depending on which functionalities are still working in the original network. In addition, given the significant traffic increase in the hours immediately following the disaster [62], the network functions of the system should be able to scale with the traffic load.

Connectivity Provisioning in Challenging Contexts. Despite the ever increasing proliferation of connectivity across the world, there are specific scenarios in which providing access to the Internet is still challenging. The most indicative example is providing connectivity to villages in developing countries, or villages with particularly harsh climate conditions. Network operators are typically not interested in deploying their infrastructure in such villages because the upfront expense is higher than the expected revenue [78]. Also, providing connectivity in flights or ships is challenging due to the limited space available for the actual network deployment and the difficulties in connecting to the rest of the Internet [32]. In these scenarios, the network should provide the ability to run the most critical network functions where the resources available are very limited, *e.g.,* in the aircraft, and to migrate the non-critical network functions where resources are more plentiful, *e.g.,* in the cloud. Moreover, the network must offer this capability in a cost-efficient manner because the available capital for deploying and maintaining the network is typically limited [38].

Tactical Network. Soldiers on a mission need a network to communicate and organize among themselves. At the same time, they cannot use the public telecommunication infrastructure for various reasons. First, the telecommunication infrastructure is likely to be damaged in the battlefield. Second, even if the infrastructure is properly working, it could be damaged or tapped by the enemy at any time, thus, using it would represent a threat for the soldiers [24]. Therefore, the network required by soldiers i) must be private and ii) it must allow migrating the network functions into a few computing nodes, which in turn may also be capable of moving.

Flash Crowds. Situations such as concerts or popular sport events determine sudden spikes in the traffic load in the network due to the high number of users concentrated in a limited geographical area [69]. In these contexts, the network is not only required to scale adequately, but it must allow the network operator to select the specific network functions to scale, depending on how the new users are utilizing network services.

Both industry and academia have proposed several solutions to meet the requirements of one or more of these use cases. Among the proposed solutions, the one we identified as the most suitable to meet also the requirement of the 5G use cases is the idea of a Network-In-a-Box (NIB). As we show in Subsection 2.1.2, we can indeed leverage several NIBs to obtain a meta-network whose network functions can be scaled and migrated as needed. In the following, we describe the NIB concept, we illustrate its relationship with the flexibility required by 5G use cases in detail, and we discuss its limitations.

2.1.1 Network-In-a-Box

A Network-In-a-Box (NIB) is a portable system that encapsulates an arbitrary set of network functions. The primary goal of a NIB is to substitute, complement, or improve a pre-existing telecommunication network. The network functions that are either missing or insufficient in the original network, *e.g.,* due to the damages of a natural disaster, are replaced or integrated by the network functions hosted in the NIB. For example, the NIB could substitute the Evolved Packet Core (EPC) of an LTE network if the base stations are suddenly detached from the core network [81].

As shown in Figure 2.1, we can abstract three communication channels in a NIB. The first one is the channel used for service provisioning, which targets the end users of the network, *e.g.,* survivors in an after-disaster scenario. The second one is the backhauling channel, which connects the NIB to an external network, *e.g.,* the Internet. The last channel is the interoperability channel, which the NIB uses to connect to network functions of the pre-existing network or to other NIBs. Each channel can be implemented through a variety of technologies, depending on the use cases the NIB is used for. For example, the service provisioning channel can be implemented using the 2^{nd} Generation (2G) technology to maximize the number of users [99, 102], while the backhauling channel can be implemented using a satellite link to minimize the dependencies with any pre-existing network [30, 97]. Note also that the communication channels might or might not be implemented by a NIB. For example, a NIB that substitutes an entire LTE network does not need to implement the interoperability channel.

We identified three key features characterizing NIBs, namely ease of deployment, edge services, and Self-Organizing Network (SON) principles. In the following, we provide a short summary for each of these key feature.

Ease of Deployment. Given that the majority of its use cases feature a tight time budget, a NIB is often evaluated on how easily and how fast it can

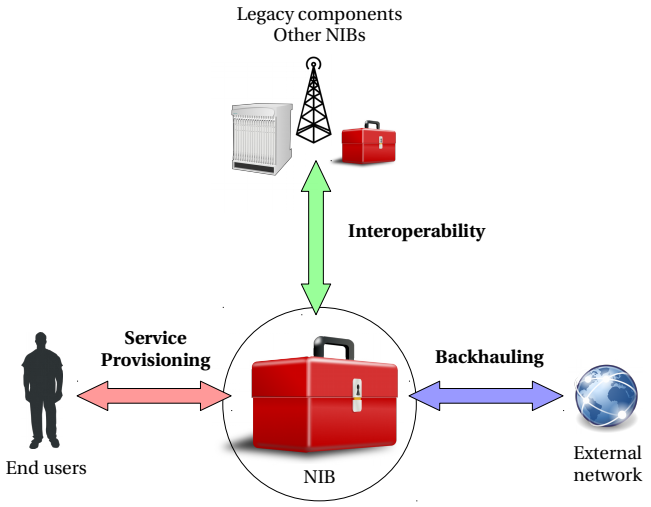


Figure 2.1: A Network-In-a-Box (NIB) has three communication channels: towards the end users (service provisioning), towards the Internet (backhauling), and towards other components of the telecommunication infrastructure or other NIBs (interoperability). Illustration from Publication 2.

be deployed. The deployment involves two phases, i) placing the physical components of the NIB, and ii) configuring them. Factors that play a role in determining the ease of deployment are thus the number of physical elements composing the NIB, their volume and weight, the transport mean to perform their placement, *e.g.*, backpack or aerial platform [25], and the time they require to be configured.

Edge Services. NIBs typically provide to the end users services that are run locally at the NIB, which are called edge services because the NIB operates at the edge of the network. These services include general-purpose services, such as firewalling, public safety services, such as Push-To-Talk (PTT) communication [26], and security services, such as jamming avoidance [24]. There are several reasons for which NIBs provide edge services. The first one is that the services involve the end users communicating only with the NIB, making the service independent from any server in an external network and improving its reliability. Edge services also limit the communication over the backhauling link, which collects the traffic from all the end users and is thus likely to become the bottleneck of the NIB. Finally, the services are typically used only by the end users of the NIB, thus it is preferred to run them locally to the NIB.

Self-Organizing Network (SON) Principles. Manufacturers often design NIBs following the SON principles, according to which the network should require minimal human intervention for tasks such as initial deployment (self-configuration), adaptability to the workload at run-time (self-optimization), and recovery from failures (self-healing). NIBs following the SON principles include services such as Automatic Neighbor Relation (ANR), which allows the NIB to configure autonomously with other base stations nearby [11, 49].

2.1.2 Relationship with Network Flexibility

We have seen how NIBs are designed to meet the flexibility requirements of use cases such as after-disaster scenarios and connectivity provisioning in challenging contexts. At this point, a key question is if NIBs are capable of meeting the flexibility requirements of 5G use cases as well. Indeed, while the NIBs' original use cases all encompass a relatively small geographical area, 5G use cases span much wider areas and the requests for them are much more jeopardized [15]. Moreover, the requirements of 5G use cases are much more stringent [15, 76].

A way to approach the flexibility required by 5G use cases is to organize several NIBs in a meta-network, *i.e.,* a network of NIBs. The NIBs in the meta-network do not operate independently, but they are coordinated through a single management portal. The network operator could deploy the NIBs to cover a wide geographical area while having a unitary view of the network. A network of NIBs offers several advantages to the network operator. First, the network operator can selectively activate and deactivate the network functions inside each NIB, enabling her to freely migrate the network functions and organize the virtual topology of the network as needed. Second, the network operator can scale the network by activating the network function(s) under pressure in many of the NIBs composing the network. Finally, the network operator can configure a NIB to activate a network function in case the NIB currently running the network function suddenly fails. This last example is illustrated in Figure 2.2.

2.1.3 Limitations

Despite the capabilities of NIBs and the advantages a network of NIBs could bring, we conclude that the state of the art of NIBs is not able to provide the network flexibility required by 5G use cases. There are two major facts that corroborate our conclusion. First, none of the examined NIBs allow the network operator to play with the granularity of the network functions

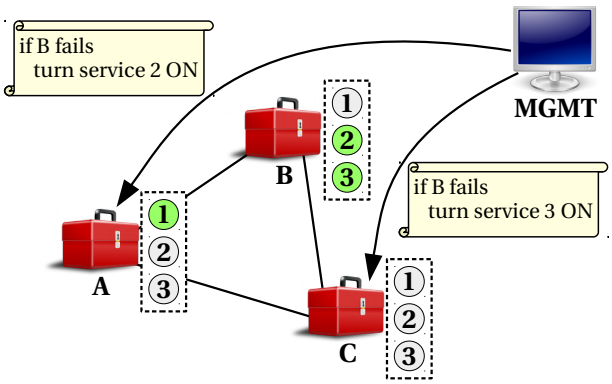


Figure 2.2: A fallback plan in a network of NIBs. Through the management portal, the network operator instructs NIBs A and C to switch ON services 2 and 3, respectively, in case NIB B suddenly fails. Illustration from Publication 2.

because NIBs only allow activating or deactivating network functions as a whole. There is no way for the network operator to isolate a group of functionalities of a network function and make it an independent network function, or to decouple the state management of the network functions from the data store adopted for storing the state information. One of the consequences is that, for example, the state of the network functions is managed internally by each NIB and it cannot be externalized in a dedicated data store, which can be unacceptable for use cases with high availability requirements [45]. Second, the capability of the network to scale is limited by the number and the hardware capabilities of the NIBs in the network. 5G use cases require cloud capabilities to have the network functions to scale adequately, *e.g.*, launching hundreds of NF instances when experiencing a load peak. Instead, if the scalability of the network is tied to the number of NIBs available, then the network operator must overprovision the network with a number of NIBs dimensioned for the worst case, thus incurring high costs.

In summary, NIBs certainly represent a promising step towards the achievement of network flexibility, but they are insufficient to meet the requirements of 5G use cases. Before studying new techniques to improve network flexibility, we need to make sure that our assumptions and data on the network and on the requirements are realistic. In the following, we describe the information we collected to model mobile networks and 5G use cases in our experiments.

2.2 Modeling Mobile Networks

Research in mobile networks has been chronically suffering from a lack of datasets and traces [68, 96]. There are various reasons for the reluctance of network equipment manufacturers and network operators to share their data with the scientific community, such as the risks of backfiring in the future, *e.g.,* advantaging a competitor or involuntarily revealing the identity of customers with the development of new de-anonymization techniques. As a consequence, researchers leverage models to represent mobile networks and their traffic, but creating a realistic model is challenging when the information about the network to represent is scarce. This is the case with 5G networks, for which i) there is only a small number of deployments with limited capabilities [19], and ii) information is scattered among a range of technical documents and scientific papers. For this reason, we designed two mobile network models by gathering the information we obtained by examining technical and scientific documentation, interviewing experts, and performing measurements on mobile network testbeds.

In this thesis, a model consists of two components, i) a representation of the substrate network, *i.e.,* the computing nodes and the links connecting them, and ii) a representation of the virtual network, *i.e.,* the network functions and the virtual links in between, which are dimensioned for the use cases the network is serving [40]. We abstract both the substrate network and the virtual network as graphs, but while the substrate network is a connected graph, the virtual network is a collection of disjoint graphs because each disjoint graph represents a different network slice with dedicated network functions and virtual links. The computing nodes in the substrate network have two attributes, i) the processing capacity, and ii) the memory capacity. Network functions in the virtual network have the same attributes, but they correspond to demands rather than capacities. A model can have a more abstract representation for the network functions, *i.e.,* without specifying the number of instances and with a single value of the attributes for the entire network function (Subsection 2.2.1), or a more detailed representation, *i.e.,* specifying the number of instances and labeling each instance with the attributes (Subsection 2.2.2). The links in the substrate network have two attributes as well, i) bandwidth capacity, and ii) latency incurred. In the virtual network, the network functions communicated among each other requiring a certain amount of bandwidth and requiring the communication to incur a delay smaller than a given budget. Note that the presented models provide a degree of freedom to the researcher to tailor the network representation for the objectives of the evaluation. For example, we can increase the capacity of computing nodes

and links to account for the hardware improvements in the years to come (see Publication 3), or we can customize the coverage of each use case, *i.e.*, which computing nodes receive requests for which use case (see Publication 4).

To obtain a meaningful representation of next-generation networks, we experimented with two different techniques. In the following, we describe the details of the two models of mobile network we designed leveraging the two techniques. In the first model, called LTE-based, we reinterpret the data of pre-existing LTE networks in the context of 5G. For example, we model network slices by multiplexing several virtualized LTE networks on the same substrate network. Mobile operators can leverage this model to understand how to make use of the pre-existing LTE infrastructure to offer 5G-like services. In the second model, called 5G-based, we use data in 5G specifications and measurements collected from 5G trials. New network operators can leverage this clean-slate model to design their 5G networks from scratch.

2.2.1 LTE-based Model

Substrate Network

To get the topology of real networks, we examined SDNlib [1] and Internet Topology Zoo [2], and we selected the New York topology from the first one and the Beijing and Tokyo topologies from the second one. We removed duplicate links and discarded disjoint nodes to obtain connected graphs. To model the processing capacity, we assume each computing node corresponds to a small-scale data center with a certain number of servers. We attribute to each server the highest processing capacity reported in the official results of the Standard Performance Evaluation Corporation (SPEC) benchmark suite of the fourth quarter of 2017 [4]. We assigned 100 servers for each computing node in the substrate network, which is in line with the number of servers in micro-data centers [103]. Then, the processing capacity of each computing node corresponds to the sum of the capacity of the servers assigned to it. Please note that we do not consider the memory of the computing nodes in this model. We consider the links in the substrate network to be point-to-point fiber links having 10 Gigabits per second (Gbps) of bandwidth and a latency of 1 ms [54].

Virtual Network

We consider two network slices in the virtual network, corresponding to two different use cases. The use cases correspond to two different quality classes

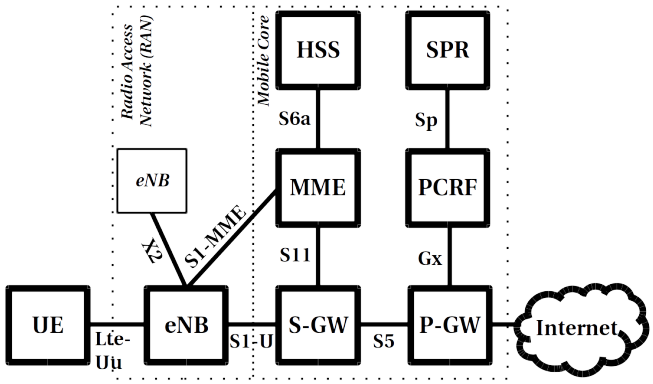


Figure 2.3: The network functions in an LTE network. In the LTE-based Model, a network slice is instantiated as a dedicated set of LTE network functions. Illustration from Publication 1.

as defined in LTE specifications [13], namely streaming of conversational video, with Quality Class Identifier (QCI) 2, and buffered streaming of video, with QCI 8, which we chose because the use cases correspond to the biggest share of traffic on the Internet [90]. In line with similar studies [88], we represent a network slice of the virtual network as a dedicated LTE network. Figure 2.3 illustrates the network functions and the virtual links composing each slice.

To determine the processing, bandwidth, and latency requirements of each network slice, we abstract the traffic incurred by each network slice as a set of control-plane and data-plane procedures. Each procedure involves a sequence of signals exchanged among a subset of the network functions. The control-plane procedures we consider are Initial Attach (IA), *i.e.,* when a User Equipment (UE) first attaches to the network; Active to Idle (AtI) transition, *i.e.,* when the UE goes in idle mode to save energy; Idle to Active (ItA) transition, *i.e.,* when the UE wakes up from its idle state; X2 handover, *i.e.,* when the UE relocates from a base station to another one. We examined the 3^{rd} Generation Partnership Project (3GPP) specifications [12] to identify the signals exchanged by the network functions when the procedures are triggered. The data-plane procedures we consider are data upload and data download, which are both represented as two consecutive signals, from evolved Node B (eNB) to S-GW to P-GW and from P-GW to S-GW to eNB, respectively.

To determine the bandwidth required by each of such procedures, we conducted measurements on the test network deployed on our campus [5].

Name	Direction	Size (B)
UE Context Release Request	eNB → MME	90
Release Access Bearers Request	MME → S-GW	54
Release Access Bearers Response	S-GW → MME	60
UE Context Release Command	MME → eNB	102
UE Context Release Complete	eNB → MME	98

Table 2.1: Size of signals exchanged by the network functions during an Active-to-Idle (AtI) transition. Table from Publication 3.

We measured the size in bytes of the signals exchanged by the network functions during the procedures considered. For example, Table 2.1 shows the size of the signals during an AtI transition.[1] For each use case, we associate a rate at which each of the procedures is requested. Using the request rate, we determine i) the number of signals each network function processes, and thus its computing demand, and ii) the amount of bandwidth required between any two network functions. We consider the rates at which control-plane procedures are requested from Metsälä *et al.* [71] for the first use case, and the rates from Tabbane *et al.* [94] for the second use case. We consider the rates of data-plane procedures reported in a white paper from Huawei [48]. For each use case, the procedures are requested with the specified rates at all the computing nodes labeled as ingress nodes (see Section 3.2).

Finally, we formulate the latency requirements of each use case by assigning a maximum delay budget for the completion of each of the procedures. For the control-plane procedures, we consider the delay budgets reported by Savic *et al.* [91]. We obtained the delay budgets for the data-plane procedures using the 3GPP specifications [13].

2.2.2 5G-based Model

Substrate Network

The topology of the substrate network consists of three hierarchical levels [22, 27, 104], as shown in Figure 2.4. The outermost level is the pre-aggregation level, in which computing nodes are organized in a ring. The computing nodes composing the pre-aggregation ring are called Centralized Units (CUs) and they correspond to the nodes which base stations are directly attached to [14]. The middle level is the aggregation level, which

[1] The measurements of the signals for the other procedures can be found in Appendix A.

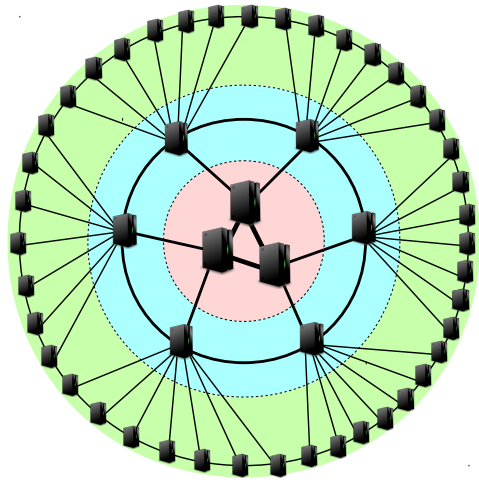


Figure 2.4: In the 5G-based Model, the substrate network comprises three hierarchical levels, namely pre-aggregation ring (outer, green), aggregation ring (middle, cyan), and core mesh (inner, red). Illustration from Publication 4.

organizes its nodes in a ring as well. Finally, we have the innermost level, the core level, in which nodes are organized in a mesh. The hierarchy is so that each node in the core is connected to up to 2 nodes in the aggregation ring, and a node in the aggregation ring is connected to at most 7 nodes in the pre-aggregation ring [17]. The number of nodes in the substrate network depends on the coverage area of the network, taking into account that each CU handles the traffic of a 4 km^2 area.[2]

We followed an approach similar to the one adopted for the LTE-based model to attribute the CPU and memory capacity to the computing nodes. As in the LTE-based model, each computing node is equipped with a certain number of servers, and each server is equipped with 28 CPU cores and 64 GB of memory [80]. We then vary the number of servers clustered at each computing node, taking into account that resources are more scarce towards the pre-aggregation ring and more plentiful towards the core [100]. Nodes are thus equipped with 4, 32, and 64 servers in pre-aggregation, aggregation, and core levels, respectively.

In modeling the links between the computing nodes, we leveraged forecasts on 5G networks released by network equipment companies and institutions [29, 54, 74, 79]. The links connecting the computing nodes comprise

[2]The details on how we compute the coverage area of a CU can be found in Publication 4.

a varying number of 100 Gbps optic fiber connections. Similarly to the computational capacity, the bandwidth available is higher towards the core mesh; pre-aggregation ring, aggregation ring, and core mesh have links with 200 Gbps, 400 Gbps, and 800 Gbps, respectively. Moreover, we attribute 200 Gbps to the links between pre-aggregation ring and aggregation ring and 400 Gbps to the links between aggregation ring and core mesh.

To model the latency of the links in the substrate network, we leveraged two facts, i) the latency between pre-aggregation ring and core mesh is approximately 10 ms [74], and ii) the geographical distance between nodes in the pre-aggregation ring and nodes in the aggregation ring is around 1/4 of the distance between nodes in the aggregation ring and nodes in the core mesh [52]. We attribute a latency of 2 ms to the links between the two rings, and a latency of 8 ms to the links between aggregation ring and core mesh. We also attribute a latency of 2 ms to all links connecting nodes in the same level.

Virtual Network

We consider three different "macro" use cases, namely enhanced Mobile BroadBand (eMBB), Ultra-Reliable Low-Latency Communication (URLLC), and massive Machine-Type Communication (mMTC), and we associate a dedicated network slice to each of them. We focus on these three use cases because the requirements of any other use case can be obtained combining the requirements of the macro-use cases [53]. In the 5G-based Model we consider each network slice consisting of a Service Function Chain (SFC), *i.e.,* a set of VNFs traversed in sequence. Each VNF is instantiated through a varying number of VMs, as illustrated in Figure 2.5, and each VM requires 4 CPU cores and 4 GB of memory [41].

Table 2.2 describes the bandwidth and latency requirements of the three use cases considered. We have collected the requirements analyzing 5G specifications and white papers [15, 76]. Each use case demands the same amount of bandwidth at each link of its SFC. Note that the bandwidth requirement is nevertheless split evenly among the links between the VMs composing the VNFs at each hop of the SFC. For example, in Figure 2.5 the VMs composing VNF 1 and VNF 2 have four links in between, and thus each link demands 25% of the total bandwidth required between the two VNFs. Similarly to the LTE-based Model, the traffic of a use case must be able to traverse the entire SFC within the End-to-End (E2E) latency budget of the use case.

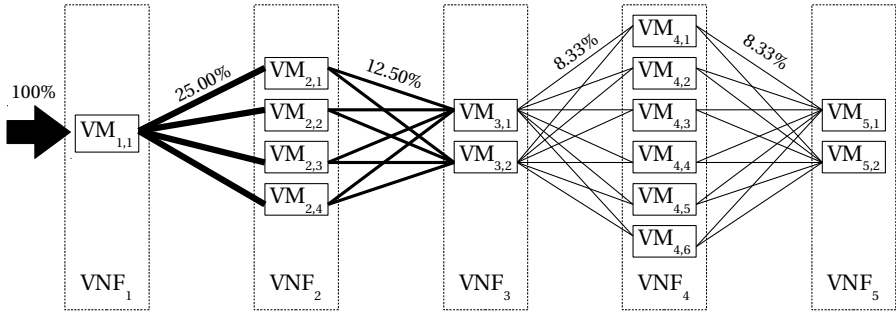


Figure 2.5: In the 5G-based Model, a network slice is implemented as a sequence of Virtual Network Functions (VNFs), each of which is instantiated through a varying number of Virtual Machines (VMs). Illustration from Publication 4.

Use cases	**Bandwidth / km^2**		**E2E Latency**
	Download	Upload	
eMBB	750 Gbps	125 Gbps	10 ms
URLLC	10 Gbps	10 Gbps	1 ms
mMTC	100 Gbps	100 Gbps	50 ms

Table 2.2: Bandwidth and latency requirements of enhanced Mobile Broad-Band (eMBB), Ultra-Reliable Low Latency Communication (URLLC), and massive Machine-Type Communication (mMTC) [15, 76]. Table from Publication 4.

2.3 Summary

In this chapter, we have explored the concept of NIB, *i.e.*, the state-of-the-art technique that is most suitable to provide network flexibility to the best of our knowledge, which we confirmed not being sufficient to meet the requirements of 5G use cases. We have also presented two models for representing mobile networks that integrate the information available on next-generation mobile networks, which are scattered among a range of sources. We hope this contribution will help research in mobile networks and allow researchers to avoid common pitfalls in formulating subsequent mobile network models. In the following chapter, we make use of the models we presented to validate our proposals of network function decomposition.

Chapter 3

Decomposing Network Functions

In this chapter we illustrate our proposals for decomposing network functions. The techniques illustrated in this section address the inefficiencies which are limiting network operators in adapting their networks for different use cases. By adopting these techniques, network functions can be split into a set of more fine-grained network functions, which provide the network operators with more options for tailoring the network as needed.

3.1 Leveraging Networking Planes

We start with examining LTE networks, which are widespread and used by billions of users every day [37]. Figure 2.3 in the previous chapter provides a schematic representation of an LTE network. It features two sub-networks, i) the Radio Access Network (RAN), which consists of the base stations, also called eNBs, and ii) the core network, which comprises the network functions taking care of duties such as user authentication and connecting the users to the Internet. A more complete description of the functionalities provided by each network function can be found in Publication 1. From a high-level perspective, an LTE network takes care of several tasks, *e.g.,* it forwards the data packets back and forth between the users and the Internet. LTE networks also take care of storing subscriber-specific information, such as authentication parameters. The network functions communicate with each other and concur in carrying out these tasks in a holistic manner.

LTE networks are considered extremely inflexible and thus inadequate for the 5G use cases [75, 85]. LTE networks typically consist of legacy network functions that are sold as black boxes to the network operator, who has no possibility to customize nor to upgrade them [20]. Moreover, these

legacy network functions mix functionalities which belong to different networking planes [66,89]. For example, a S-GW takes care of forwarding data packets, but it also provides control plane functionalities, such as the generation of identifiers for the GTP tunnels. Another example is the Home Subscriber Server (HSS), which stores subscriber information but also generates the authentication parameters required during the security procedures. This makes the scaling of the network functions more troublesome because the network operator cannot scale the overloaded functionalities only, but she must rather scale the entire network function, thus wasting resources [56]. Finally, these legacy network functions are typically implemented as single-function devices rather than being virtualized [20]. This not only hinders scalability, but it also makes the migration of the network functions very cumbersome.

The described problems manifest through signaling overhead issues that LTE network regularly experience, commonly referred to as signaling storms [77]. The ever-increasing number of mobile subscribers [37], as well as the behavior of widespread mobile applications, which are characterized by frequent "heartbeat" messages [46], implies a significant increase in the number of signals that the network functions of LTE networks must handle. Nevertheless, the network functions are not able to adapt to these sudden spikes of load, which cause congestion in the network and downtime. To cope with these events, network operators need a solution to reduce the number of signals exchanged by the network functions.

3.1.1 The Approach

By examining LTE networks, we identified three main networking planes, *i.e.,* high-level tasks conducted by the mobile network. Two of them are provided by SDN, which are i) sending the data packets (*forwarding plane*) and ii) managing the data flows between the users and the Internet (*control plane*). Nevertheless, a mobile network also takes care of storing and providing subscriber-specific information, such as authentication parameters and QoS profiles. We thus identified an additional networking plane, *i.e.,* the management of subscribers' information, which we call *storage plane.* Then, we can decompose the network functions by using these networking planes, *i.e.,* grouping the functionalities concurring to different planes into different network functions.[1] Figure 3.1 shows the result of this process on an LTE network. For example, we decompose the network functions eNB, S-GW, and P-GW into two network functions, a forwarding NF and a con-

[1] In this thesis, we are not considering the management plane because it corresponds to a meta-plane which network operators use to administer the network functions [28].

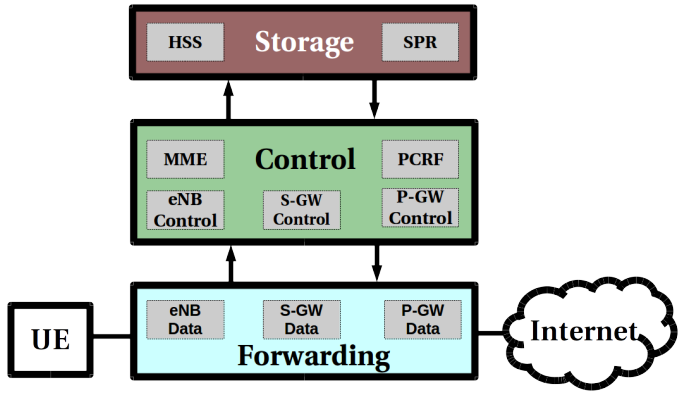


Figure 3.1: Abstracting a mobile network into three networking planes, namely, sending data packets (forwarding), managing the data flows (control), and storing subscribers' information (storage). Illustration from Publication 1.

trol NF. The forwarding NF takes care of forwarding packets and enforcing the appropriate QoS rules, while the control NF provides functionalities related with the management of the data flows, such as generating the identifiers for GTP tunnels and radio bearers.

The key advantages of decomposing the network functions using the networking planes of a mobile network are as follows. First, each of the obtained network functions has well-defined requirements. A control NF requires only a computing platform, while a storage NF only needs a database platform. Similarly, a forwarding NF only requires a platform to send packets and enforce QoS rules. In this way, it is much easier for a network operator to scale the network functions and to provision the network without resource wastage. For example, if control NFs are under a heavy load, the network operator only needs additional computational resources, *e.g.,* leveraging a cloud facility. Second, the obtained network functions are less resource-demanding, and thus it is easier to instantiate them in the network. Finally, by using the decomposed network functions, the network operator can experiment with different ways of coalescing the network functions, for example to reduce the amount of signals transmitted on the links of the substrate network.

Figure 3.2 shows a few examples of how the network operator can refactor the mobile network, *i.e.,* organize and coalesce the network functions to achieve different objectives. For example, compared to the original LTE architecture in Figure 3.2(a), the Split RAN & Core refactoring in Fig-

ure 3.2(b) attempts to make the RAN independent from the core by instantiating a dedicated control NF in the RAN. Instead, the Thin Edge refactoring in Figure 3.2(c) aims to make the RAN a set of remotely-controlled data packet forwarders, thus coalescing all control NFs in the core network. Finally, the Intelligent Edge refactoring in Figure 3.2(d) switches the position of the control NFs to the RAN while keeping the storage NFs in the core.

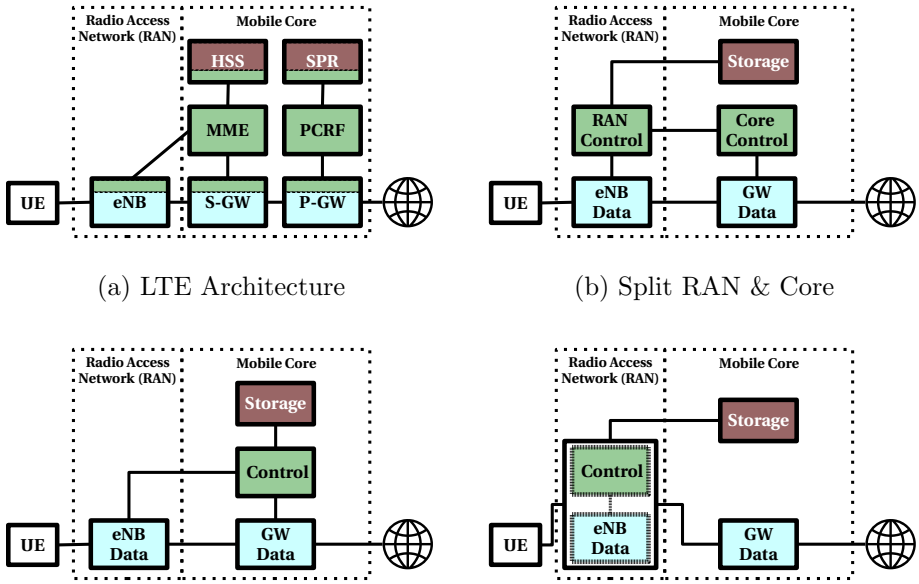
3.1.2 Addressing Signaling Overheads

We consider the aforementioned signaling overhead issues, and we aim to evaluate which of the three approaches is more effective in mitigating signaling overheads. Note that when two network functions are coalesced in the same computing node, the signals they exchange do not leave the computing node where they are running, thus minimizing the impact on the network links. We consider the signals exchanged during the procedures which are triggered most commonly in LTE networks, namely Initial Attach (IA), Active to Idle (AtI) transition, Idle to Active (ItA) transition (both in UE-triggered and network-triggered variants), X2 Handover (X2H), and S1 Handover (S1H). We then consider the frequencies with which the procedures are triggered in an LTE network as reported in the dataset of Metsälä *et al.* [71] to obtain the total amount of signals exchanged due to each procedure.

Table 3.1 shows the results of our analysis. We can see that Intelligent Edge is able to reduce significantly the amount of signals exchanged in the network, and it is thus indicated to mitigate the signaling overheads caused by a high number of subscribers using always-on applications. This example showcases the potentialities of a network architecture in which the network functions are decomposed using the networking planes of a mobile network. While the Intelligent Edge refactoring has shown to be the most effective in mitigating signaling overheads, the network operator can refactor the network differently to meet the requirements coming from another use case.

3.2 Leveraging Traffic Sources and Destinations

In 5G networks, each network slice has dedicated virtual resources that are instantiated on the physical resources of a substrate network. More specifically, the instantiation of a network slice describes where its virtual resources are instantiated, *e.g.,* VNF A is instantiated at node 1 and VNF B is instantiated at node 2. Each network slice also has multiple ingress and



(a) LTE Architecture (b) Split RAN & Core

(c) Thin Edge (d) Intelligent Edge

Figure 3.2: In the LTE architecture, several network functions participate to tasks that belong to different networking planes, *e.g.*, the S-GW participates to both control plane and data plane tasks. Decomposing the network functions using the networking planes allows customizing the way in which the network functions are instantiated in the network, thus enabling network operators to tailor the network to meet different requirements. Illustrations (b), (c), and (d) are taken from Publication 1.

egress points, *i.e.*, network nodes through which traffic flows in the network and out of the network, respectively. The ingress nodes correspond to the base stations in the substrate network, while the egress nodes correspond to the set of User Plane Functions (UPFs) the network slice uses to reach the Internet [16]. The traffic from the base stations is conveyed through a set of network functions, which process the traffic and push it towards the different UPFs. This "bow-tie-shaped" organization is sub-optimal for two reasons, i) the network incurs a high bandwidth consumption because we convey the traffic from the ingress nodes to the network functions and from the network functions to the egress nodes, and ii) the centralized network functions are resource-hungry because they need to process a high volume of traffic. In this context, it is difficult to migrate the network functions, for example to make room for additional network slices. In the worst case,

	Frequency of signals					
Implementation	*IA*	*AtI*	*ItA (UE)*	*ItA (Net)*	*X2H*	*S1H*
LTE (Baseline)	17.5	204	247	255	120	4.4
Split RAN & Core	13	272	285	285	160	4
Thin Edge	12	204	247	240	128	3.2
Intelligent Edge	8.5	102	190	180	96	2.4

Table 3.1: Frequency of signals considering the different approaches to refactor the LTE network. For each procedure, the frequency corresponds to the number of signals exchanged by the network functions per busy hour per subscriber per base station. Intelligent Edge is the most effective in mitigating signaling overheads. Table from Publication 1.

the network is unable to host an additional network slice even if it would actually have the resources for hosting it.

The left side of Figure 3.3 shows an example of the inefficiency of the current network slicing model. The network operator coalesces network functions A and B to prevent the high volume of traffic between them from flowing on the links of the network. Nevertheless, A and B are resource-hungry network functions, and the only node which can host them is node 3. The obtained instantiation of the network slice is rather inefficient because the traffic from nodes 1 and 2 is conveyed to node 3, and then split again towards nodes 4 and 5, thus consuming a significant amount of bandwidth in the substrate network.

To address this inefficiency, we propose to identify all the pairs of ingress nodes and egress nodes for which a share of traffic comes in the network from the first node and goes out from the network through the second node. In the example of Figure 3.3, the traffic entering from node 1 goes out through node 4, while the traffic entering from node 2 goes out through node 5, so we have two pairs, i) nodes 1 and 4, and ii) nodes 2 and 5. We can thus decompose each network function into a group of network functions, each one processing the traffic of a specific pair of ingress and egress nodes. For example, we decompose network function A into two network functions, one processing the traffic between nodes 1 and 4, and one processing the traffic between nodes 2 and 5. By decomposing network functions using this approach, we obtain dedicated network functions for each pair of ingress and egress nodes within a network slice, thus we decompose a network slice into μslices. The benefits of μslicing are exemplified in the right side of Figure 3.3. The network functions we obtain after the decomposition are less resource-hungry and they are easier to migrate and to fit into the com-

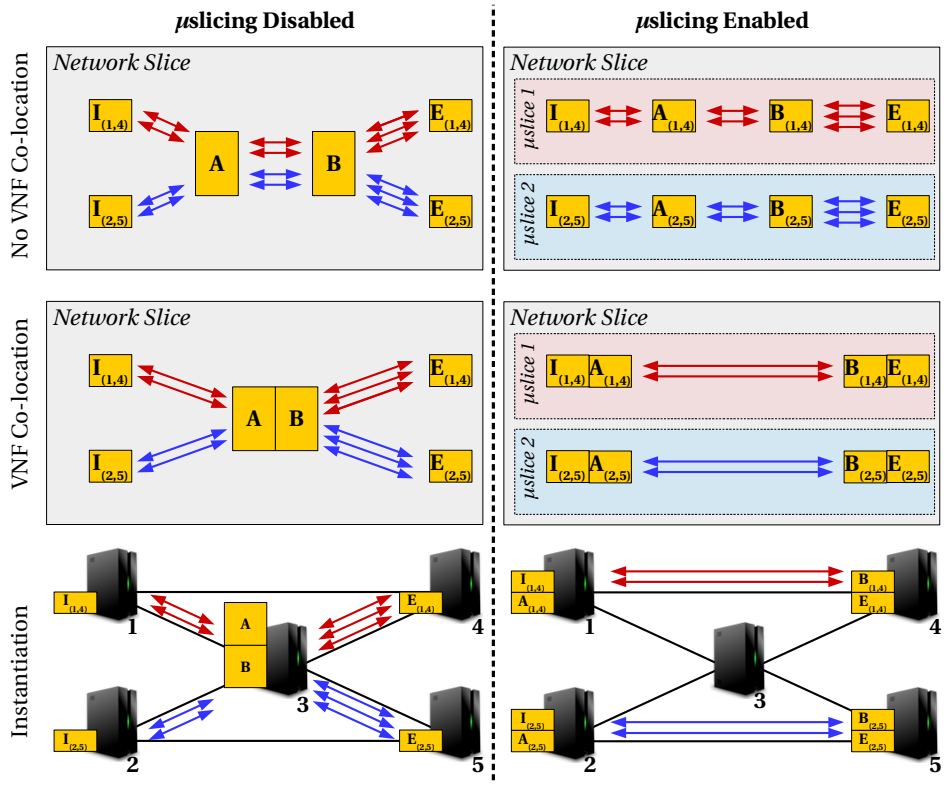


Figure 3.3: In this example, the traffic goes through a set of centralized, resource-hungry network functions, resulting in high bandwidth consumption. We can decompose each network function into a set of network functions, each of which processes the traffic flowing between a pair of ingress and egress nodes, thus obtaining μslices. Illustration from Publication 3.

puting nodes of the substrate network, providing to the network operator additional options for tailoring the network for different objectives, *e.g.*, saving bandwidth.

3.2.1 A Model for the Approach

To assess the validity of the proposal, we designed and implemented an ILP model. The idea of the model is to instantiate the network slices in the substrate network so that i) the requirements of the network slices are met and ii) the substrate network is not overloaded, while simultaneously iii) saving as much bandwidth as possible. In effect, the problem the model

aims to solve is an example of a resource allocation problem [58]. The model includes a way to toggle μslicing ON or OFF. By switching μslicing OFF, we can evaluate the benefits and the drawbacks of μslicing when compared to vanilla network slicing. The goals of our evaluation are to assess i) if μslicing allows the network operator to save more bandwidth than network slicing, and ii) if μslicing allows hosting the network slices in input when network slicing would have failed instead.

We illustrate now the key concepts of the ILP model, while a detailed description can be found in Publication 3.[2] The capacity of the substrate network, as well as the requirements of the use cases, are abstracted as computation, bandwidth, and latency. Moreover, each network slice declares a set of ingress-egress node pairs in the substrate network. The model thus takes care that all the traffic originating at the ingress node flows through the appropriate set of network functions and eventually ends in the egress node. The decision variables of the model are defined as follows:

$$locV_{n,s,v} = \begin{cases} 1 & \text{if VNF } v \text{ serving } \mu\text{slice } s \text{ is instantiated at node } n \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

The model thus decides where to instantiate the network functions of each μslice in the substrate network, while the virtual links between the network functions are instantiated on the shortest path between the computing nodes hosting the network functions. As previously mentioned, the objective of the model is to minimize the usage of bandwidth of the links in the substrate network, which can be interpreted as maximizing the communication between network functions which are co-located in the same computing node. The resulting objective function follows:

$$\begin{aligned} &maximize: \\ &\sum_{\substack{s \in N \\ v_1, v_2 \in V \\ v_1 \neq v_2}} comS_{s,v_1,v_2} \cdot \left(\sum_{n \in N} locV_{n,s,v_1} \cdot locV_{n,s,v_2} \right) + \\ &\sum_{\substack{s_1 \in N \\ v \in V}} comS_{s_1,v,v} \cdot \left(\sum_{\substack{n, s_2 \in N \\ s_1 \neq s_2}} locV_{n,s_1,v} \cdot locV_{n,s_2,v} \right). \end{aligned} \tag{3.2}$$

[2] An improvement to the model reported in Publication 3 can be found in Appendix B.

The objective function consists of a sum of two terms. The first term deals with communications between different network functions serving the same μslice, which correspond to the majority of the communications. The second term deals with communications between instances of the same network function serving different μslices, which occur only during exceptional procedures, *e.g.,* handovers. Note that $comS_{s,v_1,v_2}$ corresponds to the bandwidth required between network functions v_1 and v_2 when serving μslice s. We also designed optimizations to reduce the complexity of the model by i) making it linear, thus avoiding the multiplications between the decision variables in the objective function, and ii) reducing the total amount of decision variables.

3.2.2 Evaluating the Benefits

We implemented the ILP model using IBM Cplex Optimization Studio 12.7.1 [7], and we made our implementation publicly available.[3] For our evaluation, we used the inputs of the LTE-based Model (Subsection 2.2.1). To take into account the improvements of the hardware in the years to come, we multiply the computing capacity of the nodes and the bandwidth of the links with a series of factors. More specifically, we consider the factors (x1, x2.5, x5, x7.5, x10) for both the number of servers p in each computing node, thus obtaining computing capacities (p_1, p_2, p_3, p_4, p_5), and the bandwidth c in each link, thus obtaining bandwidth values (c_1, c_2, c_3, c_4, c_5). As a result, each computing node is equipped with a number of servers from 100 (p_1) to 1000 (p_5), and the bandwidth of each link goes from 10 Gbps (c_1) to 100 Gbps (c_5). In our experiments, we did not use the same factors to reduce the latency because this would imply the need for physically moving the computing nodes, which might not be possible from a practical standpoint. For each (network topology, computing capacity, bandwidth) tuple, we generate ten different problem instances by varying the pairs of ingress and egress nodes of each network slice, and we run the ILP model to solve them, considering μslicing both enabled and disabled. For each run, we allowed the solver to run for a max of ten minutes. We chose such a tight time budget because 5G use cases require network slices to be instantiated and reconfigured potentially many times per day [98].

In Table 3.2 we summarize the results of our evaluation to answer our first question. For each (network topology, computing capacity, bandwidth) tuple, we separated the savings in control-plane traffic and the savings in data-plane traffic, and for both of them we computed the average with

[3] `https://version.helsinki.fi/matteo.pozza/avatarifip19`

Topology	Computing Capacity	Bandwidth				
		c_1	c_2	c_3	c_4	c_5
Beijing	p_5	↑ , ↑	2.68, 2.99	2.36, 1.98	2.21, 2.12	2.43, 2.35
	p_4	↑ , ↑	2.34, 2.57	2.28, 2.32	2.28, 2.23	2.27, 2.25
	p_3	↑ , ↑	2.41, 2.50	2.26, 2.02	2.27, 2.16	2.23, 1.98
	p_2	— , —	2.36, 2.71	2.29, 2.24	2.37, 2.24	2.34, 2.24
	p_1	— , —	↑ , ↑	↑ , ↑	↑ , ↑	↑ , ↑
Tokyo	p_5	↑ , ↑	2.63, 2.70	2.34, 2.36	2.22, 2.23	2.30, 2.33
	p_4	↑ , ↑	2.46, 2.99	2.29, 2.44	2.24, 2.35	2.26, 2.37
	p_3	↑ , ↑	2.49, 2.58	2.26, 2.30	2.34, 2.29	2.23, 2.25
	p_2	↑ , ↑	2.40, 2.88	2.33, 2.24	2.20, 2.36	2.30, 2.40
	p_1	↑ , ↑	↑ , ↑	↑ , ↑	↑ , ↑	↑ , ↑
New York	p_5	2.40, 4.11	2.39, 2.57	2.34, 2.53	2.45, 2.52	2.26, 2.51
	p_4	2.22, 2.80	2.30, 2.46	2.42, 2.26	2.39, 2.50	2.49, 2.59
	p_3	3.39, 3.10	2.33, 2.56	2.52, 2.64	2.31, 2.51	2.25, 2.45
	p_2	2.19, 2.75	2.37, 2.57	2.37, 2.33	2.38, 2.67	2.46, 2.43
	p_1	↑ , ↑	↑ , ↑	↑ , ↑	↑ , ↑	↑ , ↑

Table 3.2: Ratio of average traffic savings $\frac{\mu slicing\ enabled}{\mu slicing\ disabled}$. For each (topology, computing capacity, bandwidth) tuple, the first value concerns the control plane and the second value concerns the data plane. Table from Publication 3.

μslicing enabled and with μslicing disabled. Table 3.2 shows the ratios between the μslicing-enabled values and the μslicing-disabled values; ratios higher than one indicate that μslicing is more effective in saving bandwidth, while ratios lower than one indicate that μslicing is not effective in saving bandwidth. An arrow pointing upwards (↑) indicates that the solver was able to find solutions to the problem instances only when μslicing was enabled, while a long dash (—) indicates that the solver was not able to find solutions to the problem instances at all. We can see that μslicing always allows the network operator to save around two times more traffic compared to when μslicing is disabled.

Figure 3.4 aims to reply to our second evaluation question, *i.e.,* assessing if μslicing allows instantiating the network slices in the substrate network when vanilla network slicing would fail instead. More specifically, Figure 3.4 sheds more light on the results presented in Table 3.2 showing that vanilla network slicing fails and μslicing succeeds when considering certain (network topology, computing capacity, bandwidth) tuples. For each tuple, we consider the ratio of problem instances solved when μslicing is enabled, *i.e.,* the feasibility ratio with μslicing enabled, and the feasibility ratio with

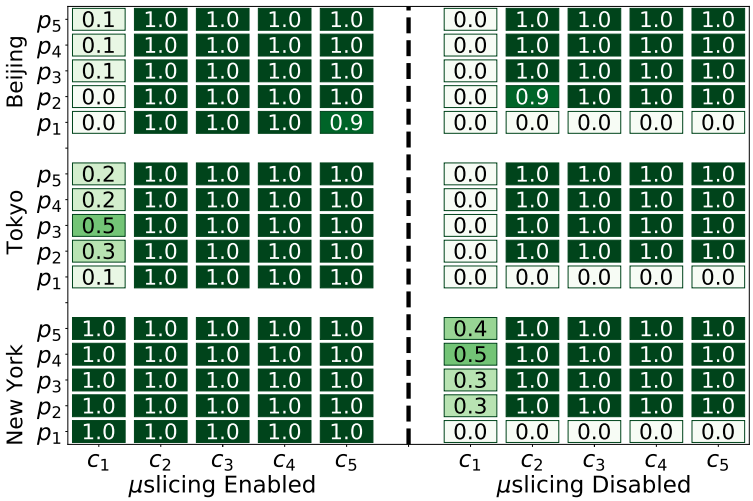


Figure 3.4: Feasibility ratio, *i.e.,* fraction of solved problem instances, for each (topology, computing capacity, bandwidth) tuple. Considering the scenarios with limited resources, *i.e.,* c_1 or p_1, the solver is able to solve a higher fraction of problem instances when μslicing is enabled. Illustration from Publication 3.

μslicing disabled. For example, a feasibility ratio equal to one means that the solver was able to solve all ten problem instances. By examining the data in Figure 3.4, we can see that μslicing is indeed more effective than vanilla network slicing when considering substrate networks with limited resources, *i.e.,* when considering c_1 or p_1.

We can conclude by saying that μslicing, *i.e.,* decomposing network functions using the sources and targets of traffic, indeed allows network operators to save bandwidth. Most importantly, this decomposition allows network operators to host the network slices in the substrate network when it would not have been possible otherwise.

3.3 Decoupling State Management from Data Store

Network functions are built to be scalable. Each network function consists of a set of instances whose size varies with the load in the network, and each instance is designed to fully leverage the resources made available in the computing node it runs on. This imposes non-trivial challenges in the handling of the network function's state, *i.e.,* the information used by the

network function to provide its functionalities. For example, a network function performing Network Address Translation, simply referred to as NAT, handles a pool of (IP address, port) pairs that are used to masquerade the identities of the senders of the flows traversing the network function, and managing the pool requires a tight coordination between the instances of the NAT. For this reason, researchers have developed state management systems [43,84], *i.e.,* dedicated systems that take care of the entire life cycle of state information on behalf of the network functions.

For managing state information, these systems internally use a data store, which is chosen depending on the features the system is designed to offer. For example, StatelessNF [57] uses a remote Key-Value Store (KVS) to increase reliability, while S6 [101] leverages a Distributed Hash Table (DHT) within the network function instances to boost performance. As shown in the top part of Figure 3.5, the chosen data store affects the Application Programming Interface (API) that the state management system exposes to the network functions. A change in the API thus results in a time-consuming and error-prone refactoring of the network functions [43, 61].

Nevertheless, using the network functions with different use cases involves also changing the data store used for managing the state of the network functions with the data store most suited for the requirements of the use case(s) being served. There is thus a need for a strong decoupling between the state management of a network function and data store adopted for storing the state, so that network operators can change the data store being used without the need for refactoring the network functions. To this end, we propose FlexState[4], a state management system that realizes this decomposition. FlexState exposes a single API, which developers use to build the network functions, while offering to the network operators an easy mechanism for changing the data store adopted for state management. Crucially, FlexState decomposes the network functions while preserving their scalability.

3.3.1 Key Enablers

To realize the decoupling between the state management and the data store adopted, FlexState leverages two key enablers, i) the API exposed to the packet processing logic of the network functions, and ii) a range of data store drivers, which translate the API calls using the query language of the data store. The main objective in designing the FlexState API is to fulfill the needs network functions have in terms of accessing and modifying

[4] `https://version.helsinki.fi/matteo.pozza/flexstate`

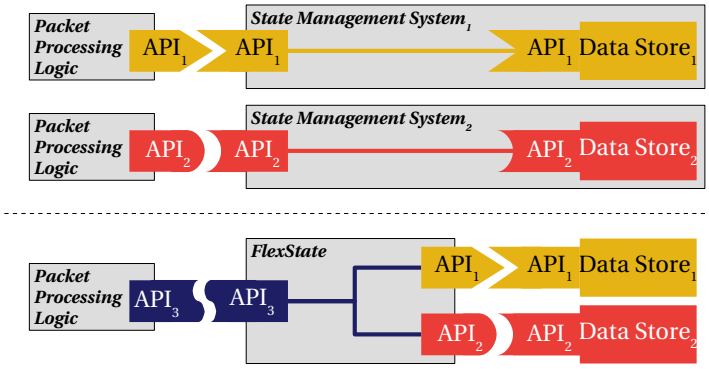


Figure 3.5: A change in the data store used for state management requires significant refactoring efforts on the network functions. The proposed system, FlexState, exposes a single Application Programming Interface (API) to the network functions and it allows network operators to easily change the data store adopted. Illustration from Manuscript 1.

state information, so that the packet processing logic does not need to be refactored when a new data store is released. As shown in Table 3.3, we examined the APIs of the state management systems to identify the set of features to include in the FlexState API. In the following, we present how the FlexState API supports the two most widely used features, namely name-value pairs and collections. A discussion on how to support the remaining features can be found in Manuscript 1.

Name-Value Pairs. The API allows storing a generic blob of bytes using a string as an identifier, *i.e.*, a name-value pair. The operations supported by the pair correspond to the operations prescribed by a Create, Read, Update, Delete (CRUD) interface. A special type of name-value pair is the counter, *i.e.*, a name-value pair whose value is an integer. Counters are widely used in network functions for implementing functionalities such as counting the total number of packets traversing the network function. In addition to the CRUD operations, the counter also supports an additional call, `add(value)`, which increments the counter by `value` directly in the data store.

Collections. The API supports collections, such as lists, sets, and maps. These data structures are used in a variety of network function tasks, such as associating an (IP address, port) pair to each flow going through a NAT. In addition to the CRUD operations, the calls supported by the collections

System	API Features: Name-Value Pairs	Collections	Consistency Tuning	Locking	Timers	Merging
Split/Merge [84]	✓	○	✓	✓	✓	✓
OpenNF [43]	✓	○	✓	○	○	✓
StatelessNF [57]	✓	○	○	○	✓	○
S6 [101]	✓	✓	✓	○	○	✓
libVNF [73]	✓	○	○	○	○	○
CHC [60]	✓	✓	○	○	○	○

Table 3.3: Supported features of the APIs of state management systems. Note that ✓indicates a supported feature, and ○ indicates an unsupported feature. The variability in the supported features results in customized APIs which are incompatible with each other. Table from Manuscript 1.

are inspired to the calls offered by the collections in the C++ standard container library [6]. Similarly to name-value pairs, we offer a special collection, a countermap, *i.e.*, a set of pairs where each pair has a unique key and a counter. The countermap exposes an additional `addTo(key, value)` call, which increments by `value` the counter of the pair identified by `key`.

When receiving the calls from the API, FlexState leverages a range of data store drivers, which translate the calls using the query language of the data store. Table 3.4 shows an example of translation considering two carrier-grade data stores. The network operator provides in input to FlexState a configuration file, which specifies the data store driver to be used and the parameters to connect to the data store, *i.e.*, IP address and port. It is thus enough to modify the configuration file to have FlexState adopting a different data store for state management. Moreover, when a new data store is released, it is enough to develop the corresponding driver following the specifications of the FlexState API and integrate the driver within FlexState, after which the network operator can modify the configuration file to instruct FlexState to use the new driver.

While decoupling the state management of the network functions from the data store adopted, FlexState must make sure that the scalability of network functions is not affected. For this reason, we designed FlexState by leveraging the principles of partitioning [10], also called sharding, according to which each CPU core running a network function instance has exclusive handling of a part of the state information. In other words, the entire state of a network function is divided among the NF instances first,

Type & Call	Redis	Cassandra
Counter add(n)	INCRBY counter_id n	UPDATE CounterTable SET value = value + n WHERE key=counter_id
Map insert(k,n)	HSET map_id k n	INSERT INTO MapTable (key1, key2, value) VALUES (map_id, k, n)
Countermap addTo(k,n)	HINCRBY cmap_id k n	UPDATE CountermapTable SET value = value + n WHERE key1=cmap_id AND key2=k

Table 3.4: Example of translating the FlexState API calls using the query language of Redis [9] and Cassandra [3]. Table from Manuscript 1.

and then each instance-specific portion is divided among the CPU cores of the server on which the instance is running. The advantage of this approach is that CPU cores do not need to communicate between each other because they have no shared data, thus avoiding the slowdowns of inter-CPU synchronization. Consequently, scalability is improved because the more CPU cores are assigned to a NF instance, the better the instance performs as more packets are processed in parallel. A challenge in adopting partitioning is the need for designing network function state in a partitioning-aware manner. For example, considering the NAT example, the pool of (IP address, port) pairs need to be carefully partitioned among the NF instances and the CPU cores of each NF instance. We discuss this aspect in more detail in Manuscript 1.

3.3.2 Implementation and Evaluation

To assess the feasibility of the system and to evaluate its performance, we implemented FlexState in our testbed. Our implementation of FlexState allows substituting the data store adopted for state management by changing just a configuration file in json format. We evaluated our implementation with the objective of showing that it is indeed possible to change the data store adopted without refactoring the network functions while preserving the scalability of the network functions. We configured the testbed to send to FlexState 50K random TCP flows saturating a 10 Gbps link, *i.e.,* at 14.88 Mpps [34], for a time interval of 15 seconds, and we repeated each test 10 times. We implemented two of the most commonly deployed network functions atop of FlexState, *i.e.,* a NAT and a load balancer [93], and the drivers for two data stores, *i.e.,* Redis [9] and Cassandra [3]. We chose Redis and Cassandra because they are designed to meet different

requirements, *i.e.,* Redis offers strong consistency [73] while Cassandra offers high availability [63]. We also implemented the driver for a custom in-memory hashmap, which is used for benchmarking purposes only. Note that a network operator might run the network function and the data store used for state management in different computing nodes to meet strict reliability requirements. To measure the impact that this choice has on the performance, we run Redis and Cassandra both locally, *i.e.,* in the same computing node running the NF instance, and remotely, *i.e.,* in a different computing node.

Figure 3.6 shows the rate of packets processed by the network functions using FlexState when varying the data store adopted and the number of CPU cores assigned to FlexState. We can see that the performance of the network functions improve with the number of CPU cores assigned to FlexState, so we can thus conclude that the network functions using FlexState are indeed able to scale with the resources assigned. We can also see that FlexState allows the network functions to process packets at 10 Mpps, which is in line with the packet processing rate recorded by other state management systems [34, 57]. Note that the network functions are capable of recording this packet processing rate thanks also to specific performance optimizations that we designed and implemented in FlexState. These performance optimizations are discussed in Section 4.1.

We observe that the performance of both network functions drop when we assign more than 24 CPU cores to FlexState. The reason behind this behavior lays in the architecture of the testbed in which we ran out tests. The computing nodes of the testbed are equipped with 24 physical cores, and hyperthreading is enabled on the nodes, so the nodes are equipped with 24 additional virtual cores multiplexed over the physical cores. In our test, we have configured the nodes to make use of the physical cores first, and to use the virtual cores only when there are no spare physical cores anymore. Nevertheless, using a virtual core is sub-optimal for a network function because the processing of the packets of the corresponding physical core needs to be interrupted regularly, hence the drop in performance.

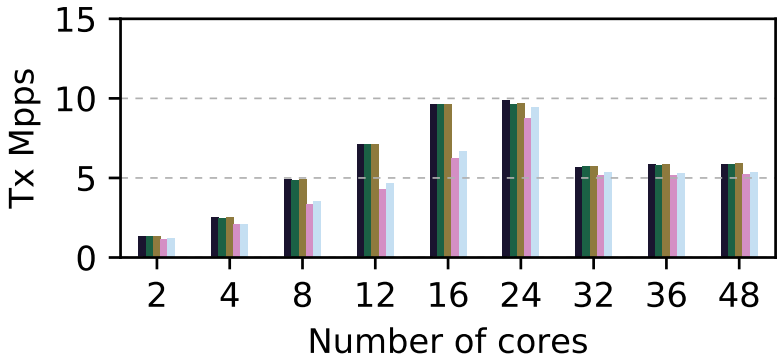
We can also see that there are no significant differences in performance between running the data store locally or remotely, and in a few cases running the data store remotely is even advantageous. This counter-intuitive behavior can be explained as follows. The design of FlexState decouples the packet processing loop from the communication with the data store, and thus the packet processing rate should not be affected by the location of the data store (Section 4.1). However, running the data store locally can lead the operating system to interleave the execution of FlexState with the

execution of the data store on the same CPU cores, thus possibly worsening the performance.

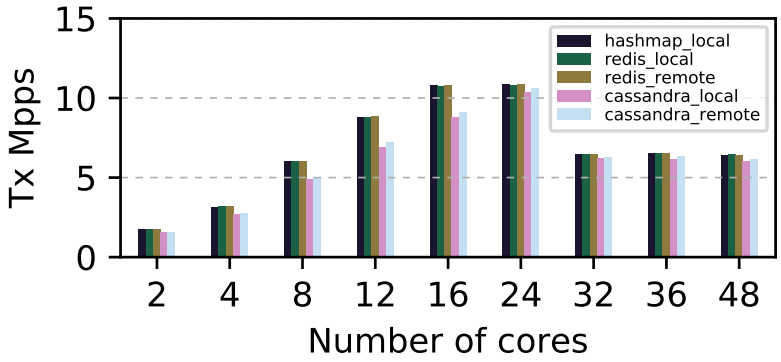
To conclude, FlexState demonstrates that it is indeed possible to decouple the state management in network functions from the data store used for storing state information. The decoupling allows network operators i) to substitute the data store used for state management without the need for refactoring the network functions and ii) to preserve the scalability of the network functions.

3.4 Summary

We have described three different ways in which we can decompose network functions: using the three identified networking planes, using the sources and the destinations of the traffic, and decoupling the state management from the data store used for storing state information. We have shown that each of these decompositions enables network operators to address the inefficiencies that limit network flexibility. The techniques illustrated in this chapter are thus key enablers to make networks ready for the challenges brought by 5G use cases.



(a) NAT



(b) Load Balancer

Figure 3.6: The network functions perform better when the number of CPU cores assigned to FlexState increases, *i.e.*, the network functions are indeed able to scale. The drop in performance recorded when adopting more than 24 cores is due to the usage of virtual cores. Illustrations from Manuscript 1.

Chapter 4

Addressing Limitations of Network Function Decomposition

So far we have seen how the proposed ways of decomposing network functions improve network flexibility. Network function decomposition is, nonetheless, a double-edged sword: on one hand, it helps in addressing several inefficiencies, and on the other hand it brings drawbacks. For example, the increase in the number of the network functions brought by network function decomposition can make network management tasks more cumbersome. In the following, we illustrate two key problems that stem from, or are exacerbated by, network function decomposition. We also illustrate the methods we designed to mitigate these problems and we demonstrate their effectiveness, showing that it is indeed possible to enjoy the benefits of network function decomposition without passively undergoing its drawbacks.

4.1 Overheads of Decoupling State Management from Data Store

In Section 3.3 we described FlexState, a system that realizes a strong decoupling between the state management in network functions and the data store used to keep their state. To achieve the decoupling, FlexState leverages an Application Programming Interface (API) and a range of data store drivers. The API is used by the packet processing logic of the network functions to issue operations on the state, while the data store drivers perform the API calls on the data stores. The abstractions provided by FlexState, *i.e.,* the API and the data store drivers, communicate synchronously with the data store by default. In other words, when an API call is issued, the

driver performs the corresponding operation on the data store, and the result obtained from the data store is returned to the caller. Nevertheless, there are two aspects of this workflow which compromise the performance of network functions. First, the caller always waits for the response to an API call to come back even when the result is not essential for the caller, *e.g.,* an update call only returns an acknowledgement about the completion of the operation. For these calls, the caller could continue its work without waiting synchronously for the response from the data store. Second, the rate at which network functions process packets is different from the rate at which data stores perform operations, but the two operations are normally coupled together, *i.e.,* FlexState issues an operation to the data store as soon as the packet processing logic asks for it. As a consequence, the rate at which the network function operates corresponds to the lowest between the packet processing rate and the rate of operations the data store supports. The combined effect of these two aspects is that network functions whose state management is decoupled from the data store experience a significant drop in performance, *i.e.,* they are unable to process packets at line rate.

4.1.1 Addressing the Drop in Performance

To address the two issues identified, we propose to use *no_wait* calls and asynchronous updates, respectively.

***no_wait* calls.** We extend FlexState API by introducing *no_wait* calls, whose semantic prescribes to ignore any result returned by the call. When using a no_wait call, the caller continues its work normally, while the operation associated with the call is scheduled on a different thread in the background. For example, a network function can leverage the no_wait version of the Update call to continue to process packets right after scheduling an update to a state information while the update is performed in background. In case the network function performs state operations for every processed packet, no_wait calls become essential for keeping a high packet processing rate, but some API calls are not suitable for a no_wait variant. For example, a network function issues a `read` call when the result of the call is needed to perform a certain task, thus there is little usefulness in providing a corresponding no_wait call. Consequently, network function developers should avoid issuing this type of calls within the packet processing loop.

Asynchronous updates. We complement FlexState with two additions, i) a per-CPU-core local cache for state information, and ii) a timer that

regulates the frequency with which the changes in the local cache are pushed to the data store. Instead of issuing the operations immediately to the data store, FlexState performs the operations on a local copy of the state, thus allowing fast processing of the packets. Simultaneously, a timer regulates the frequency with which a background thread commits the changes to the local copy of the state to the data store. The network operator sets the frequency with which the state is synchronized on the data store in a dedicated field of the configuration file fed in input to FlexState. In this way, the network operator can adjust FlexState to the requirements of the use case, *e.g.,* achieving high availability with a high synchronization frequency, and to the rate of operations supported by the chosen data store.

4.1.2 Assessing the Improvements

To quantify the benefits of the proposed optimizations, we built on top of FlexState a simple "counter" network function, *i.e.,* a network function which counts the number of packets going through it. In particular, we built two versions of the network function. The first version always waits for the API calls to return before proceeding further and it communicates synchronously with the data store (sync counter). Instead, the second version implements the two optimizations, thus it does not wait for the API call to return when scheduling increments to its internal counter, and the packet processing rate is decoupled from the rate of operations of the data store (async counter). We evaluated the two versions of the network function using the same inputs described in Subsection 3.3.2. In the evaluation, we configured both versions of the network function to use the in-memory hashmap, *i.e.,* the scenario in which the sync counter performs best because the state management incurs no network latency. Figure 4.1 shows how the two versions perform when changing the number of CPU cores assigned to FlexState. We can see that the async version clearly outperforms the sync version. While the async version achieves packet processing rates that are in line with the performance of state-of-the-art network functions, *i.e.,* 12 Mpps, the sync version is not capable of processing packets faster than 2 Mpps.

We conducted additional tests to showcase the capability of the optimizations to i) enable network functions to process packets at line rate, and ii) make the packet processing rate of the network function independent from the data store adopted and from the location of the data store adopted. We considered three network functions, *i.e.,* the async version of the counter network function, a NAT, and a load balancer, and we configured FlexState to perform the tests with each one of the data stores

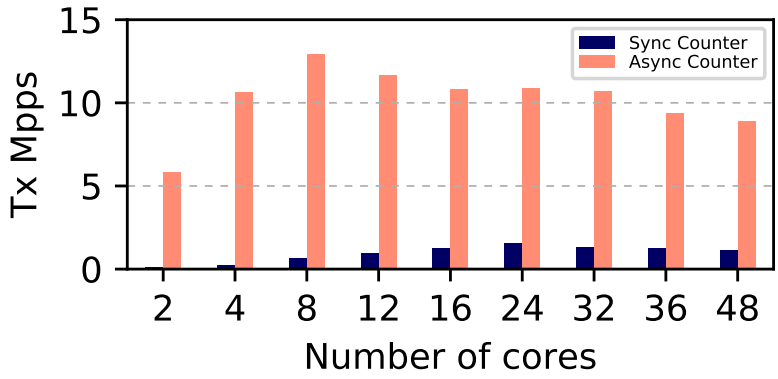


Figure 4.1: We implemented a simple network function that counts the packets flowing through it, without optimizations (sync) and with optimizations (async). To avoid any effect on the performance due to network latency, we configured the network function to use a local, in-memory hashmap. The async version reaches a packet processing rate of 12 Mpps, while the sync version never records a rate higher than 2 Mpps. Illustration from Manuscript 1.

available, *i.e.*, hashmap, Redis, and Cassandra. Moreover, we consider each location for the data stores, *i.e.*, local and remote. Figure 4.2 illustrates the results from our tests. Each bar represents the best rate we recorded by varying the number of CPU cores assigned to FlexState when considering the specific (network function, data store, data store location) tuple. We can observe that all network functions approach a packet processing rate of 10 Mpps, which is in line with the performance recorded adopting other state management systems [34, 57]. We can also note that varying the data store and its location does not significantly affect the performance recorded by the network functions. These results thus confirm that the optimizations are indeed effective in mitigating the performance drawbacks introduced by decoupling the state management in network functions from the data store used to keep their state.

We are not the first to advocate for no_wait calls and caching mechanisms, and previous works have also highlighted their benefits [60, 101]. We show that they are also effective when the state management of network functions is decoupled from the data store adopted. Moreover, we introduce a configurable timer, which the network operator can set depending on the use case and the capabilities of the data store adopted.

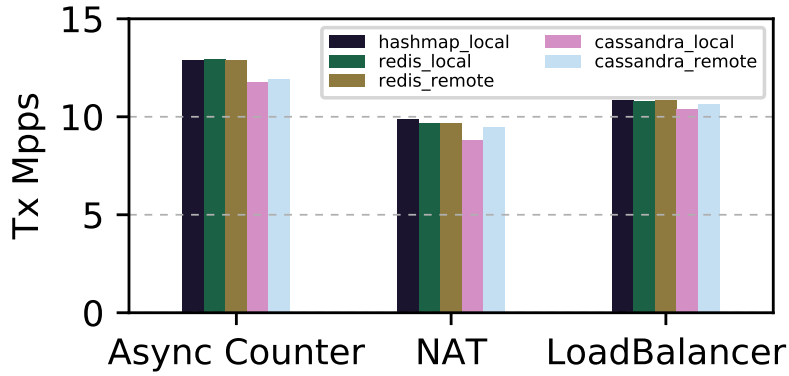


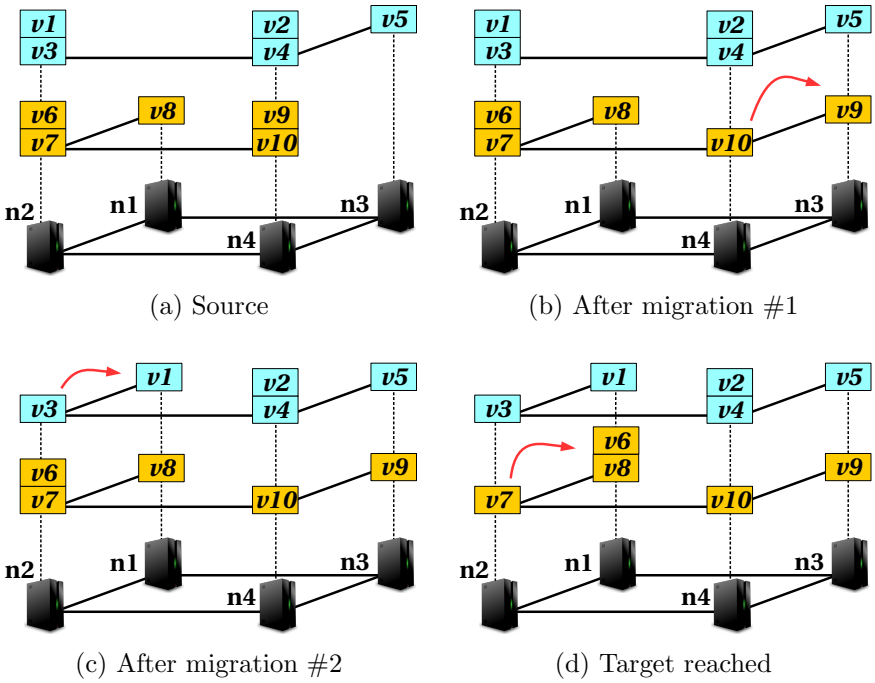
Figure 4.2: Adopting the optimizations enables the network functions to process packets at line rate, *i.e.*, 10 Mpps, regardless of the data store adopted and the location chosen for the data store. Illustration from Manuscript 1.

4.2 Reconfiguring Networks with a High Number of Network Functions

Network Function Virtualization (NFV) enables network operators to instantiate and migrate the network functions in the network according to the requirements from the use cases and optimization policies, *i.e.*, saving bandwidth. In other words, NFV allows network operators to customize the network *configuration*, *i.e.*, where the Virtual Network Functions (VNFs) are instantiated in the network[1]. The variability in the traffic load, as well as the addition or the removal of use cases being served, are examples of factors that trigger the need for *reconfiguring* the network, *i.e.*, migrating the VNFs in the network to obtain a more desirable configuration. Figure 4.3 exemplifies the process of reconfiguring a network, which entails a sequence of VNF migrations from the initial configuration, called source, to obtain a new configuration, called target.

The stringent requirements coming from 5G use cases, as well as the introduction of network slicing, bring a set of non-trivial challenges to the network reconfiguration process:

[1]In Section 3.2 we use the expression *network slice instantiation*, which describes where the virtual resources of the network slice are mapped on the physical resources of the substrate network. The network configuration describes the instantiations of the network slices altogether.



(a) Source (b) After migration #1

(c) After migration #2 (d) Target reached

Figure 4.3: Reconfiguring a network involves migrating the VNFs, which are initially instantiated according to a source configuration, to obtain a different configuration, called target. Illustration from Publication 4.

1. Due to the diverse set of requirements of the use cases, networks are expected to undergo reconfigurations more and more often. The time budget for planning and actuating the reconfiguration is therefore very limited [98].[2]

2. The Service-Level Objectives (SLOs) that network operators agree upon are characterized by extremely limited tolerance on their violation [67]. Therefore, the configurations traversed during the reconfiguration process must satisfy all the requirements of the use cases, *i.e.*, they must correspond to *feasible* configurations.

3. Network slicing introduces dedicated network functions for each use case (Section 3.2). The increasing number of network functions to

[2]In this thesis, we focus only on the planning of the network reconfiguration, *i.e.*, finding the sequence of migrations to reconfigure the network according to the target configuration, and not on the actuation of the reconfiguration plan, *i.e.*, performing the migrations to obtain the target configuration.

handle during the reconfiguration process therefore makes the problem more cumbersome. This is exacerbated by network function decomposition, which can increase further the number of network functions.

We can study the network reconfiguration problem by abstracting it using a migration graph, as shown in Figure 4.4. In a migration graph, each node represents a network configuration, *i.e.,* a placement of the virtual resources in the network, and each edge represents a migration of a VNF from one node in the network to another one. In the migration graph, the initial configuration (source) and the desired configuration (target) correspond to two distinct nodes. Therefore, the reconfiguration problem corresponds to identifying a path between the source node and the target node in the migration graph so that i) each node in the path represents a feasible configuration, and ii) the path is minimal, *i.e.,* involving a number of migrations as low as possible.

Researchers have successfully used A* search to solve network reconfiguration problems in the past [33], *i.e.,* to find the shortest path between source and target nodes in the migration graph. Nevertheless, the algorithm in its original form fails in providing solutions to the reconfiguration problem in 5G networks, especially considering the limited time budget available. The worst-case time complexity of A* search is indeed $\mathcal{O}(b^d)$, where b is the branching factor, *i.e.,* the maximum number of successors of any node, and d is the depth at which the target node is found [87]. In common artificial intelligence problems, such as traversing the game tree of a chess game, b can be as high as 35 while the value of d is around 100 [44]. Considering the migration graph, at each configuration we can migrate any VNF in any computing node in the network, so the branching factor of each node in the migration graph is $V * N$, where V is the total number of VNFs in the network and N is the total number of computing nodes. Network slicing and network function decomposition increase V significantly, thus affecting the complexity of the problem and making it even more hard to solve within the limited time budget available.

4.2.1 Tackling the Problem

We propose Matryoshka, an algorithmic approach to solve the reconfiguration problem in 5G networks. Matryoshka consists of three parts, i) a set of optimizations for A* search, which reduce the width of the migration graph and speed up the search, ii) a divide-and-conquer approach to perform A* search, and iii) a method to run several A* searches in parallel.

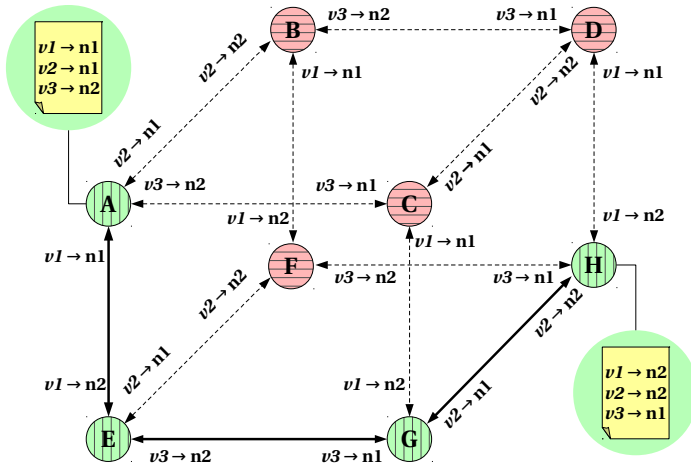
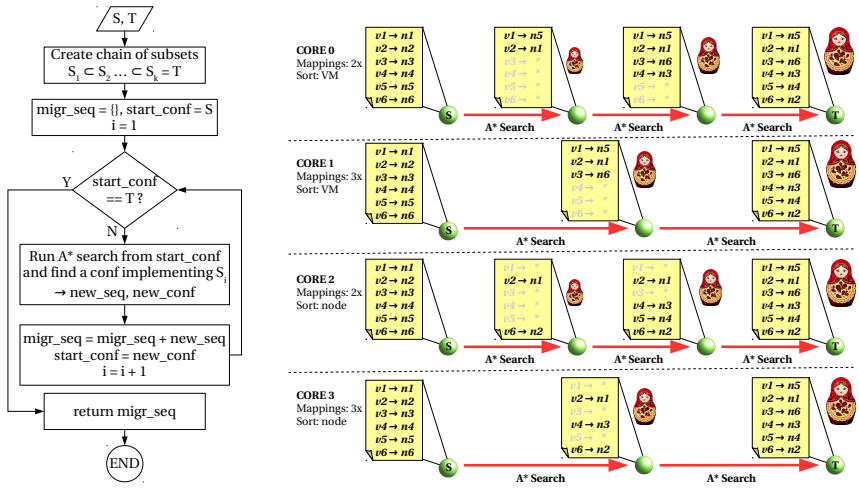


Figure 4.4: The reconfiguration problem can be studied using a migration graph, whose nodes represent configurations of the network and edges represent migrations of VNFs. Green nodes with vertical hatching correspond to feasible configurations, while red nodes with horizontal hatching correspond to infeasible configurations. Illustration from Publication 4.

We present here the last two parts, which correspond to the design pillars of Matryoshka, while a detailed description of the A* search optimizations can be found in Publication 4. Our implementation of Matryoshka is publicly available at `https://version.helsinki.fi/matteo.pozza/matryoshka`.

Divide-and-conquer approach. As shown in Figure 4.4, we can describe each configuration as a set of *mappings*, where each mapping indicates the computing node where a VNF is hosted. For example, we use the mapping $v2 \rightarrow n2$ to indicate that VNF $v2$ is hosted at the computing node $n2$. When running A* search on the migration graph, the algorithm starts from the source configuration S, and it stops only when it finds a node whose configuration has all the mappings equal to the ones of the target configuration T. Instead of considering the mappings of T all at once, we create a chain of subsets of the mappings of T S_1, S_2, ... $S_k = T$ such that $\forall i < j, S_i \subset S_j$.[3] We run A* search starting from S and considering only the mappings in S_1. When we find a configuration Z_1 which contains all the mappings in S_1, i) we record the sequence of migrations from S to Z_1, ii) we set the new source configuration to Z_1, and iii) we run A* search

[3]We elaborate on the strategy for the creation of the chain of subsets when we discuss how Matryoshka runs several A* searches in parallel.



(a) Divide-and-conquer (b) Parallelization

Figure 4.5: Using the divide-and-conquer approach, Matryoshka runs A* search focusing at every step only on a subset of the VNF migrations to perform. Matryoshka also boosts the chances of finding a solution to the problem by running multiple A* searches in parallel. Illustrations from Publication 4.

considering the mappings in S_2. The approach, which is described in Figure 4.5(a), continues considering the subsets in the chain till the target T is reached. Following this approach, we reduce the search scope of A* search by prioritizing the configurations containing one or more of the mappings in the subset over the multitude of configurations available, thus simplifying the task of the A* search at each step.

Parallelization. There is no single way of creating the chain of subsets S_1, S_2, ... S_k because one has the freedom of deciding i) the criterion used in selecting the mappings for each subset, and ii) the number of mappings included in each subset. For example, we can decide to sort the mappings using the target computing nodes, so that we reconfigure the network one computing node at a time. Nevertheless, there is no clear indication on why one strategy in selecting the mappings is better than another. Therefore, Matryoshka leverages the available CPU cores to run several A* searches in parallel, each one adopting a different strategy in the creation of the chain of subsets, as shown in Figure 4.5(b). There is not a minimum number of strategies to try, but the more strategies are tried in parallel, the higher the

chances of finding a solution quickly. The A* search running in parallel can be configured in *independent* and *sharing* modes. In independent mode, each A* search runs independently, and Matryoshka stops as soon as one of the A* searches finds a complete sequence of migrations to reconfigure the network from S to T. In sharing mode, Matryoshka stops as soon as one of the A* searches finds a partial solution, *i.e.,* a configuration containing a subset of the mappings of T, then it sets all the A* searches to consider the partial solution as the new S, and the A* searches are restarted again. The cycle is repeated till configuration T is reached.

4.2.2 Evaluating the Approach

To evaluate the effectiveness of Matryoshka, we need a set of problem instances, *i.e.,* pairs of source-target configurations for which we need to find a sequence of VNF migrations. We created a dataset of 241 problem instances [4] using the inputs provided by the 5G-based Model, which is described in Subsection 2.2.2.[5] For each problem instance, we run Matryoshka, both in independent mode (MAT-I) and in sharing mode (MAT-S), and the state-of-the-art A* search (SOTA) described in the work of Dow *et al.* [33], allowing up to 60 minutes of running time. Figure 4.6 shows the Cumulative Distribution Function (CDF) of the percentage of problem instances solved over time. We can see that Matryoshka outperforms the state-of-the-art A* search because it solves around 10 times more problem instances considering the whole 60-minute time window. Moreover, Matryoshka is much faster because it solves around 8 times more problem instances already within the first 10 minutes of execution.

In Figure 4.6 we observe that although Matryoshka outperforms the state of the art, there are still around 50% of the problem instances which are not solved, even when considering 60 minutes of solving time. While Matryoshka corresponds to a good starting point, additional work is required to completely master the network reconfiguration problem. We discuss about future work in this direction in Section 5.3.

[4] `https://version.helsinki.fi/matteo.pozza/matryoshkadata`

[5] The 5G-based model also considers the NF instances composing each network function. When performing migrations using the inputs of the 5G-based model, we do not migrate an entire network function with all its NF instances, but we migrate single NF instances, thus allowing the NF instances of the same network function to be potentially running in different computing nodes.

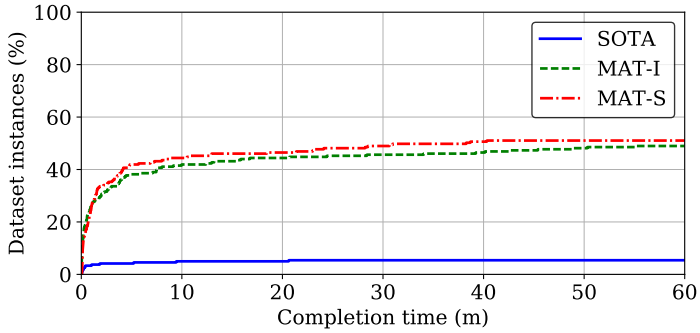


Figure 4.6: Cumulative Distribution Function (CDF) of the number of problem instances solved over time. Matryoshka (MAT-I and MAT-S) outperforms the state of the art (SOTA) by finding solutions to 8 times more problem instances compared to SOTA within the first 10 minutes. Illustration from Publication 4.

4.3 Summary

This chapter pursued a two-fold aim. First, the chapter provided evidence that decomposing network functions can also bring disadvantages. For example, the increase in the number of network functions in the network can make network management tasks more challenging. Second, the chapter shows that it is indeed possible to enjoy the benefits brought by network function decomposition *and* mitigate the drawbacks coming along at the same time. We illustrated the techniques we designed to mitigate two key drawbacks that we identified and we demonstrated their effectiveness.

Chapter 5

Conclusion

In this chapter, we summarize the outcomes from the work presented in this thesis. First, we examine again the research question and we assess the achievement of the objectives that we formulated in the beginning of the thesis (Section 5.1). We then provide a unified view of the several improvements to the state of the art we presented, highlighting the overall scientific contribution brought by the thesis (Section 5.2). Finally, we discuss the directions we envision for future work (Section 5.3).

5.1 Research Questions Revisited

RQ1. What are the state-of-the-art techniques to improve network flexibility?

We have examined the scientific literature, and we have identified a set of use cases that are known for triggering a need for flexibility in the network, *e.g.*, providing connectivity services to the survivors of a natural disaster. Among the solutions that have been proposed for these use cases, we identified Networks-In-a-Box (NIBs) as the most suitable solution to meet the flexibility requirements of the 5G use cases as well. NIBs enclose all the network functions required to run a fully fledged network, and they are capable of self-organizing to work alone as well as with other network functions. Thanks to these design features, NIBs are able to meet the requirements of use cases such as after-disaster scenarios and provisioning of Internet access in developing countries. In our analysis, we found that NIBs are nevertheless insufficient to meet the flexibility required by 5G use cases, especially when considering aspects such as the granularity of the network functions and scalability. We conclude that the state of the art

lacks solutions to provide the flexibility required by 5G use cases, which motivates the work conducted to address the remaining research questions.

RQ2. How can we faithfully represent a mobile network and the requirements of the 5G use cases?

In Section 2.2 we have illustrated the approach we followed to create mobile network models for our experiments. The approach consists in combining the information coming from i) technical and scientific documentation, such as 3GPP specifications, ii) interviews with experts in the field, and iii) measurements performed in our testbed. We showcase our approach by presenting two models we built. The first model, *i.e.,* LTE-based, uses information from current LTE networks to represent 5G-enabled scenarios, for example by representing the network slices as disjoint sets of LTE network functions. The LTE-based model is thus more suited to represent the networks during the early steps in the transition from LTE to 5G. The second model, *i.e.,* 5G-based, represents 5G networks by leveraging forecasts about the traffic in the years to come and information from technical specifications. The 5G-based model is more indicated for representing 5G networks which are built from scratch.

RQ3. Can we efficiently mitigate signaling overheads by decomposing the network functions using the networking planes?

In Section 3.1 we have presented a methodology to decompose the network functions using the networking planes. Specifically, we identified three networking planes a network function can participate in, *i.e.,* data forwarding, network control, and storing services. We then split each network function into fine-grained network functions, where each network function provides the functionalities for a single plane. We have shown that the obtained architecture enables network operators to tailor the network for a variety of use cases. As an example, we have shown that coalescing the control NFs at the edge of the network allows reducing significantly the number of signals exchanged in the network. We thus answer to RQ3 positively.

RQ4. Can we optimize the usage of network resources by decomposing the network functions using sources and destinations of the traffic?

The uplink traffic of a network slice originates at several base stations, then it is conveyed through a small set of network functions, and then it flows out towards several gateways, while the downlink traffic follows the same path, but in the opposite direction. Conveying the traffic from several

sources to a few network functions makes the network functions resource-hungry and therefore difficult to migrate. In addition, in Section 3.2 we have shown that this architecture results in high bandwidth consumption, ultimately leading to the network being unable to host additional network slices. We propose to decompose each network function in a set of network functions, each of which processes the traffic flowing between a specific pair of source and target nodes. Given the smaller workload, the obtained network functions are less resource-hungry and they are easier to migrate in the network, thus enabling network operators to instantiate them more efficiently. In our evaluation, we observed that the obtained architecture allows saving around two times more of the bandwidth than normal network slicing, thus answering RQ4 positively.

RQ5. Can we adapt the network functions to different use cases by changing the data store used for managing the state in an efficient manner?

To meet the different requirements from 5G use cases, network operators need to change the data store used for managing the state of the network functions, but this operation currently involves refactoring heavily the network functions. In Section 3.3 we propose FlexState, a system that decouples the state management from the data store adopted, thus making the packet processing logic independent from the data store chosen for state management. Crucially, our system is designed so that network functions can scale with the resources available. We have successfully tested FlexState with two of the most commonly deployed network functions, namely a NAT and a load balancer. More specifically, we were able to connect the network functions with a variety of data stores by just changing a configuration file. We also verified that the network functions improve their performance when we increase the amount of resources assigned to them, *i.e.*, they are indeed able to scale. Therefore, we can answer RQ5 positively.

RQ6. Can we decouple the state management in network functions from the data store adopted and preserve line-rate performance?

Network functions incur severe performance drawbacks due to the indirection introduced by decoupling the state management from the data store used to keep the state information. In designing FlexState, we have included two key performance optimizations to address this issue, namely no_wait calls and asynchronous updates (Section 4.1). When issuing a no_wait call, the caller does not wait for the response from the state man-

agement to come back, but instead it continues with the subsequent operations. no_wait calls are thus useful for speeding up the packet processing loop because the state management operations are scheduled in a thread in the background. Asynchronous updates enable decoupling the packet processing loop from the interactions with the data store, so that the network function and the data store can operate at different rates. Our tests show that network functions are able to process packets at line rate thanks to the described optimizations, so the answer to RQ6 is positive.

RQ7. How can we reconfigure networks with a high number of network functions?

Traditional approaches for planning network reconfiguration fail when dealing with the massive amount of network functions in 5G networks. The problem is exacerbated by the increase in the number of network functions to handle due to network function decomposition. In Section 4.2 we propose Matryoshka, an algorithmic technique that adopts a divide-and-conquer approach and uses multiple CPU cores to speed up the process of finding a solution. Our results show that i) Matryoshka outperforms the state of the art in terms of number of problem instances solved, and ii) there is still a significant number of problem instances for which Matryoshka is not able to identify a solution within the given time budget. We can conclude that Matryoshka does not completely address the problem, but it rather corresponds to a first step in the pursuit of a definitive solution.

5.2 Scientific Contribution

This thesis provides a number of contributions in the field of network planning and design, which are detailed in the publications appended to the thesis and they are now available to the scientific community. Among them, we highlight the ones we believe to be the two key contributions. The first one consists of a methodology to build realistic models for mobile networks, which can then be used in experiments. The methodology prescribes to define a mobile network by detailing out two components, i) the substrate network, and ii) the virtual network. Each of the two components can be abstracted as a graph: in the substrate network, the vertexes are computing nodes and the edges are links, while in the virtual network the vertexes correspond to network functions and the edges correspond to virtual links. The entities in the substrate network are labeled with capacities, *i.e.,* amount of resources available. Conversely, the entities in the virtual network are labeled with demands. To dimension both capacities

and demands, as well as to choose the topology of the graphs, we prescribe consulting three sources, i) technical and scientific documentation, such as 3GPP specifications, ii) interviews with experts in the domain, and iii) measurements on testbeds. We exemplify how to apply the methodology in Section 2.2 by describing two models we used in our experiments.

As the second key contribution, the thesis provides a range of tools that can be used for future work in studying how to improve network flexibility. In addition to the already-discussed models of mobile network, which can be used in other experiments, researchers can leverage the techniques we proposed to decompose network functions in different contexts, such as with application-layer network functions, as well as look for alternative ways to improve network flexibility. We also illustrated two of the key problems that arise with improved network flexibility, namely drops in performance and increased complexity of network management tasks. Researchers can now leverage this knowledge to propose new techniques of improving network flexibility which are explicitly designed to avoid or mitigate these drawbacks in a preventive manner.

5.3 Future Work

While the work described in this thesis certainly improves the flexibility of networks, there is still a long path to walk before achieving full network flexibility. Figure 5.1 shows what we envision to be the end of the path, *i.e.,* the architecture of a fully flexible network. The central element is the network Artificial Intelligence (AI), which receives three types of input. First, the use cases being served by the network are described by a set of requirements, which are fed to the network AI. Second, the network operator provides a set of objectives, *e.g.,* minimizing the bandwidth usage, or minimizing the usage of the resources in the computing nodes. Finally, the network itself provides information about its resources, in terms of both maximum capacity and current usage. The network AI uses the received input to decide for each network function the granularity, *i.e.,* how much it should be decomposed, the scale, *i.e.,* how many NF instances should be active, and the location, *i.e.,* if the network function should be migrated in the network. Once the new best configuration of the network has been identified, the network AI generates the instructions to reconfigure the network accordingly. Depending on the context, the network operator decides the frequency of the iterations of the described feedback loop, *e.g.,* a couple of times per day as well as every few minutes.

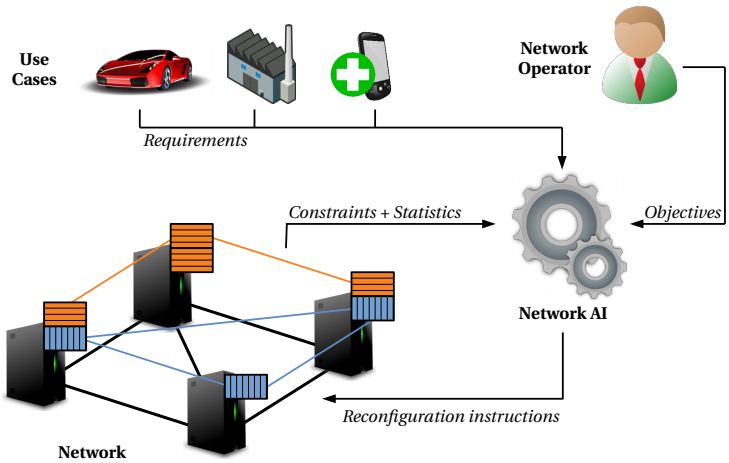


Figure 5.1: A vision of a flexible network. The Artificial Intelligence (AI) receives in input i) the requirements from the use cases, ii) the objectives from the network operator, and iii) the constraints of the physical resources in the network, together with the statistics describing the usage of the network. Based on the inputs, the network AI decides granularity, scale, and location of the network functions, then it issues the instructions to reconfigure the network accordingly.

The work in this thesis takes only a first step in realizing the components of the described network architecture. For example, in Section 3.1 we proposed three ways of configuring the network functions to achieve specific objectives, *e.g.,* reducing the signaling overheads. From the experience we accrued, we provide researchers with two recommendations. The first one is to invest time in identifying additional objectives that network operators might pursue. The second one is to design preventatively the configurations to meet such objectives, thus creating a portfolio of ready-to-use configurations for the network functions. The advantage is two-fold. First, the increase in the number of network functions to manage due to the decomposition of network functions makes it more difficult to identify the best configuration. Therefore, a set of pre-tested configurations facilitates the work of the network AI, which can use these configurations as starting points in the lookup for better configurations. Secondly, in the absence of the network AI, network operators could experience difficulties in managing a network involving a high number of network functions. The portfolio would thus correspond to a set of general guidelines that network operators can follow for configuring the network.

We would also like to raise a call for additional work in the generation of the reconfiguration instructions. In Section 4.2 we have described Matryoshka, a system we designed for this purpose. While Matryoshka certainly improves the state of the art, the results we obtain indicate that Matryoshka is not able to generate reconfiguration instructions for roughly half of the problem instances in our dataset. On the lines drawn by this thesis, we expect researchers soon to start developing new ways to decompose network functions further, which will exacerbate the problem of generating reconfiguration instructions. We recommend researchers to invest time and resources in designing techniques to tackle this problem more effectively.

To summarize, 5G use cases call for networks that are capable of adapting to different requirements. To achieve the flexibility that networks will require, we believe that decomposing the network functions is necessary but not sufficient. Networks need to be complemented with tools to fully automatize the decisions on the granularity, the scale, and the location of the network functions. Developing and making available such tools leads to a win-win situation for both the network users and the network operators. Thanks to network function decomposition, the requirements of the use cases can be met, and the network operators are relieved from the burden of managing a high number of network functions.

References

[1] SNDlib - library of test instances for Survivable fixed telecommunication Network Design. http://sndlib.zib.de/home.action, 2006. Last access: August 2020.

[2] The Internet Topology Zoo . http://www.topology-zoo.org/, 2013. Last access: August 2020.

[3] Apache Cassandra. http://cassandra.apache.org/, 2016. Last access: August 2020.

[4] SPEC - Standard Performance Evaluation Corporation - SPEC virt_sc ® 2013 - Results - Fourth Quarter 2017. https://www.spec.org/virt_sc2013/results/res2017q4/, 2017. Last access: August 2020.

[5] 5G Test Network Finland (5GTNF). https://5gtnf.fi/, 2020. Last access: August 2020.

[6] C++ Standard Library headers. https://en.cppreference.com/w/cpp/header, 2020. Last access: August 2020.

[7] IBM ILOG CPLEX Optimization Studio. https://www.ibm.com/products/ilog-cplex-optimization-studio, 2020. Last access: August 2020.

[8] Nokia Saving Lives - Intelligent solutions for worldwide rescue operations. https://networks.nokia.com/innovation/nokia-saving-lives, 2020. Last access: August 2020.

[9] Redis. https://redis.io/, 2020. Last access: August 2020.

[10] Seastar. http://seastar.io/, 2020. Last access: August 2020.

[11] 3GPP. Automatic Neighbour Relation (ANR) management - Concepts and requirements. TS 32.511, 3rd Generation Partnership Project (3GPP).

[12] 3GPP. General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access. TS 23.401, 3rd Generation Partnership Project (3GPP).

[13] 3GPP. Policy and Charging Control Architecture. TS 23.203, 3rd Generation Partnership Project (3GPP).

[14] 3rd Generation Partnership Project (3GPP). NG-RAN; Architecture description. TS 38.401.

[15] 3rd Generation Partnership Project (3GPP). Service requirements for next generation new services and markets. TS 22.261.

[16] 3rd Generation Partnership Project (3GPP). System architecture for the 5G System (5GS). TS 23.501.

[17] 5G-XHaul. Network Topology Definition. `https://www.5g-xhaul-project.eu/download/5G-XHaul_D2_4.pdf`, 2017. Last access: August 2020.

[18] 5GPPP. 5G Empowering Vertical Industries. `http://ec.europa.eu/newsroom/dae/document.cfm?doc_id=14322`, 2016. Last access: August 2020.

[19] 5GPPP. 5G Trials Roadmap. `https://5g-ppp.eu/wp-content/uploads/2018/11/5GInfraPPP_TrialsWG_Roadmap_Version4.0.pdf`, 2018. Last access: August 2020.

[20] Alcatel-Lucent. 7750 Service Router - Mobile Gateway. `http://lightspeedt.com/wp-content/uploads/2015/10/7750-Datasheet.pdf`, 2011. Last access: August 2020.

[21] X. An, R. Trivisonno, H. J. Einsiedler, D. von Hugo, K. Haensge, X. Huang, Q. Shen, D. Corujo, K. Mahmood, D. Trossen, M. Liebsch, F. Leitão, C. Phan, and F. Klamm. Architecture modularisation for next generation mobile networks. In *2017 European Conference on Networks and Communications, EuCNC 2017, Oulu, Finland, June 12-15, 2017*, pages 1–6. IEEE, 2017. DOI: `10.1109/EuCNC.2017.7980664`.

[22] S. Asif. *5G Mobile Communications: Concepts and Technologies*. CRC Press, 2018.

[23] A. Banerjee, R. Mahindra, K. Sundaresan, S. K. Kasera, K. van der Merwe, and S. Rangarajan. Scaling the LTE control-plane for future mobile access. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT 2015, Heidelberg, Germany, December 1-4, 2015*, pages 19:1–19:13. ACM, 2015. DOI: 10.1145/2716281.2836104.

[24] Bittium. LTE in Tactical Communications - White Paper. https://www.bittium.com/download/1713/white-paper-tactical-lte/pdf, 2014. Last access: August 2020.

[25] A. Chakraborty, E. Chai, K. Sundaresan, A. Khojastepour, and S. Rangarajan. SkyRAN: a self-organizing LTE RAN in the sky. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2018, Heraklion, Greece, December 04-07, 2018*, pages 280–292. ACM, 2018. DOI: 10.1145/3281411.3281437.

[26] Chemring Technology Solutions. Smartlink - Quicker response through increased collaboration. http://www.chemringts.com/~/media/Files/C/Chemring-TS-V2/documents/02467-smartlink.pdf, 2014. Last access: August 2020.

[27] Cisco. Cisco Data Center Infrastructure 2.5 Design Guide. https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCI_SRND_2_5a_book.pdf, 2011. Last access: August 2020.

[28] Cisco. IP Routing on Cisco IOS, IOS XE, and IOS XR: How a Router Works. https://www.ciscopress.com/articles/article.asp?p=2272154&seqNum=3, 2015. Last access: August 2020.

[29] Cisco. A Network Infrastructure for the Future with 5G - Introducing the NCS 500 Series of Routers. https://www.cisco.com/c/dam/m/en_us/network-intelligence/service-provider/digital-transformation/knowledge-network-webinars/pdfs/0306-msn-ckn.pdf, 2018. Last access: August 2020.

[30] Cisco. Network Emergency Response Vehicle. http://www.cisco.com/c/dam/en_us/solutions/industries/docs/gov/NERV_AAG.pdf, 2018. Last access: August 2020.

[31] Cisco. Cisco Annual Internet Report (2018–2023) White Paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf, 2020. Last access: August 2020.

[32] E. Dinc, M. Vondra, S. Hofmann, D. Schupke, M. Prytz, S. Bovelli, M. Frodigh, J. Zander, and C. Cavdar. In-Flight Broadband Connectivity: Architectures and Business Models for High Capacity Air-to-Ground Communications. *IEEE Communications Magazine*, 55(9):142–149, 2017. DOI: 10.1109/MCOM.2017.1601181.

[33] E. M. Dow and J. N. Matthews. WAYFINDER: parallel virtual machine reallocation through A* search. *Memetic Computing*, 8(4):255–267, 2016. DOI: 10.1007/s12293-016-0205-2.

[34] J. Duan, X. Yi, J. Wang, C. Wu, and F. Le. NetStar: A Future/Promise Framework for Asynchronous Network Functions. *IEEE JSAC*, 37(3):600–612, 2019. DOI: 10.1109/JSAC.2019.2894303.

[35] Ericsson. Handling of Signaling Storms in Mobile Networks. https://www.ericsson.com/49ecc9/assets/local/news/2015/8/handling-of-signaling-storms-in-mobile-networks-august.pdf, 2015. Last access: August 2020.

[36] Ericsson. A vision of the 5G core: flexibility for new business opportunities. https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/a-vision-of-the-5g-core-flexibility-for-new-business-opportunities, 2016. Last access: August 2020.

[37] Ericsson. Ericsson Mobility Report November 2019. https://www.ericsson.com/4acd7e/assets/local/mobility-report/documents/2019/emr-november-2019.pdf, 2019. Last access: August 2020.

[38] European Commission - Digital Transformation Monitor. Inflight entertainment and communication: Technologies and market. https://ec.europa.eu/growth/tools-databases/dem/monitor/sites/default/files/DTM_Aeronautics%20-%20Inflight%20v1.pdf, 2017. Last access: August 2020.

[39] European Telecommunications Standards Institute (ETSI). Network Functions Virtualisation - An Introduction, Benefits, Enablers,

Challenges & Call for Action. https://portal.etsi.org/NFV/NFV_White_Paper.pdf, 2012. Last access: August 2020.

[40] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach. Virtual Network Embedding: A Survey. *IEEE Communications Surveys and Tutorials*, 15(4):1888–1906, 2013. DOI: 10.1109/SURV.2013.013013.00155.

[41] C. Fuerst, S. Schmid, P. L. Suresh, and P. Costa. Kraken: Online and Elastic Resource Reservations for Cloud Datacenters. *IEEE/ACM Trans. Netw.*, 26(1):422–435, 2018. DOI: 10.1109/TNET.2017.2782006.

[42] X. Ge, H. Cheng, M. Guizani, and T. Han. 5G wireless backhaul networks: challenges and research advances. *IEEE Network*, 28(6):6–11, 2014. DOI: 10.1109/MNET.2014.6963798.

[43] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. OpenNF: enabling innovation in network function control. In *SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, pages 163–174, 2014. DOI: 10.1145/2619239.2626313.

[44] C. Grosan and A. Abraham. *Intelligent Systems - A Modern Approach*, volume 17 of *Intelligent Systems Reference Library*. Springer, 2011. DOI: 10.1007/978-3-642-21004-4.

[45] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015. DOI: 10.1109/MCOM.2015.7045396.

[46] Huawei. Weather the signaling storm. https://www.huawei.com/mediafiles/CORPORATE/PDF/Magazine/communicate/61/HW_094153.pdf, 2011. Last access: August 2020.

[47] Huawei. 5G: A Technology Vision. https://www.huawei.com/mediafiles/CORPORATE/PDF/Magazine/WinWin/HW_329327.pdf, 2014. Last access: August 2020.

[48] Huawei. eLTE Broadband Access Solution - QoS Technical Guide. https://actfornet.com/HUAWEI_ELTE_DOCS/Huawei%20eLTE2.3%20Broadband%20Access%20Solution%20QoS%20Technical%20Guide.pdf, 2014. Last access: August 2020.

[49] D. Iland and E. M. Belding. Emergenet: robust, rapidly deployable cellular networks. *IEEE Communications Magazine*, 52(12):74–80, 2014. DOI: 10.1109/MCOM.2014.6979955.

[50] Intel. Towards Achieving High Performance in 5G Mobile Packet Core's User Plane Function. https://builders.intel.com/docs/networkbuilders/towards-achieving-high-performance-in-5g-mobile-packet-cores-user-plane-function.pdf, 2018. Last access: August 2020.

[51] Intel. Lighting Up the 5G Core with a High-Speed User Plane on Intel®Architecture. https://builders.intel.com/docs/networkbuilders/lighting-up-the-5g-core-with-a-high-speed-user-plane-on-intel-architecture.pdf, 2019. Last access: August 2020.

[52] International Telecommunication Union (ITU). Transport network support of IMT-2020/5G. https://www.itu.int/dms_pub/itu-t/opb/tut/T-TUT-HOME-2018-2-PDF-E.pdf, 2018. Last access: August 2020.

[53] International Telecommunications Union (ITU). IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond. https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.2083-0-201509-I!!PDF-E.pdf, 2015. Last access: August 2020.

[54] M. Jaber, M. A. Imran, R. Tafazolli, and A. Tukmanov. 5G Backhaul Challenges and Emerging Research Directions: A Survey. *IEEE Access*, 4:1743–1766, 2016. DOI: 10.1109/ACCESS.2016.2556011.

[55] H. Jang, Y. Lien, and T. Tsai. Rescue information system for earthquake disasters based on MANET emergency communication platform. In *Proceedings of the International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly, IWCMC 2009, Leipzig, Germany, June 21-24, 2009*, pages 623–627. ACM, 2009. DOI: 10.1145/1582379.1582514.

[56] X. Jin, L. E. Li, L. Vanbever, and J. Rexford. SoftCell: scalable and flexible cellular core network architecture. In *Conference on emerging Networking Experiments and Technologies, CoNEXT '13, Santa Barbara, CA, USA, December 9-12, 2013*, pages 163–174. ACM, 2013. DOI: 10.1145/2535372.2535377.

[57] M. Kablan, A. Alsudais, E. Keller, and F. Le. Stateless Network Functions: Breaking the Tight Coupling of State and Processing. In *NSDI'17, Boston, MA, USA, March 27-29, 2017*, pages 97–112, 2017.

[58] N. Katoh, A. Shioura, and T. Ibaraki. *Resource Allocation Problems*, pages 2897–2988. Springer New York, New York, NY, 2013. DOI: 10.1007/978-1-4419-7997-1_44.

[59] K. Katsalis, N. Nikaein, E. Schiller, R. Favraud, and T. I. Braun. 5G Architectural Design Patterns. In *IEEE International Conference on Communication, ICC 2015, London, United Kingdom, June 8-12, 2015, Workshop Proceedings*, pages 32–37. IEEE, 2016. DOI: 10.1109/ICCW.2016.7503760.

[60] J. Khalid and A. Akella. Correctness and Performance for Stateful Chained Network Functions. In *NSDI'19, Boston, MA, February 26-28, 2019.*, pages 501–516, 2019.

[61] J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella. Paving the Way for NFV: Simplifying Middlebox Modifications Using StateAlyzr. In *NSDI'16, Santa Clara, CA, USA, March 16-18, 2016*, pages 239–253, 2016.

[62] A. Kunz, I. Tanaka, and S. S. Husain. Disaster response in 3GPP mobile networks. In *IEEE International Conference on Communications, ICC 2013, Budapest, Hungary, June 9-13, 2013, Workshops Proceedings*, pages 1226–1231. IEEE, 2013. DOI: 10.1109/ICCW.2013.6649424.

[63] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *Operating Systems Review*, 44(2):35–40, 2010. DOI: 10.1145/1773912.1773922.

[64] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann. Logically centralized?: state distribution trade-offs in software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks, HotSDN@SIGCOMM 2012, Helsinki, Finland, August 13, 2012*, pages 1–6. ACM, 2012. DOI: 10.1145/2342441.2342443.

[65] J. Li, F. R. Yu, G. Deng, C. Luo, Z. Ming, and Q. Yan. Industrial Internet: A Survey on the Enabling Technologies, Applications, and Challenges. *IEEE Communications Surveys and Tutorials*, 19(3):1504–1526, 2017. DOI: 10.1109/COMST.2017.2691349.

[66] L. E. Li, Z. M. Mao, and J. Rexford. Toward Software-Defined Cellular Networks. In *European Workshop on Software Defined Networking, EWSDN 2012, Darmstadt, Germany, October 25-26, 2012*, pages 7–12. IEEE Computer Society, 2012. DOI: 10.1109/EWSDN.2012.28.

[67] D. Lugones, J. A. Aroca, Y. Jin, A. Sala, and V. Hilt. AidOps: a data-driven provisioning of high-availability services in cloud. In *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017, Santa Clara, CA, USA, September 24-27, 2017*, pages 466–478, 2017. DOI: 10.1145/3127479.3129250.

[68] F. Malandrino, C. Chiasserini, and S. Kirkpatrick. Cellular Network Traces Towards 5G: Usage, Analysis and Generation. *IEEE Trans. Mob. Comput.*, 17(3):529–542, 2018. DOI: 10.1109/TMC.2017.2737011.

[69] Matteo Pozza. Solving Signaling Storms in LTE Networks: a Software-Defined Cellular Architecture. Master Thesis, University of Padua, http://tesi.cab.unipd.it/53297/1/tesi_Pozza.pdf, 2016. Last access: August 2020.

[70] N. McKeown, T. E. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner. OpenFlow: enabling innovation in campus networks. *Computer Communication Review*, 38(2):69–74, 2008. DOI: 10.1145/1355734.1355746.

[71] E. M. Metsälä and J. Salmelin. *LTE Backhaul - Planning and Optimization*. Wiley, 2015.

[72] K. Miranda, A. Molinaro, and T. Razafindralambo. A survey on rapidly deployable solutions for post-disaster networks. *IEEE Communications Magazine*, 54(4):117–123, 2016. DOI: 10.1109/MCOM.2016.7452275.

[73] P. Naik, A. Kanase, T. Patel, and M. Vutukuru. libVNF: Building Virtual Network Functions Made Easy. In *Proceedings of the ACM Symposium on Cloud Computing, SoCC 2018, Carlsbad, CA, USA, October 11-13, 2018*, pages 212–224, 2018. DOI: 10.1145/3267809.3267831.

[74] Netmanias. Brief Survey on 5G RAN Practical Deployment Scenario: From 2-tier (4G) to 3-tier (5G). https://www.netmanias.com/en/post/reports/13103/5g-c-ran-fronthaul/brief-survey-on-5g-ran-practical-deployment-

scenario-from-2-tier-4g-to-3-tier-5g, 2018. Last access: August 2020.

[75] Netmanias. Progression from 4G to 5G. https://www.netmanias.com/en/post/blog/14214/5g/progression-from-4g-to-5g, 2019. Last access: August 2020.

[76] Next Generation Mobile Networks (NGMN). 5G White Paper. https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf, 2015. Last access: August 2020.

[77] Nokia. Managing LTE Core Network Signaling Traffic. https://insight.nokia.com/managing-lte-core-network-signaling-traffic, 2013. Last access: August 2020.

[78] Nokia. The remote and rural connectivity - A new community-based network model. https://www.cambridgewireless.co.uk/media/uploads/resources/Small%20Cell%20Group/29.06.17/SmallCell-29.06.17-Nokia-Enrico_Gattinotta.pdf, 2017. Last access: August 2020.

[79] Nokia. Microwave Network Evolution - Software Defined Networking and a Layer 3 VPN Vision. https://pages.nokia.com/T00275.5G.MW.white.paper.html, 2018. Last access: August 2020.

[80] Nokia. Nokia AirFrame open edge server. https://onestore.nokia.com/asset/205107, 2019. Last access: August 2020.

[81] Polaris Networks. LTE NetEPC. http://www.polarisnetworks.net/lte-netepc.html, 2020. Last access: August 2020.

[82] Z. A. Qazi, P. K. Penumarthi, V. Sekar, V. Gopalakrishnan, K. Joshi, and S. R. Das. KLEIN: A Minimally Disruptive Design for an Elastic Cellular Core. In *Proceedings of the Symposium on SDN Research, SOSR 2016, Santa Clara, CA, USA, March 14 - 15, 2016*, page 2. ACM, 2016. DOI: 10.1145/2890955.2890961.

[83] Qualcomm. Ultra-Reliable Low-Latency 5G for Industrial Automation. https://www.qualcomm.com/media/documents/files/read-the-white-paper-by-heavy-reading.pdf, 2018. Last access: August 2020.

[84] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield. Split/Merge: System Support for Elastic Execution in Virtual Middleboxes. In *NSDI'13, Lombard, IL, USA, April 2-5, 2013*, pages 227–240, 2013.

[85] P. Rost, A. Banchs, I. Berberana, M. Breitbach, M. Doll, H. Droste, C. Mannweiler, M. A. Puente, K. Samdanis, and B. Sayadi. Mobile network architecture evolution toward 5G. *IEEE Communications Magazine*, 54(5):84–91, 2016. DOI: 10.1109/MCOM.2016.7470940.

[86] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, D. Aziz, and H. Bakker. Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks. *IEEE Communications Magazine*, 55(5):72–79, 2017. DOI: 10.1109/MCOM.2017.1600920.

[87] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach, 3rd Edition*. Prentice Hall, 2010.

[88] J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, and X. Costa-Pérez. Overbooking network slices through yield-driven end-to-end orchestration. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2018, Heraklion, Greece, December 04-07, 2018*, pages 353–365. ACM, 2018. DOI: 10.1145/3281411.3281435.

[89] M. R. Sama, L. M. Contreras, J. Kaippallimalil, I. Akiyoshi, H. Qian, and H. Ni. Software-defined control of the virtualized mobile packet core. *IEEE Communications Magazine*, 53(2):107–115, 2015. DOI: 10.1109/MCOM.2015.7045398.

[90] Sandvine. The Mobile Internet Phenomena Report. https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2020/Phenomena/Mobile%20Phenomena%20Report%201H%202020%2020200219.pdf, 2020. Last access: August 2020.

[91] Z. Savic. LTE Design and Deployment Strategies. https://www.cisco.com/c/dam/global/en_ae/assets/expo2011/saudiarabia/pdfs/lte-design-and-deployment-strategies-zeljko-savic.pdf, 2011. Last access: August 2020.

[92] S. Sharma, R. Miller, and A. Francini. A Cloud-Native Approach to 5G Network Slicing. *IEEE Communications Magazine*, 55(8):120–127, 2017. DOI: 10.1109/MCOM.2017.1600942.

[93] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu. NFP: Enabling Network Function Parallelism in NFV. In *SIGCOMM'17, Los Angeles, CA, USA, August 21-25, 2017*, pages 43–56, 2017. DOI: 10.1145/3098822.3098826.

[94] S. Tabbane. Core network and transmission dimensioning. https://www.itu.int/en/ITU-D/Regional-Presence/AsiaPacific/SiteAssets/Pages/Events/2016/Aug-WBB-Iran/Wirelessbroadband/core%20network%20dimensioning.pdf, 2016. Last access: August 2020.

[95] M. Trevisan, D. Giordano, I. Drago, M. Mellia, and M. M. Munafò. Five years at the edge: watching internet from the ISP network. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2018, Heraklion, Greece, December 04-07, 2018*, pages 1–12. ACM, 2018. DOI: 10.1145/3281411.3281433.

[96] H. D. Trinh, N. Bui, J. Widmer, L. Giupponi, and P. Dini. Analysis and modeling of mobile traffic using real traces. In *28th IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications, PIMRC 2017, Montreal, QC, Canada, October 8-13, 2017*, pages 1–6. IEEE, 2017. DOI: 10.1109/PIMRC.2017.8292200.

[97] N. Uchida, K. Takahata, Y. Shibata, and N. Shiratori. Never Die Network Based on Cognitive Wireless Network and Satellite System for Large Scale Disaster. *JoWUA*, 3(3):74–93, 2012. DOI: 10.22667/JOWUA.2012.09.31.074.

[98] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos. The Algorithmic Aspects of Network Slicing. *IEEE Communications Magazine*, 55(8):112–119, 2017. DOI: 10.1109/MCOM.2017.1600939.

[99] Vodafone. Instant Network Emergency Response. https://www.vodafone.com/about/vodafone-foundation/our-projects/instant-network-emergency-response, 2020. Last access: August 2020.

[100] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang. A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications. *IEEE Access*, 5:6757–6779, 2017. DOI: 10.1109/ACCESS.2017.2685434.

[101] S. Woo, J. Sherry, S. Han, S. Moon, S. Ratnasamy, and S. Shenker. Elastic Scaling of Stateful Network Functions. In *NSDI'18, Renton, WA, USA, April 9-11, 2018*, pages 299–312, 2018.

[102] T. Wypych, R. Angelo, and F. Kuester. AirGSM: An unmanned, flying GSM cellular base station for flexible field communications. In *2012 IEEE Aerospace Conference*, pages 1–9, March 2012. DOI: 10.1109/AERO.2012.6187134.

[103] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang. Zenith: Utility-Aware Resource Allocation for Edge Computing. In *IEEE International Conference on Edge Computing, EDGE 2017, Honolulu, HI, USA, June 25-30, 2017*, pages 47–54. IEEE Computer Society, 2017. DOI: 10.1109/IEEE.EDGE.2017.15.

[104] Zakaria Tayq. Fronthaul integration and monitoring in 5G networks. Doctoral Dissertation, University of Limoges, https://tel.archives-ouvertes.fr/tel-01708493/document, 2017. Last access: August 2020.

Appendix A

Size of Signals of LTE Procedures

We report here the size of the signals of LTE procedures we have measured in our mobile network testbed. Please note the following aspects:

- All sizes are reported in bytes.
- We do not report the size of the signals exchanged between UE and eNB.
- In our testbed, each signal between MME and HSS corresponded to two database queries, so we use the sum of the size of the two queries to model the size of the signal.
- In our testbed, some network functions are run together ,*e.g.,* S-GW and P-GW, and it is not possible to measure the size of the signals exchanged between them. For such signals, we used the average size of a signal across all procedures, *i.e.,* 251.03 bytes.

Initial Attach

Name	**Direction**	**Size (B)**
Initial UE Message	eNB → MME	214
Authentication Information Request	MME → HSS	923
Authentication Information Answer	HSS → MME	624
Authentication Request	MME → eNB	138
Authentication Response	eNB → MME	138
Security Mode Command	MME → eNB	122
Security Mode Complete	eNB → MME	146
Update Location Request	MME → HSS	917
Update Location Answer	HSS → MME	2009

Create Session Request	MME → S-GW	262
Create Session Request	S-GW → P-GW	251.03
EPS Session Establishment Notification	P-GW → PCRF	251.03
Profile Request	PCRF → SPR	251.03
Profile Response	SPR → PCRF	251.03
EPS Session Establishment Ack	PCRF → P-GW	251.03
Create Session Response	P-GW → S-GW	251.03
Create Session Response	S-GW → MME	141
Initial Context Setup Request	MME → eNB	306
Initial Context Setup Response	eNB → MME	118
UE Capability Info Indication	eNB → MME	166
Attach Complete	eNB → MME	122
Modify Bearer Request	MME → S-GW	89
Modify Bearer Response	S-GW → MME	88

Idle-to-Active Transition

Name	Direction	Size (B)
Downlink Data Notification	S-GW → MME	251.03
Downlink Data Notification Ack	MME → S-GW	251.03
Paging	MME → eNB	251.03
Initial UE Message	eNB → MME	122
Initial Context Setup Request	MME → eNB	182
Initial Context Setup Response	eNB → MME	118
UE Capability Info Indication	eNB → MME	166
Modify Bearer Request	MME → S-GW	89
Modify Bearer Response	S-GW → MME	88

X2 Handover

Name	Direction	Size (B)
Handover Request	SeNB → TeNB	466
Handover Request Ack	TeNB → SeNB	174
Sequence Number Status Transfer	SeNB → TeNB	122
Path Switch Request	TeNB → MME	134
Modify Bearer Request	MME → S-GW	89
Modify Bearer Request	S-GW → P-GW	251.03
EPS Session Modification Notification	P-GW → PCRF	251.03

EPS Session Modification Ack	PCRF → P-GW	251.03
Modify Bearer Response	P-GW → S-GW	251.03
Modify Bearer Response	S-GW → MME	88
Path Switch Request Ack	MME → TeNB	170
UE Context Release	TeNB → SeNB	102

Downlink Procedure

Name	**Direction**	**Size (B)**
Downlink Packet	P-GW → S-GW	1494
Downlink Packet	S-GW → eNB	1494

Uplink Procedure

Name	**Direction**	**Size (B)**
Uplink Packet	eNB → S-GW	1494
Uplink Packet	S-GW → P-GW	1494

Appendix B

Improvements to the ILP Model

We designed the ILP model described in Publication 3 to consider latency constraints as well. In particular, the model ensures that procedures, such as initial attach and handovers, are carried out within the latency budgets prescribed by the use cases. To do so, we formulated the following constraints:

$$\begin{aligned} &\sum_{\substack{n_1,n_2\in N\\ v_1,v_2\in V}} pathDel_{n_1,n_2}\cdot IvcomP_{p,v_1,v_2}\cdot locVPair^{n_1,s,v_1}_{n_2,s,v_2} \\ &+\sum_{\substack{s_2,n_1,n_2\in N\\ v\in V}} pathDel_{n_1,n_2}\cdot IvcomP_{p,v,v}\cdot locVPair^{\ n_1,s,v}_{n_2,s_2,v} \\ &\qquad\qquad \leq mdelP_p, \quad \forall s\in N,\ p\in P, \end{aligned} \tag{B.1}$$

where

- N is the set of the computing nodes, V is the set of VNFs, and P is the set of procedures;
- $pathDel_{n_1,n_2}$ is the latency incurred in traversing the path between nodes n_1 and n_2;
- $IvcomP_{p,v_1,v_2}$ is a binary flag that indicates if VNFs v_1 and v_2 communicate to carry out procedure p or not;
- $locVPair^{n_1,s_1,v_1}_{n_2,s_2,v_2}$ is a binary flag that indicates if i) VNF v_1 serving μslice s_1 is instantiated at node n_1 and ii) VNF v_2 serving μslice s_2 is instantiated at node n_2;
- $mdelP_p$ is the latency budget for completing procedure p.

The constraints contains the sum of two terms in the left-hand side. Given a certain procedure p and a μslice s,[1] the first term accounts for the delay incurred in carrying out the procedure by VNFs that are different, *e.g.,* S-GW and MME exchanging signals during an initial attach. The second term accounts for any delay incurred in carrying out the procedure by instances of the same VNF, *e.g.,* eNBs transferring user data during a X2 handover. Note that normal procedures do not incur any delay due to communication between instances of the same VNF, the only exception being the X2 handover, during which the Source eNB transmits user data to the Target eNB.

Focusing on the second term of the sum in the left-hand side of the constraints B.1, we can note that the summation encompasses all the μslices of the original network slice, *i.e.,* the summation considers all $s2 \in N$. When considering the X2 handover procedures requested by μslice s, this term therefore accounts for the delay incurred between the eNB serving s and all the other eNBs serving the other μslices. In this form, the current formulation of the constraint is thus too stringent. Instead, when considering the X2 handover procedure, the constraint should consider every pair of μslices separately, so that only the latency incurred by the pair of eNBs of the two μslices is taken into account. The formulation of the constraints can be easily fixed as follows:

$$\begin{aligned} &\sum_{\substack{n_1,n_2 \in N \\ v_1,v_2 \in V}} pathDel_{n_1,n_2} \cdot IvcomP_{p,v_1,v_2} \cdot locVPair^{n_1,s_1,v_1}_{n_2,s_1,v_2} \\ &\quad + \sum_{\substack{n_1,n_2 \in N \\ v \in V}} pathDel_{n_1,n_2} \cdot IvcomP_{p,v,v} \cdot locVPair^{n_1,s_1,v}_{n_2,s_2,v} \\ &\qquad\qquad \leq mdelP_p, \quad \forall s1, s2 \in N,\ p \in P. \end{aligned} \tag{B.2}$$

Note that the new formulation does not affect the delay constraints for the other procedures because they do not incur any delay due to the communication of VNFs of the same type, *i.e.,* the contribution of the second term of the sum in the left-hand side is always 0. The only drawback of the formulation is the increase in the number of constraints because we are iterating over and additional index, *i.e.,* $s2 \in N$. Nevertheless, given that the contribution of such second term is null for the majority of the procedures, this results in duplicate constraints, *i.e.,* $|N|$ constraints equal to each other for each pair $s1 \in N$, $p \in P$, which are easily identified and discarded by modern solvers [7].

[1] Note that each μslice s is identified by its ingress node, therefore $s \in N$.

Note also that the usage of the formulation of the constraints B.1 is unlikely to have affected the validity of the results presented in Section 3.2 for the following reasons. The delay incurred by the links in the physical network is 1 ms, while the delay budget assigned to X2 handover procedures is around 500 ms [91]. Given that in our evaluation the considered topologies have a limited number of edges, *i.e.,* always less than 50 edges, and the number of μslices for each network slice is 5, we can conclude that the impact of the delay constraints in the problem instances studied is minimal.