Université Côte d'Azur
École doctorale Sciences et Technologies de l'Information et de la Communication

P.h.D. Thesis

to obtain the title of

Docteur en Sciences
de l'Université Côte d'Azur
Mention: Informatique

Defended by

Nicolas Huin
COATI Project
Inria, I3S (CNRS/UNS)

Energy efficient Software Defined Networks

Supervised by:
Frédéric Giroire & Dino Lopez

Defended: 28th September 2017

Reviewers:	Laurent Lefèvre	- Inria (Lyon, France)
	Arie M.C.A Koster	- RWTH Aachen (Germany)
Examinators:	Walid Dabbous	- Inria (Sophia Antipolis, France)
	Frédéric Giroire	- CNRS (Sophia Antipolis, France)
	Brigitte Jaumard	- Concordia University (Canada)
	Jérémie Leguay	- Huawei Technologies (Paris, France)
	Dino Lopez	- Université Côte d'Azur (Sophia Antipolis, France)
	François Vanderbeck	- University of Bordeaux 1 (Bordeaux, France)

Résumé

Au cours des dernières années, la croissance des architectures de réseaux de télécommunication a rapidement augmenté pour suivre un trafic en plein essor. En outre, leur consommation d'énergie est devenue un enjeu important, tant pour son impact économique qu'écologique. De multiples approches ont été proposées pour la réduire. Dans cette thèse, nous nous concentrons sur l'approche Energy Aware Routing (EAR) qui consiste à fournir un routage valide tout en diminuant le nombre d'équipements réseau actifs.

Cependant, les réseaux actuels ne sont pas adaptés au déploiement de politiques vertes globales en raison de leur gestion distribuée et de la nature fermée des périphériques réseau actuels. Les paradigmes de Software Defined Network (SDN) et de Network Function Virtualization (NFV) promettent de faciliter le déploiement de politiques vertes. En effet, le premier sépare le plan de contrôle et de données et offre donc une gestion centralisée du réseau. Le second propose de découpler le logiciel et le matériel des fonctions réseau et permet une plus grande flexibilité dans la création et la gestion des services réseau.

Dans cette thèse, nous nous concentrons sur les défis posés par ces paradigmes pour le déploiement de politiques EAR. Nous consacrons les deux premières parties aux SDNs. Nous étudions d'abord les contraintes de taille de table de routage causées par la complexité accrue des règles, puis le déploiement progressif de périphériques SDN dans un réseau actuel. Nous concentrons notre attention sur NFV dans la dernière partie, et plus particulièrement nous étudions les chaînes de fonctions de services.

Abstract

In the recent years, the growth of the architecture of telecommunication networks has been quickly increasing to keep up with a booming traffic. Moreover, the energy consumption of these infrastructures is becoming a growing issue, both for its economic and ecological impact. Multiple approaches were proposed to reduce the networks' power consumption such as decreasing the number of active elements. Indeed, networks are designed to handle high traffic, e.g., during the day, but are over-provisioned during the night. In this thesis, we focus on disabling links and routers inside the network while keeping a valid routing. This approach is known as Energy Aware Routing (EAR).

However current networks are not adapted to support the deployment of network-wide green policies due to their distributed management and the black-box nature of current network devices.The SDN and NFV paradigms bear the promise of bringing green policies to reality.The first one decouples the control and data plane and thus enable a centralized control of the network.The second one proposes to decouple the software and hardware of network functions and allows more flexibility in the creation and management of network services.

In this thesis, we focus on the challenges brought by these two paradigms for the deployment of EAR policies. We dedicated the first two parts to the SDN paradigm. We first study the forwarding table size constraints due to an increased complexity of rules. We then study the progressive deployment of SDN devices alongside legacy ones. We focus our attention on the NFV paradigm in the last part, and more particularly, we study the Service Function Chaining problem.

Contents

List of acronyms

ALR	Adaptive Link Rate
ASIC	Application-Specific Integrated Circuit
BFD	Bidirectional Forwarding Detection
CAGR	Compound Annual Growth Rate
CAM	Content-Addressable Memory
CapEx	Capital Expenditure
CaPex	Capital Expenditure
CDF	Cumulative Distribution Function
CG	Column Generation
CPU	Central Processing Unit
CR	Classical Routing
DPI	Deep Packet Inspection
EAR	Energy Aware Routing
EARC	Energy Aware Routing with Compression
EE-SFC	Energy-Efficient Service Function Chaining
EEE	Energy Efficient Ethernet
ENAtoR	SENAtoR without Smooth Link Shutdown
GRE	Generic Routing Encapsulation
hEAR	Energy Aware Routing in a hybrid SDN network
ICMP	Internet Control Message Protocol
ICT	Information and Communication Technologies

ILP	Integer Linear Program
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
LP	Linear Program
LPI	Low Power Idle
MAC	Media Access Control
MANO	Management & Orchestration
MCF	MultiCommodity Flow
MILP	Mixed Integer Linear Program
MP	Master Problem
MTU	Maximum Transmission Unit
NFV	Network Function Virtualization
NIC	Network Interface Card
ONF	Open Network Foundation
OpEx	Operational Expenditure
OSPF	Open Shortest Path First
OVS	Open vSwitch
P2P	Peer-to-peer
PoP	Point of Presence
PP	Pricing Problem
QoE	Quality of Experience
QoS	Quality of Service

RAM	Random Access Memory
RFC	Request for Comments
RMP	Restricted Master Problem
RTO	Retransmission Timeout
SDN	Software Defined Network
SENAtoR	Smooth ENergy Aware Routing
SFC	Service Function Chain
SLA	Service Level Agreement
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
TE	Traffic Engineering
ToR	Top of the Rack
ToS	Type of Service
UDP	User Datagram Protocol
vCPU	Virtual Central Processing Unit
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNF	Virtual Network Function
VoIP	Voice over IP

Chapter 1
Introduction

1.1 Motivations

With the recent trends of cloud computing, development of Internet of Things (IoT) and video on demand, the architecture of telecommunication networks needs to grow at an increasing rate to keep up with this booming traffic. The energy consumption of these infrastructures is becoming a growing issue, both for its economic and ecological impact. Estimations are that the Information and Communication Technologies (ICT) sector annual growth is greater than the worldwide energy consumption. Communication networks, personal computers, and data centers show a growth of 10%, 5%, and 4%, respectively, while worldwide energy consumption shows a 3% growth only [Van+14]. With the rising cost of energy and the increased sensitivity of environmental issues, it is becoming important to consider the reduction of the ICT's energy footprint.

In the recent years, component manufacturers for personal computers, smartphones and servers put a focus on reducing energy consumption and bringing CPU, GPU and other components close to energy proportionality. However, network appliances are still far from this ideal power consumption [Cha+08]. For example, network devices are still an important part of the power usage of a data center. According to [Abt+10], switches and routers represent 15% of the total energy consumption of a data center at peak traffic and about 50% when the traffic is low. Indeed, communication networks are designed to handle peak traffic and are thus over-provisioned during low traffic period, e.g., at night time. This leads to significant power-hungry idle devices during low load period. By shutting down a part of the network, we could achieve important energy savings. This could be done either by providing more energy efficient network appliances or by consolidating traffic in a few elements. Consolidation of the traffic can be achieved in several ways such as Virtual Machine (VM) migration in data centers [LLW11], dynamic base station switching for mobile network [Zho+09] or the minimization of the number of active devices, called Energy Aware Routing (EAR). In this thesis, we focus on EAR for backbone networks.

Current networks are not adapted to support the deployment of network-wide green policies. Legacy protocols used in these networks operate in a distributed manner, limiting the possibility of centralized green decisions. Indeed, legacy routers manage both the control and the data plane. The control plane represents the part of the network that takes the routing decisions. The data plane represents the part of the network in charge of forwarding packets according to the choices of the control plane. The emerging Software Defined Network (SDN) paradigm bears the promises to fast forward the deployment of more efficient green policies inside a network. By regrouping the network control plane inside one or multiple controllers, SDN provides a centralized view and control over the network. Routers are stripped of their intelligence and are demoted to forwarding devices. The management of the network is done by different applications running on the controllers such as routing or security applications. The controllers provide rules to the router for them to match flows against using, for instance, the OpenFlow API [McK+08].

Another roadblock for energy savings is the current implementation of network functions. Network operators can provide different types of services to their clients, and each service requires a particular set of functions to apply to the traffic, such as Deep Packet Inspection (DPI), firewall or video encoders. In legacy networks, each function is executed using a particular hardware called middlebox. These middleboxes reduce the flexibility of operators to deploy new services as operators cannot easily move them in the network. This lack of mobility also cripples the deployment of energy savings policies in the network. The locations of middleboxes are usually chosen to maximize the performances during rush hours. However, when traffic slows down during the night, the position of the middleboxes still constrains the request's paths.

The Network Function Virtualization (NFV) initiative, launched in 2012 [Gem+16], brings flexibility to network functions by replacing middleboxes with general purpose servers. These servers can hosts many VMs that can then execute any network functions. This paradigm would help the deployment of green policies tremendously, as we would be able to move around the functions on the network to adapt to the traffic.

In this thesis, we look at how to leverage the SDN and NFV paradigms to enable the deployment of EAR. More specifically, we consider the new constraints that come with these new technologies and how they affect the design of green policies.

1.2 Energy efficiency in networks

Since the work of Gupta et al. [GS03], a lot of works have considered the problem of energy efficiency of networks and we can observe two main complementary directions. The first one focuses on the network hardware and their power consumption. The goal is to minimize their energy usage, without impacting their performances, by providing more efficient circuits, efficient power-saving techniques and bringing them closer to energy proportionality. The second way considers the whole network and optimizes the routing of the data for reduced power consumption. The principle is to aggregate the traffic on a subset of hardware to minimize the number of active equipment.

In this thesis, we focus solely on network-wide energy efficiency for wired networks. Nevertheless, as they widely dictate the design of green policies for the whole network, we first present an overview of hardware energy efficient solutions, followed by the power models that derive from these mechanisms.

1.2.1 Hardware energy efficiency

As previously stated, a lot of work has been done to design servers close to energy proportional, but only a few improvement has been made on the network side of ICT, for all kinds of networks. While network hardware represents 15% of the power consumption of a data center during peak time, this number goes up to 50% during low-load periods [Abt+10]. This energy *inefficiency* during low traffic periods also holds for Internet Service Provider (ISP)'s networks, that we study in this thesis.

Indeed, in [Cha+08], Chabarek et al. conducted a study over a couple of devices and showed that an idle router, i.e., a router not forwarding any packets, consumes about 80% of the energy of a router at full capacity. Their proposed model decomposes the consumption of a router as the power consumption of its chassis and its network interfaces. By reducing the consumption of the interfaces, and more specifically, moving them towards energy proportionality, we could significantly lower the consumption of a router, and thus of the whole network.

We can also reduce the consumption of a router's links by shutting them

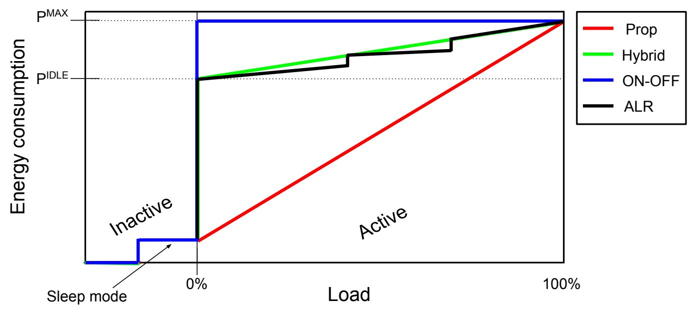


Figure 1.1: Prop, ON-OFF, Hybrid model for the power consumption of a link (u, v).

down. However, turning it back on requires a non-negligible amount of time. This reboot time could lead to significant packet losses in the network. Putting a link into sleep-mode might be a safer option at the cost of a slight trade-off between energy savings and re-activation time.

Energy Efficient Ethernet (IEEE 802.3 az standard [Chr+10]) proposes to use a Low Power Idle mode to reduce the power consumption of links when idle. We obtain the best energy savings when there is no traffic in the network. Otherwise, gains might not be significant enough when the traffic is low due to switching back and forth between the two states. In this case, it might be better to reduce the rate at which the link operates. This is the principle behind Adaptive Link Rate (ALR) [Chr+10]; Ethernet switches can change between different rates (e.g., 100 Mbit/s, 1 Gbit/s) depending on the bandwidth needed. Although it has been shown that ALR only achieve energy reduction when below 10% utilization [Hau15], it provides an alternative for reducing the energy consumption of *nearly* idle links. It can even be combined with Low Power Idle (LPI) to provide even better energy savings [Hau15].

Power model

In Figure 1.1, we present the different power models that can be used to model a link power consumption. The first model, Prop, represents a link that consumes energy proportionally to its load. An idle link would thus consume no power. This model represents a perfect scenario and is far from reality. The ON-OFF model follows the observations of [Cha+08] that a router consumption is nearly the same, idle or not. This model proposes an all or nothing approach: energy is used as soon as the link is active.

In this thesis, we use the Hybrid model, a mix of the two previous model, as it provides a good representation of the power consumption of current hardware. An active link has a baseline energy consumption and uses additional power proportional to its load. The power consumption of a link (u, v) can be given by

$$P_l(u,v) = x_{uv} \times P^{\text{IDLE}}(u,v) + \mathcal{T}_{uv} \times P^{\text{LOAD}}(u,v)$$

where x_{uv} represents the state of the link (ON or OFF), $P^{\text{IDLE}}(u,v)$ the baseline energy usage of the link, $\mathcal{T}_{uv}$ the fraction of the link's bandwidth used and $P^{\text{LOAD}}(u,v)$ the linear energy coefficient of the link. The maximum energy $P^{\text{MAX}}(u,v)$ used by a link is thus given by $P^{\text{LOAD}}(u,v) + P^{\text{IDLE}}(u,v)$. Note that it is easy to switch to an ON-OFF or Prop model by setting $P^{\text{LOAD}}(u,v)$ or $P^{\text{IDLE}}(u,v)$ at 0, respectively. For example, in Chapter 4, we only focus our study on the number of active links and thus set $P^{\text{LOAD}}(u,v)$ to 0.

The Hybrid model can even be further tuned to consider ALR. In this case, the linear part of the model would become a staircase function, as seen in Figure 1.1. However, we believe that this model would provide marginal improvement on the Hybrid model at the cost of a model with increased complexity

Note that for all models, we can differentiate inactive links into two states: offline and in sleep mode. As mentioned earlier, the difference between the two comes in a trade-off between re-activation time and energy savings.

Finally, we could consider shutting down the chassis of the router when all network interfaces are not used. However, and similarly to a link, the start-up time of a router is prohibitive and would impact the network performances negatively. Some routers provide a power-save mode that could provide slight energy savings and a low re-activation time. We further explore this power-save mode in Part II.

1.2.2 Network-wide energy efficiency

Even tough some solutions are designed to reduce the energy consumption of the network, network hardware is still far from being energy proportional. It is thus important to consider solutions that can shut down as much equipment as possible while maintaining an operational network. This approach

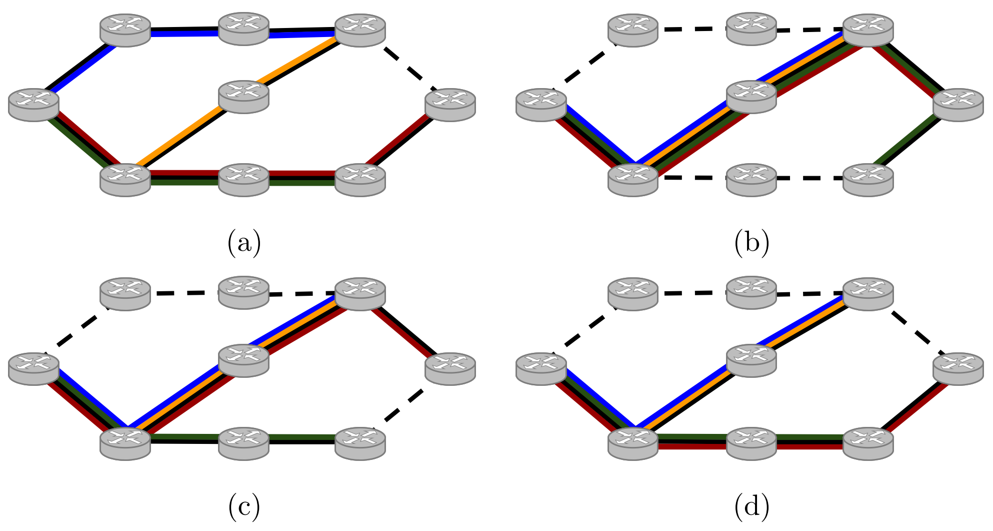


Figure 1.2: Example of Energy Aware Routing solutions.

is known as Energy Aware Routing and has led to a multitude of works in the network community, see, e.g., [CMN09; CMN12; Pha14; Rou+13] for backbone networks, [SLX10] for data center networks, [DGF12] for wireless networks or [R B+11] for a survey.

Let us now consider the network shown in Figure 1.2 composed of 9 routers, 10 links and 4 flows. We present in Figure 1.2 different EAR solutions for this network. We show in Figure 1.2a a routing that aims at minimizing the end-to-end delay of each flow. This routing gives us the possibility to shut down one link, using Energy Efficient Ethernet (EEE) for example, without any intervention of the network operator as no flow is forwarded on it. With sufficient links' capacities, we can reduce the network energy consumption, as shown in Figure 1.2b, by regrouping all flows on the links in the middle of the network. Doing this, we can shut down five links and put three routers to sleep. This approach represents the basic EAR solution. However, this solution nearly doubles the number of hops of the green flow. Depending on Quality of Service (QoS) requirements, it might be better to consider the solution presented in Figure 1.2c, where only four links are offline, but the number of hops of the green flow is reduced by 2. Works like [Cia+12] formulate the EAR problem around these constraints. Finally, the last solution, presented in Figure 1.2d, takes into consideration the reliability of the network. Indeed, if one of the equipment in the middle of the network fails, the

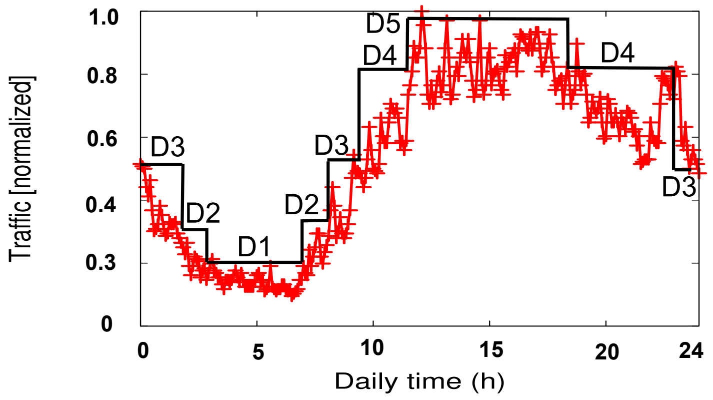


Figure 1.3: Variations of traffic on a typical France Telecom network link.

whole network is cut in half. By keeping the lower right link active, we can provide a network tolerant to one-link failure. Moreover, this solution allows to load balance the flows by redirecting the red flow instead of overloading the links in the middle. In this thesis, we only consider solutions like the one presented in Figure 1.2b. However, we evaluate our solutions using QoS metrics such as end-to-end delay and link load.

The concept is simple, but depending on the scenarios considered, different constraints can be added to the model. We present in the following the assumptions used in this thesis.

State of the links

Firstly, network interfaces' type plays a great deal in the way links can be shut down. Networks can be composed of bidirectional links, with different upload and download bandwidth. However, by shutting down one of the end point's network interface, both devices can no longer transmit data to each other. Ethernet networks show this behavior as one link can forward communications both ways.

However, if the links are unidirectional, we can fine-tune the subset of active elements in the network by choosing only to shut down communications one way or both ways. Optical fiber networks display this type of behavior as one fiber link will forward data one way.

Traffic estimation

Instability of traffic might severely impact the efficiency of green policies. As previously stated, switching the states of network equipment is not instantaneous. However, due to high multiplexing ISP traffic is stable and follows predictable daily and weekly patterns [OSP13]. We discuss ways to estimate the traffic in Chapter 6 and how to react to unpredicted changes in traffic due to failures or flash crowd.

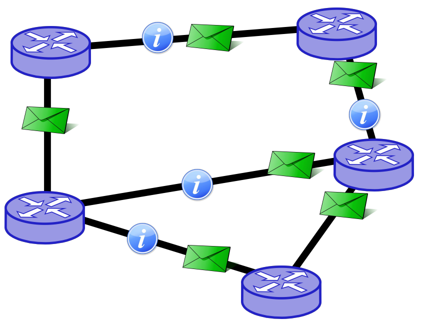
Even with predictable traffic patterns, an effective network-wide green policy has to carefully select the interval at which it should update the subset of active equipment. A too big interval will result in small energy savings, and a too small one will lead to network instability. Only a few configurations are sufficient to obtain near-optimal energy savings, as shown in [Ara+16]. Changing the state of the devices only a few times a day reduces the chance of decreasing the network performances.

Figure 1.3 shows the traffic of a typical France Telecom network link. During the night, traffic is at its lowest, and rush hours are in the afternoon. We can also see that by using five time periods denoted D1 to D5, we can have a close approximated model of the traffic. We use these five time periods in this thesis. In the case of unexpected traffic or devices failures, we could consider re-activating all devices in the network if the traffic exceeds a certain threshold.

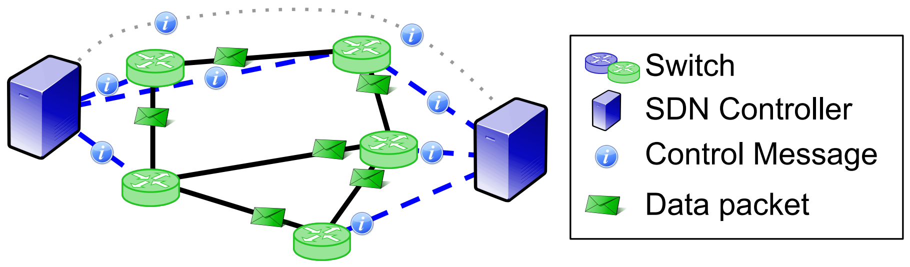
1.3 Software Defined Networks and Network Function Virtualization

The distributed management of legacy networks holds back the deployment of network-wide green policies. Indeed, designing effective green policies requires a centralized management of the network. Moreover, current network hardware are hard to configure on the fly, making hard any attempt to design dynamic policies without manual input. Virtualization and softwarization of the network, brought by the SDN and NFV paradigms, bear the promise of simplifying the management of the network by bringing abstraction and programmability of the underlying network. These two technologies would give network operators more freedom in deploying network-wide dynamic policies, including energy efficiency.

1.3.1 Software Defined Networks



(a) Legacy network



(b) SDN network

Figure 1.4: Differences between legacy and SDN networks . Blue represents the control plane and green the data plane.

The Software Defined Network paradigm is an emerging paradigm that proposes an alternative to the current legacy networks, by decoupling the data and the control plane. This new paradigm is promoted by the Open Network Foundation (ONF), created in 2011 and composed of companies such as Google, Verizon, or Facebook. Google presented their first fully operational SDN network, B4 [Jai+13], and showed that they could use their data center network at full capacity leveraging SDN centralized capability. This accomplishment demonstrated that the SDN paradigm is not just a fad and can undoubtedly bring something new in the way networks are built and managed.

Figure 1.4 illustrates the differences between legacy and SDN networks. In legacy networks, routers forward packets and also exchange control messages with other routers of the topology to take local routing decisions. In SDN networks, routers[1] are relegated to simple forwarding devices, and one or more controllers manage the control plane. As the management of the control plane is given to the controllers, routers must report to them information about the state of the network. This emerging network architecture raises new challenges and opportunities, as discussed in [Kre+15]. We now present a general view of SDN and some challenges of its deployment in large scale networks.

The use of a centralized control plane introduces concerns such as scalability, protection, and security. In legacy networks, routers share the load of the control plane, and each router only communicates with nearby devices. An SDN controller, however, needs recurrent updates from the routers to keep an accurate view of the network. For large networks, having a single controller might thus not be sufficient. Indeed, it may become overloaded due to control messages from the switches, and the end-to-end delay with routers may be too high. In this case, more than one controllers are necessary to manage the control plane, leading again to a distributed control of the network which brings new difficulties. Moreover, in addition to router and link failures, controller reliability is also an important problem for the deployment of SDN.

In particular, controller locations also plays an important part in network performance and reliability. When placing controllers in an SDN network, we need to consider its possible load, the distance to the router under its management and in the case of resiliency, the possible failure in the network. The placement of controller inside a network has been studied in works such as [HSM12; Lan+15]. When we consider the resiliency of the network, we also need to consider the possibility of failures of other controllers and the possible impact of such failures on the remaining ones. Indeed, SDN-capable devices can be configured to contact other controllers in case of failure. Resiliency for controller placement solutions can be found in [Hoc+13; GB13].

We also need to provide a way for all the controllers to exchange control messages to keep a unified view of the network. For control messages, an out-of-band network can assure the communications between the controllers but more frequently is done in-band, i.e., in the same network as the data plane, since it presents smaller Operational Expenditure (OpEx) and Capital Expenditure (CaPex). Moreover, consistency problems of the network view

[1]In this thesis, we use the terms router and switch interchangeably to designate SDN capable forwarding devices.

arise as controllers must share information to keep a consistent network view with each other. Several works already propose protocols for a distributed control plane in SDN networks [JCG14].

Finally, a whole pan of the research on SDN network focuses on the security of communications, between controllers or between the switches and the controller, see, e.g., [SOS13].

We do not, however, consider these problems in this thesis.

The OpenFlow protocol was born from the work of McKeown et al. in Stanford to bring programmability to network devices, before the SDN acronym was coined [McK+08]. It brought the tool to nurture the creation of the SDN paradigm. by providing a way for communication between routers and controllers. The routers use this protocol to report data to the controllers (e.g., link state, traffic). Since routers no longer compute decisions, they must request the action(s) to follow to the controller when they encounter traffic they have no rules to match it against. An analogy to CPU programming language can be made between OpenFlow and assembly language such as x86. Indeed, OpenFlow provides a basic set of commands to change the state of the SDN devices on the network in the same way that assembly provides basic CPU operations. Although out of the scope of this thesis, it is worth mentioning that efforts like [Bos+14] propose higher level programming languages to ease the development of network application by network operators, in the same way that high level language such as C++, Python and Java exists.

In OpenFlow 1.3, packets are matched on up to 40 fields such as source, destination addresses or Type of Service (ToS) field. Forwarding rules are no longer destination based as in legacy networks, but they are flow based. SDN-capable forwarding devices implement the more sophisticated OpenFlow rules using Ternary Content-Addressable Memories. This type of memory can, like regular Content-Addressable Memory (CAM), match bit to 0 or 1, but it can also match using "don't care" bit that can either match a 0 or a 1. The flow granularity possible with SDN comes at a price since Ternary Content-Addressable Memory (TCAM) is power hungry, more expensive and take more space in the router chassis. Due to these technical constraints, and by considering the higher complexity of OpenFlow rules, the size of a forwarding table in SDN equipment is limited (in the order of 1000 rules). Software rules can be used to increase this limit at the cost of weaker performances [Ryg+16]. We explore these constraints in Part I as well as the software-hardware trade off.

1.3.2 Hybrid networks

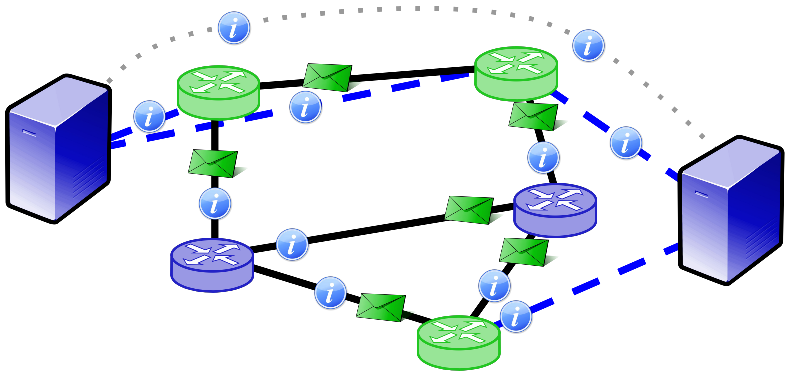


Figure 1.5: Example of an SDN-legacy network. Blue switches represent legacy devices.

Network operators might be reluctant to shelve their legacy equipment and their well known protocols for SDN devices and protocols. Progressive migration might, however, be a more realistic scenario in which legacy and SDN hardware and protocols would coexist.

New challenges arise in regards to the coexistence of legacy and SDN equipment, and most of the SDN solutions found in the literature might not be applicable in this case. Among opportunities and challenges for hybrid networks, Vissichio et al. [VVB14] present four migration scenario that considers mixing up SDN and legacy equipment. The first scenario considers a topology based migration in which whole sub-domains might be upgraded to SDN capable devices. This kind of migration is the most straightforward scenario, interactions between legacy, and SDN protocols are limited. However, in *Service* or *Class*-based migration, both paradigms coexist in the same network as shown in Figure 1.5. In both type of migration, the traffic is divided into groups (by Service or Classes, respectively) and each paradigm is in charge of a set of groups. SDN controllers must implement the legacy protocols used by the legacy devices installed in the network. SDN switches still report to their respective controllers as in a fully SDN network. Moreover, they forward legacy control messages to the controller, which is in charge of

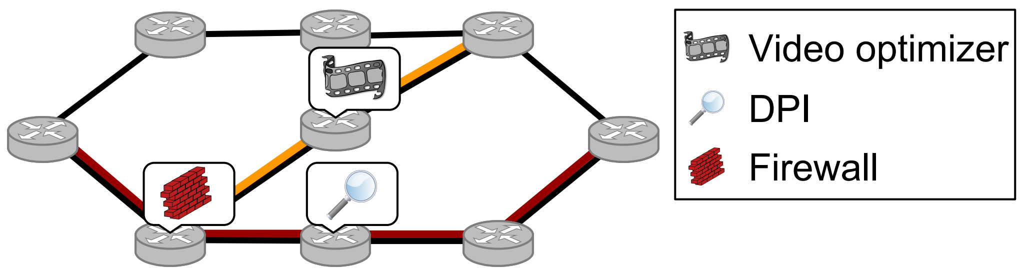


Figure 1.6: Example of Service Function Chains in a network.

answering in place of the SDN switches. Some works already leverage hybrid network capabilities for Traffic Engineering (TE) [AKL13] or robustness [Chu+15] for these types of SDN-OSPF hybrid networks. We consider such networks for Energy Aware Routing in Chapter 6.

1.3.3 Network Function Virtualization

The deployment of network services requires different network functions such as firewall, DPI, or video optimization. In legacy networks, these functions are executed by specialized hardware devices called middleboxes. These devices are hard to migrate throughout the network, limiting network operators ability to adapt to traffic and to design new services.

Virtualization of network functions, pushed by the NFV initiative [Gem+16], brings flexibility in the management of network functions. Functions can now be executed in VMs on general hardware that can be easily moved on the network. A server might run a firewall function at one point, and the same server might be running a video optimization function the next hour.

As the functions require a lot of different virtual resources (bandwidth, CPU cores, memory), NFV Management & Orchestration (MANO) is at the heart of the paradigm. It comprises Virtual Network Functions (VNFs) provisioning, as well as the management of the underlying structure (network and servers). An orchestrator is in charge of assigning locations to the function and allocating the necessary resources to satisfy the requests in the network.

Services are composed of a chain of network functions to apply in a particular order (partial or total), known as Service Function Chains and defined in [HP15]. We present, in Figure 1.6, examples of Service Function Chains applied to two different flows. The red flow subscribed to a service that requires the execution of a firewall function, followed by a DPI function. The second flow's service also requires the execution of a firewall but followed by the execution of a video optimizer. As we will see in Part III, the difficulty of finding the locations of the functions is significantly increased by the order requirement.

Like the SDN paradigm, the NFV one still has a lot to prove regarding scalability, resilience, reliability, and security, see, e.g., [Mij+16]. We can draw parallels between the two paradigms. Indeed, while SDN offers to decouple data and control plane, NFV offers to decouple functions from hardware. Even though virtual switches [Pfa+15] can execute SDN forwarding functions, performances still favor the use of hardware switches.

1.3.4 Using Software Defined Network and Network Function Virtualization for Energy Aware Routing

By enabling network abstraction and programmability, the virtualization and softwarization of the network seem promising in enabling network-wide green policies. The controllers can decide the subset of network devices to shut down using the metrology data collected using the OpenFlow protocols. The controllers can decide which subset of equipment can be put into power-save mode and reconfigure the whole network to change the forwarding paths. Multiple works already addressed the deployment of green solutions using SDN networks such as [ARM17; Wan+14; MTT14; Rah+16; LSC14]. However, few of them also considers the challenge brought by this new paradigm.

In this thesis, we focus our attention on the constraints brought by this emerging paradigm and their impact on the implementation of green policies.

For example, EAR intrinsically increases the number of links that the demands have to go through by turning down equipment, increasing the total number of required rules in the network. As previously stated, SDN table size is limited, and thus EAR might overload TCAMs. This problem is considered in Part I.

As stated, centralized control brought by SDN eases the deployment of green policies on the network. Equipment can be shut down, and the traffic can be rerouted on the remaining devices. However, in hybrid networks, this behavior might be detected as failures by legacy protocols such as Open Shortest Path First (OSPF) [Moy98]. For example, OSPF routers regularly exchange *Hello* packets with neighbors to know if the link is still functional. After a predefined interval without responses to these packets, the OSPF device declares a failure occurred on the link. We address this issue in Part II.

Finally, network functions executed by middleboxes impose hard constraints on the routing of requests in the network. The rigidity of these devices can severely impact the efficiency of energy saving solutions. NFV addresses these limitations and enables the deployment of network-wide energy policies. Network functions can be clustered on a subset of servers while the rest are shut down. The main issue with function placement comes from the way services are built. Each service comprises a set of ordered functions to apply to the corresponding requests. We thus have to consider this ordering when placing the function and make sure that requests are forwarded to the node in the right order. This is the problem we consider in Part III and we extend it to EAR.

1.4 Research Methodology

We now present the methodology used in this section. We first present the metrics studied for the evaluation of our EAR solution and then briefly present the techniques used to tackle the different problems explored in this thesis.

1.4.1 Metrics studied

The principal metric studied is, of course, the energy savings provided by our solutions. However, energy savings impact the network performances regarding the end-to-end delay, link's load or even packet losses.

Delay

By nature, Energy Aware Routing increases the length of the paths in the network by removing some shortest paths. It is thus important to study the delay of the path provided by an EAR solution to check that communications are not affected by a too substantial end-to-end delay. We consider the hop count of the paths (number of links) as well as the delay in milliseconds.

We check that delays are below the usual 50 ms required by Service Level Agreements (SLAs).

Link load

By reducing the subset of active equipment in the network, we funnel the traffic on a set of links leading to more loaded links. In the case of failure or sudden increase activity due to unforeseen events, some links might become overloaded, resulting in a degradation of the network performances due to packet losses.

Packet losses

Using Mininet [Min] emulation and our *SDN testbed*, we were also able to study the packet losses in addition to the packet delay. This metric is of particular importance in hybrid networks where the quality of the coordination between legacy and SDN plays a big part in the correct lifetime of packets.

1.4.2 Technique used

In this thesis, we used different techniques to tackle the EAR problem and the inherent constraints of the virtualization of networks. The three main methods used are the following:

- Integer Linear Programming is used to model the different problems encountered and then used to validated the quality of the solutions provided in Parts I to III.
- Greedy algorithms are used from Parts I to III.
- Finally, in Part III, we used the Column Generation decomposition model.

We further described these techniques in Chapter 2.

1.5 Thesis plan and Contributions

In this section, we go over the plan of this thesis and present the main contribution of each part. We also provided the list of publications of all works.

In Chapter 2, we first present to the reader basis on Integer Linear Programming, as this mathematical framework is widely used in this thesis. We also introduce the concept of the column generation decomposition model, used in Part III. Finally, we present the concept of Greedy Algorithm also widely used throughout this work.

We first tackle, in Part I, the table size constraints imposed by the TCAM used by SDN capable equipment. We first study, in Chapter 3, the use of aggregation rules to reduce the size of the table. We consider two types of compression (i) only using the default rule and (ii) using aggregation on source and destination in addition to the default rule. We propose an Integer Linear Program (ILP) formulation when wildcard rules are used, as well as heuristic and approximation algorithms. We validated them using random forwarding and using network table obtained from SNDlib instances [Orl+10].

We then study the *Compression Problem* in the context of Energy Aware Routing and formulate the Energy Aware Routing with Compression (EARC) problem. We propose ILP formulations for the problem for both default and wildcard rules and propose an efficient heuristic using the compression heuristics proposed in Chapter 3. We show that using wildcard rules, we can provide energy savings close to the classic EAR scenario where tables have no size constraints.

Finally, in Chapter 5, we collaborated with members of the SigNet team to look at the table constraints on an *SDN testbed*, including an HP5412zl SDN switch. We focus our study on data centers topologies. We propose MINNIE, a dynamic non-energy aware variant of the heuristic presented in Chapter 4. We also show that entry size limit can be attained with a few numbers of clients. Using the 3-approximation algorithm presented in Chapter 3, we can deploy up to 3000 servers with up to a million flows with a maximum capacity of 1000 rules of each router, with no noticeable increase in packet delay. We also compare the performance of hardware and software rules and show that similarly to [Ryg+16], even though software rules can greatly increase router table capacities, their matching performance is far from rules implemented in TCAM.

All of these results can be found in the following publications [Rif+17; Rif+16; Gir+16; Hav+15; Rif+15].

The second part of the thesis focuses on the deployment of energy policies in networks in which SDN capable equipment and legacy devices coexist. The main issue of EAR in hybrid networks is that turning off SDN capable equipment is detected as a failure by legacy protocols. Inspired by [Chu+15], we use backup tunnels to reroute traffic when disabling links. To prevent packet losses due to the detection time of a disabled link by legacy protocols, we also propose a smooth extinction mechanism. First, we model the Energy Aware Routing in a hybrid SDN network (hEAR) with tunnels problem as an ILP. As the tunnel selection increases the difficulty of the problem, we then propose an efficient heuristic to shutdown links and routers and satisfy all requests in the network. Combined with the use of tunnels, smooth link extinction, and failure/flash crowd detection mechanism, this comprises Smooth ENergy Aware Routing (SENAtoR). We show, using a Mininet testbed, that our solution enables EAR in a hybrid network without introducing additional packet losses thanks to our smooth link extinction.

Results are under submission and can be found in [Hui+17a].

Function Virtualization for Energy Aware Routing is studied in Part III. In Chapter 7, we first propose an optimal scalable model to the Service Function Chain Placement using *Column Generation*, called NFV_CG. It can solve instances on networks with up to 50 nodes and about 10 000 requests in just over a minute. We also study the trade-off between the number of NFV capable hosts and bandwidth usage in the network. We show that adding more NFV nodes greatly decreases the bandwidth up to 50% of the network. We observe diminishing returns when more than 50% of the network is NFV-capable. We then build upon this model to consider two variants of the problem.

In the first variant, we consider licenses cost constraints, i.e., limits on the number of replicas of a function that can be deployed in the network. These restrictions significantly increase the difficulty of the problem. We thus propose a two-phase heuristic that first chooses the location of the function and then use NFV_CG to route the requests.

The second variant, studied in Chapter 8, corresponds to the EAR problem with Service Function Chain (SFC) constraints. Using a baseline scenario of a legacy network (No SDN + middleboxes), we study the possible energy savings done with only SDN (and middleboxes) and a SDN/NFV scenario. We show that while SDN saves between 18 and 51% of energy during the night, the use of VNF provides 4 to 12% more energy savings.

Results of this part can be found in [HJG17; Hui+17b]

Finally, we studied structured overlay for live video streaming Peer-to-peer (P2P) systems. Since these works are far from the scope of this thesis, we provide them in Appendices A and B.

These results can be found in [GH15; GH16].

1.6 List of publications of this thesis

We now list the publications included in this thesis.

International journals

[Rif+17] Myriana Rifai, Nicolas Huin, Christelle Caillouet, Frederic Giroire, Joanna Moulierac, Dino Lopez Pacheco, and Guillaume Urvoy-Keller. "Minnie: An SDN world with few compressed forwarding rules". In: *Computer Networks* 121 (2017), pp. 185–207 (cit. on p. 17).

Submissions to international journals

[Gir+] Frédéric Giroire, Nicolas Huin, Joanna Moulierac, and Truong Khoa Phan. "Energy-aware Routing in Software-Defined Network using Compression".

[GHT] Frédéric Giroire, Nicolas Huin, and Andrea Tomassilli. "The Structured Way of Dealing with Heterogeneous Live Streaming Systems".

[HJG] Nicolas Huin, Brigitte Jaumard, and Frédéric Giroire. "Optimal Network Service Chain Provisioning".

[Hui+] Nicolas Huin, Andrea Tomassilli, Frédéric Giroire, and Brigitte Jaumard. "Energy-Efficient Service Function Chain Provisioning".

International conferences

[GH15] Frederic Giroire and Nicolas Huin. “Study of Repair Protocols for Live Video Streaming Distributed Systems”. In: *2015 IEEE Global Communications Conference (GLOBECOM)* (Dec. 2015) (cit. on p. 19).

[Rif+15] M. Rifai, N. Huin, C. Caillouet, F. Giroire, D. Lopez-Pacheco, J. Moulierac, and G. Urvoy-Keller. “Too Many SDN Rules? Compress Them with MINNIE”. In: *2015 IEEE Global Communications Conference (GLOBECOM)* (Dec. 2015) (cit. on p. 17).

[HJG17] Nicolas Huin, Brigitte Jaumard, and Frédéric Giroire. “Optimization of Network Service Chain Provisioning”. In: *IEEE International Conference on Communications 2017*. Paris, France, May 2017 (cit. on p. 18).

[Hui+17a] Nicolas Huin, Myriana Rifai, Frédéric Giroire, Dino Martı'n Lopez Pacheco, Guillaume Urvoy-Keller, and Joanna Moulierac. “Bringing Energy Aware Routing closer to Reality with SDN Hybrid Networks”. In: *2017 IEEE Global Communications Conference (GLOBECOM)*. 2017.

[Hui+17c] Nicolas Huin, Andrea Tomassilli, Frédéric Giroire, and Brigitte Jaumard. “Energy-Efficient Service Function Chain Provisioning”. In: *International Network Optimization Conference 2017*. Lisbon, Portugal, Feb. 2017 (cit. on p. 18).

National conferences

[Hav+15] Frédéric Havet, Nicolas Huin, Joanna Moulierac, and Khoa Phan. “Routage vert et compression de règles SDN”. In: *ALGOTEL 2015 - 17èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*. Beaune, France, June 2015 (cit. on p. 17).

[GH16] Frédéric Giroire and Nicolas Huin. “Étude d’un système distribué de diffusion de vidéo en direct”. In: *ALGOTEL 2016 - 18èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*. 2016 (cit. on p. 19).

[Rif+16] Myriana Rifai, Nicolas Huin, Christelle Caillouet, Frédéric Giroire, Joanna Moulierac, Dino Martı'n Lopez Pacheco, and Guillaume Urvoy-Keller. "MINNIE: enfin un monde SDN sans (trop de) règles". In: *ALGOTEL 2016 - 18èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications.* 2016 (cit. on p. 17).

Research report

[Gir+16] F. Giroire, N. Huin, J. Moulierac, and K. Phan. *Optimizing Rule Placement in Software-Defined Networks for Energy-aware Routing.* Research Report. Inria, 2016 (cit. on p. 17).

[Hui+17b] Nicolas Huin, Myriana Rifai, Frédéric Giroire, Dino Lopez Pacheco, Guillaume Urvoy-Keller, and Joanna Moulierac. *Bringing Energy Aware Routing closer to Reality with SDN Hybrid Networks.* Research Report RR-9020. INRIA Sophia Antipolis - I3S, Jan. 2017 (cit. on p. 18).

Bibliography

[Van+14] Ward Van Heddeghem, Sofie Lambert, Bart Lannoo, Didier Colle, Mario Pickavet, and Piet Demeester. “Trends in worldwide ICT electricity consumption from 2007 to 2012”. In: *Computer Communications* 50 (2014), pp. 64–76 (cit. on p. 1).

[Cha+08] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. “Power Awareness in Network Design and Routing”. In: *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*. Apr. 2008 (cit. on pp. 1, 3, 4).

[Abt+10] Dennis Abts, Michael R. Marty, Philip M. Wells, Peter Klausler, and Hong Liu. “Energy proportional datacenter networks”. In: *ACM SIGARCH Computer Architecture News* 38.3 (June 2010), p. 338 (cit. on pp. 1, 3).

[LLW11] C. C. Lin, P. Liu, and J. J. Wu. “Energy-Aware Virtual Machine Dynamic Provision and Scheduling for Cloud Computing”. In: *2011 IEEE 4th International Conference on Cloud Computing*. July 2011, pp. 736–737. DOI: 10.1109/CLOUD.2011.94 (cit. on p. 1).

[Zho+09] Sheng Zhou, Jie Gong, Zexi Yang, Zhisheng Niu, and Peng Yang. “Green mobile access network with dynamic base station energy saving”. In: *ACM MobiCom*. Vol. 9. 262. 2009, pp. 10–12 (cit. on p. 1).

[McK+08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. “OpenFlow: Enabling Innovation in Campus Networks”. In: *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), pp. 69–74 (cit. on pp. 2, 11).

[Gem+16] Aaron Gember, Prathmesh Prabhu, Zainab Ghadiyali, and Aditya Akella. “Toward software-defined middlebox networking”. In: *ACM Workshop on Hot Topics in Networks (HotNets)*. 2016, pp. 7–12 (cit. on pp. 2, 13).

[GS03] Maruti Gupta and Suresh Singh. “Greening of the Internet”. In: *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications.* ACM. 2003, pp. 19–26 (cit. on p. 3).

[Chr+10] Ken Christensen, Pedro Reviriego, Bruce Nordman, Michael Bennett, Mehrgan Mostowfi, and Juan Maestro. “IEEE 802.3az: the road to energy efficient ethernet”. In: *IEEE Communications Magazine* 48.11 (Nov. 2010), pp. 50–56. ISSN: 0163-6804. DOI: 10.1109/mcom.2010.5621967 (cit. on p. 4).

[Hau15] Timothée Haudebourd. “Analyse des principales techniques d’économie d’énergie en réseau filaire Ethernet”. MA thesis. Université de Rennes 1 - ENS Rennes, June 2015 (cit. on p. 4).

[CMN09] Luca Chiaraviglio, Marco Mellia, and Fabio Neri. “Energy-aware backbone networks: a case study”. In: *IEEE ICC*. 2009 (cit. on p. 6).

[CMN12] L. Chiaraviglio, M. Mellia, and F. Neri. “Minimizing ISP Network Energy Cost: Formulation and Solutions”. In: *IEEE/ACM Transactions on Networking* 20.2 (Apr. 2012), pp. 463–476. ISSN: 1558-2566. DOI: 10.1109/tnet.2011.2161487. URL: http://dx.doi.org/10.1109/TNET.2011.2161487 (cit. on p. 6).

[Pha14] Truong Khoa Phan. “Conception et gestion de réseaux efficaces en énergie”. PhD thesis. Nice, 2014 (cit. on p. 6).

[Rou+13] E. Le Rouzic, E. Bonetto, L. Chiaraviglio, F. Giroire, F. Idzikowski, F. Jiménez, C. Lange, J. Montalvo, F. Musumeci, I. Tahiri, A. Valenti, W. Van Heddeghem, Y. Ye, A. Bianco, and A. Pattavina. “TREND towards more energy-efficient optical networks”. In: *2013 17th International Conference on Optical Networking Design and Modeling (ONDM)*. Apr. 2013, pp. 211–216 (cit. on p. 6).

[SLX10] Yunfei Shang, Dan Li, and Mingwei Xu. “Energy-aware routing in data center network”. In: *Proceedings of the first ACM SIGCOMM workshop on Green networking.* ACM. 2010, pp. 1–8 (cit. on p. 6).

[DGF12] Floriano De Rango, Francesca Guerriero, and Peppino Fazio. “Link-stability and energy aware routing protocol in distributed wireless networks”. In: *IEEE Transactions on Parallel and Distributed systems* 23.4 (2012) (cit. on p. 6).

[R B+11] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti. “Energy Efficiency in the Future Internet: A Survey of Existing Approaches and Trends in Energy-Aware Fixed Network Infrastructures”. In: *IEEE Communication Surveys and Tutorials*. 2011 (cit. on p. 6).

[Cia+12] A. Cianfrani, V. Eramo, M. Listanti, M. Polverini, and A. V. Vasilakos. “An OSPF-Integrated Routing Strategy for QoS-Aware Energy Saving in IP Backbone Networks”. In: *IEEE Transactions on Network and Service Management* 9.3 (Sept. 2012), pp. 254–267 (cit. on p. 6).

[OSP13] R. d. O. Schmidt, R. Sadre, and A. Pras. “Gaussian traffic revisited”. In: *2013 IFIP Networking Conference*. May 2013, pp. 1–9 (cit. on p. 8).

[Ara+16] Julio Araujo, Frédéric Giroire, Joanna Moulierac, Yaning Liu, and Modrzejewski Remigiusz. “Energy Efficient Content Distribution”. In: *Computer Journal* (2016), pp. 1–18 (cit. on p. 8).

[Jai+13] Sushant Jain, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, Amin Vahdat, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, and et al. “B4: experience with a globally-deployed software defined wan”. In: *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM - SIGCOMM ’13* (2013) (cit. on p. 9).

[Kre+15] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. “Software-defined networking: A comprehensive survey”. In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76 (cit. on p. 10).

[HSM12] Brandon Heller, Rob Sherwood, and Nick McKeown. “The controller placement problem”. In: *Proceedings of the first workshop on Hot topics in software defined networks*. ACM. 2012, pp. 7–12 (cit. on p. 10).

[Lan+15] Stanislav Lange, Steffen Gebert, Thomas Zinner, Phuoc Tran-Gia, David Hock, Michael Jarschel, and Marco Hoffmann. "Heuristic approaches to the controller placement problem in large scale SDN networks". In: *IEEE Transactions on Network and Service Management* 12.1 (2015), pp. 4–17 (cit. on p. 10).

[Hoc+13] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia. "Pareto-optimal resilient controller placement in SDN-based core networks". In: *Proceedings of the 2013 25th International Teletraffic Congress (ITC).* Sept. 2013, pp. 1–9 (cit. on p. 10).

[GB13] Minzhe Guo and Prabir Bhattacharya. "Controller placement for improving resilience of software-defined networks". In: *Networking and Distributed Computing (ICNDC), 2013 Fourth International Conference on.* IEEE. 2013, pp. 23–27 (cit. on p. 10).

[JCG14] Y. Jimenez, C. Cervello-Pastor, and A.J. Garcia. "On the controller placement for designing a distributed SDN control layer". In: *IFIP Networking Conference.* 2014, pp. 1–9 (cit. on p. 11).

[SOS13] Sandra Scott-Hayward, Gemma O'Callaghan, and Sakir Sezer. "SDN security: A survey". In: *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For.* IEEE. 2013, pp. 1–7 (cit. on p. 11).

[Bos+14] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. "P4: Programming Protocol-independent Packet Processors". In: *SIGCOMM Comput. Commun. Rev.* 44.3 (July 2014), pp. 87–95 (cit. on p. 11).

[Ryg+16] Piotr Rygielski, Marian Seliuchenko, Samuel Kounev, and Mykhailo Klymash. "Performance Analysis of SDN Switches with Hardware and Software Flow Tables". In: *Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools (ValueTools 2016).* 2016 (cit. on pp. 11, 17).

[VVB14] Stefano Vissicchio, Laurent Vanbever, and Olivier Bonaventure. “Opportunities and research challenges of hybrid software defined networks”. In: *ACM SIGCOMM Computer Communication Review* 44.2 (Apr. 2014), pp. 70–75. DOI: 10.1145/2602204.2602216 (cit. on p. 12).

[AKL13] Sugam Agarwal, Murali Kodialam, and TV Lakshman. “Traffic engineering in software defined networks”. In: *INFOCOM, 2013 Proceedings IEEE*. IEEE. 2013, pp. 2211–2219 (cit. on p. 13).

[Chu+15] Cing-Yu Chu, Kang Xi, Min Luo, and H Jonathan Chao. “Congestion-aware single link failure recovery in hybrid SDN networks”. In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE. 2015, pp. 1086–1094 (cit. on pp. 13, 18).

[HP15] J. Halpern and C. Pignataro. *Service Function Chaining (SFC) Architecture*. RFC 7665. RFC Editor, Oct. 2015. DOI: 10.17487/RFC7665 (cit. on p. 14).

[Mij+16] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. “Network function virtualization: State-of-the-art and research challenges”. In: *IEEE Communications Surveys & Tutorials* 18.1 (2016), pp. 236–262 (cit. on p. 14).

[Pfa+15] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. “The Design and Implementation of Open vSwitch”. In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 117–130 (cit. on p. 14).

[ARM17] Mohamad Khattar Awad, Yousef Rafique, and Rym A. M’Hallah. “Energy-aware routing for software-defined networks with discrete link rates: A benders decomposition-based heuristic approach”. In: *Sustainable Computing: Informatics and Systems* 13 (Mar. 2017), pp. 31–41 (cit. on p. 14).

[Wan+14] Rui Wang, Zhipeng Jiang, Suixiang Gao, Wenguo Yang, Yinben Xia, and Mingming Zhu. "Energy-aware routing algorithms in Software-Defined Networks". In: *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014* (June 2014) (cit. on p. 14).

[MTT14] Adam Markiewicz, Phuong Nga Tran, and Andreas Timm-Giel. "Energy consumption optimization for software defined networks considering dynamic traffic". In: *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)* (Oct. 2014) (cit. on p. 14).

[Rah+16] Mahshid Rahnamay-Naeini, Sonali Sen Baidya, Ehsan Siavashi, and Nasir Ghani. "A traffic and resource-aware energy-saving mechanism in software defined networks". In: *2016 International Conference on Computing, Networking and Communications (ICNC)* (Feb. 2016) (cit. on p. 14).

[LSC14] Dan Li, Yunfei Shang, and Congjie Chen. "Software defined green data center network with exclusive routing". In: *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. IEEE, Apr. 2014 (cit. on p. 14).

[Moy98] J. Moy. *OSPF Version 2*. RFC 2328. Ascend Communications, Inc., Apr. 1998. URL: https://www.rfc-editor.org/rfc/rfc2328.txt (cit. on p. 15).

[Min] Team Mininet. *Mininet*. http://mininet.org/ (cit. on p. 16).

[Orl+10] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. "SNDlib 1.0–Survivable Network Design Library". In: *Networks* 55.3 (2010), pp. 276–286 (cit. on p. 17).

[Rif+17] Myriana Rifai, Nicolas Huin, Christelle Caillouet, Frederic Giroire, Joanna Moulierac, Dino Lopez Pacheco, and Guillaume Urvoy-Keller. "Minnie: An SDN world with few compressed forwarding rules". In: *Computer Networks* 121 (2017), pp. 185–207 (cit. on p. 17).

[Rif+16] Myriana Rifai, Nicolas Huin, Christelle Caillouet, Frédéric Giroire, Joanna Moulierac, Dino Martı'n Lopez Pacheco, and Guillaume Urvoy-Keller. "MINNIE: enfin un monde SDN sans (trop de) règles". In: *ALGOTEL 2016 - 18èmes Rencontres*

Francophones sur les Aspects Algorithmiques des Télécommunications. 2016 (cit. on p. 17).

[Gir+16] F. Giroire, N. Huin, J. Moulierac, and K. Phan. *Optimizing Rule Placement in Software-Defined Networks for Energy-aware Routing.* Research Report. Inria, 2016 (cit. on p. 17).

[Hav+15] Frédéric Havet, Nicolas Huin, Joanna Moulierac, and Khoa Phan. "Routage vert et compression de règles SDN". In: *ALGOTEL 2015 - 17èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications.* Beaune, France, June 2015 (cit. on p. 17).

[Rif+15] M. Rifai, N. Huin, C. Caillouet, F. Giroire, D. Lopez-Pacheco, J. Moulierac, and G. Urvoy-Keller. "Too Many SDN Rules? Compress Them with MINNIE". In: *2015 IEEE Global Communications Conference (GLOBECOM)* (Dec. 2015) (cit. on p. 17).

[Hui+17a] Nicolas Huin, Myriana Rifai, Frédéric Giroire, Dino Lopez Pacheco, Guillaume Urvoy-Keller, and Joanna Moulierac. *Bringing Energy Aware Routing closer to Reality with SDN Hybrid Networks.* Research Report RR-9020. INRIA Sophia Antipolis - I3S, Jan. 2017 (cit. on p. 18).

[HJG17] Nicolas Huin, Brigitte Jaumard, and Frédéric Giroire. "Optimization of Network Service Chain Provisioning". In: *IEEE International Conference on Communications 2017.* Paris, France, May 2017 (cit. on p. 18).

[Hui+17b] Nicolas Huin, Andrea Tomassilli, Frédéric Giroire, and Brigitte Jaumard. "Energy-Efficient Service Function Chain Provisioning". In: *International Network Optimization Conference 2017.* Lisbon, Portugal, Feb. 2017 (cit. on p. 18).

[GH15] Frederic Giroire and Nicolas Huin. "Study of Repair Protocols for Live Video Streaming Distributed Systems". In: *2015 IEEE Global Communications Conference (GLOBECOM)* (Dec. 2015) (cit. on p. 19).

[GH16] Frédéric Giroire and Nicolas Huin. "Étude d'un système distribué de diffusion de vidéo en direct". In: *ALGOTEL 2016 - 18èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications.* 2016 (cit. on p. 19).

[Gir+] Frédéric Giroire, Nicolas Huin, Joanna Moulierac, and Truong Khoa Phan. “Energy-aware Routing in Software-Defined Network using Compression”.

[HJG] Nicolas Huin, Brigitte Jaumard, and Frédéric Giroire. “Optimal Network Service Chain Provisioning”.

[Hui+] Nicolas Huin, Andrea Tomassilli, Frédéric Giroire, and Brigitte Jaumard. “Energy-Efficient Service Function Chain Provisioning”.

[GHT] Frédéric Giroire, Nicolas Huin, and Andrea Tomassilli. “The Structured Way of Dealing with Heterogeneous Live Streaming Systems”.

[Hui+17c] Nicolas Huin, Myriana Rifai, Frédéric Giroire, Dino Martı’n Lopez Pacheco, Guillaume Urvoy-Keller, and Joanna Moulierac. “Bringing Energy Aware Routing closer to Reality with SDN Hybrid Networks”. In: *2017 IEEE Global Communications Conference (GLOBECOM)*. 2017.

Chapter 2

Preliminaries

Contents

In this chapter, we present the basis of the different techniques used throughout this thesis. Three main techniques are used: Linear Programming, Column Generation, and Greedy Algorithm. For all of them, we provide a brief introduction and provide an example applied to the Energy Aware Routing (EAR) problem.

2.1 Linear Programming

Linear Programming is a powerful mathematical framework used to solve optimization problems (minimization or maximization). It comprises a function to optimize, called the *objective function* and a set of constraints expressed as (in)equalities. As indicated by the name of the framework, the objective function and the (in)equalities are linear. All linear programs can be written in the following *canonical form*:

$$\begin{aligned} &\min c^T x \\ \text{s.t. } &Ax \geq b \\ &x \geq 0 \end{aligned} \tag{2.1}$$

where x represents the vector of variables to determine, A is a matrix of known coefficients and c and b are vectors of known coefficients.

An important concept of Linear Programming is the concept of *duality*. Every *primal* problem can be transformed into its dual problem. For example, the dual of Problem 2.1 is

$$\begin{aligned} &\max b^T y \\ \text{s.t. } &A^T y \leq c \\ &y \geq 0 \end{aligned} \tag{2.2}$$

where y is the vector of variables of the dual problem. The dual of a dual is the primal problem. Two important duality theorems exist. The weak duality theorem states that the value of the objective value of any feasible solution of the dual is an upper bound on the optimal objective value of the primal solution. The strong duality theorem states that if x^* is the optimal solution of the primal, then there exists an optimal solution of the dual, y^*, and

$$c^T x^* = b^T y^*$$

These theorems are used to provide bounds on the objective value and are exploited by primal-dual algorithms or the column generation decomposition model.

When one or more variables are integers, we talk about Mixed Integer Linear Programming (or Integer Linear Programming when all variables are integers). Linear Programs can be solved easily, depending on their sizes, by state of the art solvers. However, solving Integer Linear Programs (ILPs) and Mixed Integer Linear Programs (MILPs) is known to be NP-Hard. Many exact methods have proposed such as branch-and-bound, cutting planes, column generation or Bender's decomposition. For a more in-depth view of linear programming, we point the reader to [Chv83].

2.1.1 Example: Multi-commodity flow problem

We now present, as an example, the formulation of the multi-commodity flow problem. The MultiCommodity Flow (MCF) problem is a well-known network flow problem where multiple flows request between different sources and destinations, called commodities [AMO93]. More formally, we model a network as a digraph $D = (V, A)$ where each arc $(u, v) \in A$ has a capacity C_{uv}. We have k commodities $S_1, S_2, \ldots, S_k$ defined by $S_i = (s_i, t_i, D_i)$ where s_i and t_i are the source and destination and D_i the charge of the flow. The goal is to find a path for each commodity such that link capacities are respected.

We can easily formulate this problem using Integer Linear Programming or Linear Programming if the flow can be split between multiple paths. We first introduce the set of variables $f^i_{uv} \in [0, 1]$ that represent the fraction of flow going through the link (u, v) for the commodity S_i. If the flows are un-splittable, i.e., a commodity can only be forwarded on exactly one path, we have that $f^i_{uv} \in \{0, 1\}$. We then need to satisfy the following two sets of constraints to have a valid solution.

Flow conservation constraints For a given commodity S_i and a given node $u \in V \setminus \{s_i, t_i\}$, the sum of incoming flows must be equal to the sum of outgoing flows of the commodity. For the node s_i, the difference between the outgoing flows and the incoming must be equal to one, i.e., all flow has to exit the source node. It is reversed for the destination node t_i: the difference between the incoming and the outgoing flows must be equal to D_i, i.e., all flow has to enter the destination node. These constraints are expressed as follows.

$$\sum_{v \in N^+(u)} f^i_{uv} - \sum_{v \in N^-(u)} f^i_{vu} = \begin{cases} 1 & \text{if } u = s_i \\ -1 & \text{if } u = t_i \\ 0 & \text{else} \end{cases} \qquad \forall i \in \{1 \ldots k\}, u \in V \tag{2.3}$$

Link capacity constraints The following set of constraints states that the sum of the flows on a link must not exceed its capacity.

$$\sum_{i=1}^{k} D_i \times f^i_{uv} \leq C_{uv} \qquad \forall (u, v) \in A \tag{2.4}$$

Objective function Multiple objective functions can be considered for the MCF problem. First, we could consider the minimum cost variant of the MCF problem. A cost to forward flows is given to each arc (u, v), denoted Q_{uv}. The objective is to minimize the total cost of forwarding each commodity and is given by

$$\min \sum_{(u,v)\in A} \sum_{i=1}^{k} Q_{uv} \times f_{uv}^{i} \tag{2.5}$$

In the context of Energy Aware Routing, we want to reduce the number of active links in the network. Indeed, the EAR problem consists in routing a set of requests (or commodities) through a network. We could thus use the MCF formulation for the EAR problem. In this case, we need to introduce a new set of variables, $x_{uv} \in \{0, 1\}$, that represent the state of each arc (active or not). We also need to slightly alter the link capacity constraints to forbid forwarding of flows on inactive links. The constraints thus become

$$\sum_{i=1}^{k} D_i \times f_{uv}^{i} \leq C_{uv} \times x_{uv} \qquad \forall (u, v) \in E \tag{2.6}$$

and the objective function is given by

$$\min \sum_{(u,v)\in A} x_{uv} \tag{2.7}$$

We have presented a formulation of the MCF known as the *edge formulation* since we use a variable for each arc and commodity. A second formulation referred to as the *path formulation* exists. It uses, for each commodity, a variable for each path between its end-point. This considerably increases the number of variables of the formulation since the number of paths in a graph is exponential. Nonetheless, using the column generation decomposition model, we can greatly reduce the number of paths considered as we explain in the next section.

2.2 Column Generation

Column generation is an efficient algorithm to solve large *linear programs*. This technique originated from the observation that most variables do not

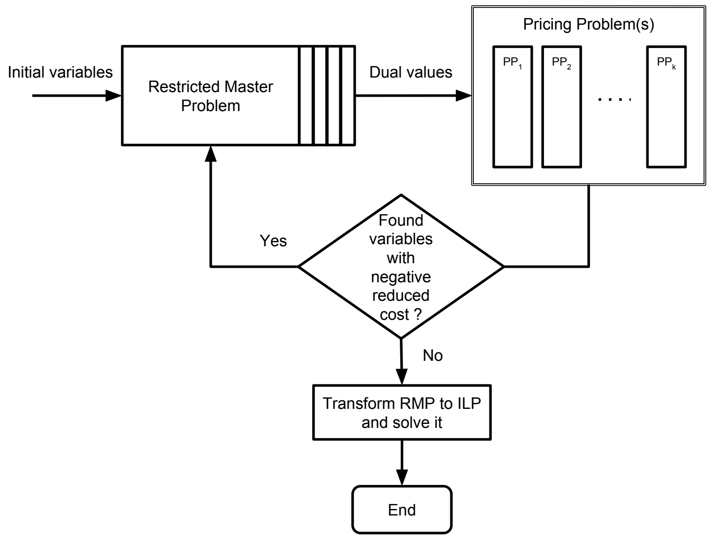

Initial variables
Restricted Master Problem
Dual values
Pricing Problem(s)
PP_1
PP_2
....
PP_k
Yes
Found variables with negative reduced cost ?
No
Transform RMP to ILP and solve it
End

Figure 2.1: Flowchart of the Column Generation at root node algorithm.

belong in the optimal solution. This is particularly the case for *extended formulation* such as the *path formulation* mentioned earlier. We thus consider a restricted set of variables at the start of the algorithm and then find new variables (columns) that can improve the solution.

We present in Figure 2.1 the flow chart of the algorithm. The problem is divided into two problems: the Restricted Master Problem (RMP) and the Pricing Problem (PP). The RMP represents the initial formulation of the problem with a reduced number of variables. These variables are often provided by a heuristic to jump start the process. The PP is in charge of generating the new variables to add in the RMP. Depending on the formulation multiple PPs can exist for the same problem. In this case, they can be solved in parallel to speed up the process. These two problems depend on each other: first, the RMP is solved to obtain the optimal dual value of the problem. The dual values are then provided to the PP to find variables with a *negative reduced cost* (in the case of minimization). The *reduced cost* of a variable represents the value by which the objective function will improve if the variable enters the solution. Going back to the primal formulation given earlier in Equation (2.1), the reduced cost of a variable x_i is given by

$$c_i - (A^T u)_i$$

where u is the dual cost vector obtained from the solution of the RMP. The exchanges between RMP and PP continue until no variables with a negative reduced cost are found. In this case, the objective function value of the RMP is optimal. Note that the algorithm provides optimal solutions for Linear Programs. However, we can still use it to solve some Integer Linear Programs.

If the original problem is an ILP, we consider the relaxation of the problem, i.e., integrality constraints are relaxed. We use the column generation algorithm to optimally solve this relaxed formulation and then transform the RMP back to an ILP. Since we only use a subset of variables, the ILP might not find any feasible solution. This, however, does not mean that the problem is unfeasible. In the case where we obtain an integer solution x_{ILP}, it might not be the optimal solution to the problem. But we can compare it with the lower bound given by the optimal solution x^*_{LP} of the relaxed formulation. For a particular instance, we can gage the quality of the integer solution using the ratio: $\varepsilon = (x_{\text{ILP}} - x^*_{\text{LP}})/x^*_{\text{LP}}$. The closer to 0 the ratio is, the better the solution is and when it is equal to 0, the integer solution x_{ILP} is an optimal solution of the problem. While we only described *Column Generation at the Root node*, this technique can be integrated into the standard branch-and-bound algorithm to solve large ILP optimally. This is known as *branch & price*. In this thesis, we only consider *Column Generation at the Root node.*

2.2.1 Multi-commodity flow: Path formulation

As previously stated the MCF problem can also be formulated using a path formulation. As the name indicates, we use, for each commodity, a variable for every possible path between the source and destination of the commodity. We denote $\mathcal{P}_i$ the set of paths for the commodity i and $z^i_p \in [0, 1]$ the variable that represents the fraction of the flow going through the path $p \in \mathcal{P}_i$. As for the edge formulation, if the flows are un-splittable we have $z^i_p \in \{0, 1\}$. If we want to minimize the number of active links in the network, we formulate the problem as follows:

Objective

$$\min \sum_{(u,v)\in A} x_{uv} \tag{2.8}$$

One path per commodity

$$\sum_{p \in \mathcal{P}_i} z_p^i = 1 \qquad \forall i \in \{1 \dots k\} \tag{2.9}$$

Link capacity constraints

$$\sum_{i=1}^{k} \sum_{p \in \mathcal{P}_i} \delta_{uv}^p D_i \times z_p^i \leq C_{uv} \times x_{uv} \qquad \forall (u,v) \in A \tag{2.10}$$

where $\delta_{uv}^p = 1$ if link $(u,v) \in p$. It is clear that the number of variables is exponential and that no solver will be able to construct the model for large enough instances. Instead, we can start with a small subset of paths for each commodity, e.g., the shortest paths, and use a PP for each commodity to find paths improving the solution. Each PP consists in finding a constrained path with the minimum reduced cost. The reduced cost for a path of a given commodity i is given by

$$\pi_i^{(2.9)} - D_i \times \sum_{(u,v) \in A} \pi_{uv}^{(2.10)} \times \delta_{uv}$$

where $\pi_i^{(2.9)}$ and $\pi_{uv}^{(2.10)}$ correspond to the dual values of their respective constraints in the RMP and δ_{uv} is a variable equal to 1 if the link is (u,v) is in the path. The PP for a commodity i can be formulated as follows, where dual values are given as input.

Objective function.

$$\min \pi_i^{(2.9)} - \sum_{(u,v) \in A} \pi_{uv}^{(2.10)} D_i \times \delta_{uv} \tag{2.11}$$

Flow conservation constraints

$$\sum_{v \in N^+(u)} \delta_{uv} - \sum_{v \in N^-(u)} \delta_{vu} = \begin{cases} 1 \text{ if } u = s_i \\ -1 \text{ if } u = t_i \\ 0 \text{ else} \end{cases} \qquad u \in V \tag{2.12}$$

Link capacity constraints

$$D_i \times \delta_{uv} \leq C_{uv} \qquad (u,v) \in A \tag{2.13}$$

Since we have one PP per commodity, we can parallelize the search of new variables since PPs are independent of each other. This takes advantages of modern CPU architectures that propose multiple processing cores and increases the resolution speed.

2.3 Greedy Algorithms

Greedy algorithms represent the class of algorithms where decisions are taken greedily, and no backtracking is done until a solution is found (see [Cor+01] for further reading). At each step of the algorithm, we use the local optimum to build a solution. The main advantage of these algorithms is that they provide fast and easy to implement algorithms to use in practice.

In general, greedy algorithms have the following components:

- A set of possible candidates to choose from.
- A selection function that selects the best valid candidate to add to the solution.

Some problems can be solved to optimality by greedy algorithms as the shortest path problem using Dijkstra's algorithm or the unweighted interval scheduling problem. In other cases, a greedy algorithm might not give an optimal solution but can provide a good approximation of the optimal solution, e.g., a 2-approximation greedy algorithm exists for the vertex cover problem. In the worst case, greedy algorithms offer no performance guarantee. This is why we use the previously mentioned mathematical tools to compare our greedy algorithms' performances with the optimal solution that they provide. This is the case for the EAR problem, that has been shown in [GMM12] to be NP-Complete, with no polynomial-time constant-factor approximation algorithm existing.

2.3.1 Greedy Energy Aware Routing

The Energy Aware Routing problem is a equivalent to a MCF problem where the goal is to satisfy all requests and minimizing the number of active links

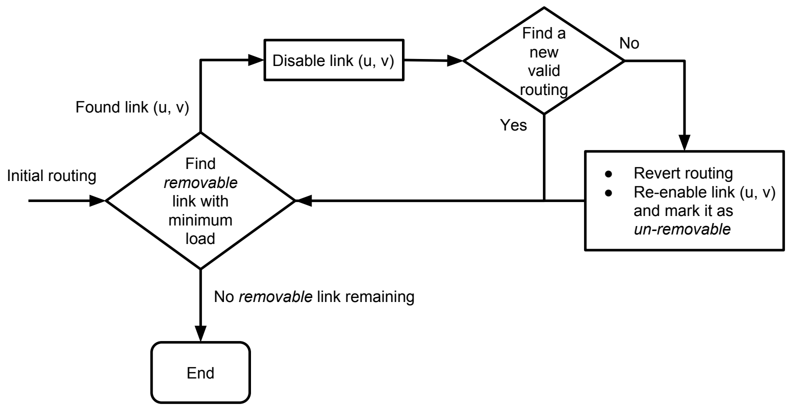

Disable link (u, v)
Find a new valid routing
No
Found link (u, v)
Yes
Initial routing
Find *removable* link with minimum load
Revert routing
Re-enable link (u, v) and mark it as *un-removable*
No *removable* link remaining
End

Figure 2.2: Flow chart of the greedy algorithm for the Energy Aware Routing problem

while respecting link capacities. We now present the greedy algorithm shown in Figure 2.2. It was originally proposed in [GMM12] for undirected graphs.

In this algorithm, the set of possible candidates to choose from is the set of links in the network. Initially, we find a valid routing with all active links. The selection function selects the link with the least amount of traffic on it. A link (u, v) can be added to the set of inactive links if and only if all requests can be routed when (u, v) and the previously selected links are inactive. If no valid routing can be found, the link is removed from the set of candidates. The algorithm stops when all links have been considered.

This algorithm constitutes a basis on which the algorithms proposed in this thesis are built upon. Indeed, in each of the problems considered the main goal is to minimize the number of active links. The routing constraints are thus the only changing components of the algorithm and depend on the problem considered, e.g., paths are constrained by the limited table capacity for the *Compression Problem* or by the execution order of the functions for the Service Function Chaining problem.

Chapters	Techniques used
Chapter 3	ILP, Greedy
Chapter 4	ILP, Greedy
Chapter 5	Greedy
Chapter 6	ILP, Greedy
Chapter 7	ILP, Column Generation
Chapter 8	ILP, Greedy, Column Generation

Table 2.1: Summary of the techniques used in each chapter

2.4 Summary of the technique used for each chapter

Table 2.1 summarizes the techniques used in each chapter of this thesis. While greedy algorithms and ILP formulations are the most used techniques, the last part of the thesis focuses on the Column Generation technique.

Bibliography

[Chv83] V. Chvatal. *Linear Programming*. Freeman, 1983 (cit. on p. 31).

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993 (cit. on p. 31).

[Cor+01] Thomas H. Cormen, Charles Eric Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. Vol. 6. MIT press Cambridge, 2001 (cit. on p. 37).

[GMM12] F. Giroire, D. Mazauric, and J. Moulierac. "Energy Efficient Routing by Switching-Off Network Interfaces". In: ed. by Naima Kaabouch and Wen-Chen Hu. IGI Global, June 2012. Chap. 10 - Energy-Aware Systems and Networking for Sustainable Initiatives, pp. 207–236 (cit. on pp. 37, 38).

Part I

Forwarding Rules Space Constraints

Forwarding Rule Space Constraints

Context

In classical networks, routers use distributed routing protocols such as Open Shortest Path First (OSPF) [Moy98] to decide on which interfaces packets should be forwarded. In Software Defined Network (SDN) networks, one or several controllers take care of path computations, and routers become simple forwarding devices. When a packet arrives, with a new destination for which no routing rule exists, the router contacts a controller that provides a route to the destination. Then, the router stores this route as a rule in its SDN table and uses it for next incoming matching packets. This separation of the control plane from the data plane allows a smoother control over routing and an easier management of the routers.

Also, SDNs aim at applying flow-based forwarding rules instead of destination based rules (as in legacy routers) to provide a finer control of the network traffic. For instance, in OpenFlow 1.3, forwarding decisions can be made taking into account from zero up to a maximum of 40 fields of a Transmission Control Protocol (TCP) or a User Datagram Protocol (UDP) packet. When any of the 40 fields should be ignored when forwarding a packet, such a field is set to “don’t care” bits. Due to the complexity of SDN forwarding rules, SDN forwarding devices need Ternary Content-Addressable Memories (TCAMs) to store their routing table (as classical Content-Addressable Memory (CAM) can only perform binary operations). However, TCAMs are more power hungry, expensive and physically bigger than binary CAMs available in legacy routers. Consequently, the available TCAM memory in routers is limited. Indeed, a typical switch supports between around a couple of thousands to no more than 25 thousands of 12-tuples forwarding rules, as reported in [Ste+12].

Undoubtedly, emerging switches will support larger forwarding tables [Bos+13], but TCAMs still introduce a fundamental trade-off between forwarding table size and other concerns like cost and power. The maximum size of routing tables is thus limited and represents a significant concern for the deployment of SDN technologies. This problem has been addressed in previous works, as discussed later, using different strategies, such as routing table compression [GMP14; Hu+15], or distribution of forwarding rules [Coh+14].

In this part, we examine a more general framework in which *table compression* using wildcard rules is possible. Compression of SDN rules was discussed in [GMP14]. The authors propose algorithms to reduce the size of the tables, but only by using a default rule. We consider here a stronger compression methodology in which any packet header field may be compressed. Considering *multiple field aggregation* is an important improvement as it allows a more efficient compression of routing tables, leaving more space in the TCAM to apply advanced routing policies, like load-balancing and to implement quality of service policies. In the following, we focus on compression of rules based on sources and destinations. However, our solution also applies if other fields are considered, such as Type of Service (ToS) field or transport protocol. The chapter is organized as follows.

Contributions and plan

We first present, in Chapter 3, the *Compression Problem* which consists in reducing the size of the forwarding tables using the default rule and aggregation rule on the source and destination. Several solutions are proposed in Section 3.2 such as an Integer Linear Program, an approximation algorithm, and a greedy heuristic algorithm. We compare them in Section 3.3 using randomly generated tables and using network tables obtained from simulations on SDNlib instances.

In Chapter 4, we define and explore the Energy Aware Routing with Compression (EARC) Problem. As the EAR problem is known to be NP-hard [Gir+10] (and thus EARC), we propose two Integer Linear Programs for the default rule and multi-field compression (see Section 4.2) as well as an efficient heuristic algorithm, in Section 4.3. The heuristic is composed of three modules: a routing module, in charge of finding paths for each demand in the network while respecting link and node capacities; a compression module, responsible for the reduction of the table size in the network; and an energy saving module that decides which link to shutdown. We further validated the solutions proposed for the *Compression Problem* in Chapter 3 by using them in the compression module. We compare all solutions on four SNDlib instances in Section 4.4 and show that we can save almost as much power as possible without capacity constraints, by jointly routing and compressing routing tables.

Finally, we present a dynamic variant of the heuristic presented in Chapter 4 without the energy saving module, MINNIE, in Chapter 5. It routes the traffic and compresses routing tables to satisfy link capacity and routing table size of the different forwarding devices. We validate our algorithm on multiple well-known data center topologies in Section 5.2. We show that one can deploy networks with up to 1 million flows using MINNIE by carefully choosing a threshold for compression. We further validated MINNIE using a testbed emulating a $k = 4$ Fat-Tree data center topology in Section 5.3. We demonstrate on the one hand that even with a small number of clients the limit regarding the number of rules is reached without compression, increasing the delay of new incoming flows. MINNIE, on the other hand, reduces the number of rules needed, with no packet losses, nor noticeable additional delays using TCAM.

Related work

To support a vast range of network applications, OpenFlow rules are more complex than forwarding rules in traditional Internet Protocol (IP) routers. For instance, access-control requires matching on source - destination IP addresses, port numbers and protocol [Cas+09] whereas a load balancer may match only on source and destination IP prefixes [RDJ11]. These complicated matching can be well supported using TCAM since all rules can be read in parallel to identify the matching entries for each packet. However, as TCAM is expensive and extremely power-hungry, the on-chip TCAM size is typically limited. Several works have tackled the distribution of the forwarding policies on a network considering the table size constraints, see [Ngu+16] for a survey. These solutions can be regrouped into three main categories.

In the first category, the rule space capacity of the whole network is used by spreading the rules in order to circumvent the individual switches capacities. In [KHK13] and [Kan+13], the authors propose similar solutions, in which the set of end points policies of the network is divided and then spread over the network so that every packet is affected by all the policies. However the routing policies are not taken into consideration. In both [Ngu+15] and [Coh+14], routing policies are dealt with by changing the path of the flows to take advantages of the table space from all the switches of the network. These types of solution do however change the path used to route a flow and thus impact the Quality of Service (QoS) of the network. In [Ngu+15], the authors propose OFFICER. It creates a default path for all communications, and later, some deviations are introduced from this path using different policies to reach the destination. According to the authors, the Edge First strategy, where the deviation is performed to minimize the number of hops between the default path and the target one, offers the best trade-off between the required QoS and forwarding table size. Note however, that applying this algorithm could unnecessarily penalize the QoS of flows when the routers' forwarding tables are rarely full.

In the second category, caching techniques are used to limit the number of concurrent rules in a switch, such as CacheFlow, proposed in [Kat+16], which introduces a CacheMaster module and a shared section of software switches per TCAM (available in hardware switches only). The CacheMaster constructs the dependency tree of the rules to be installed and then distributes the rules between the TCAM and the software switches, placing the most popular rules in the hardware switch, thus enabling fast forwarding for the biggest possible amount of traffic. When a packet needs a forwarding rule not available in the TCAM, such a packet is forwarded to the software switches, which send back the packet to the hardware switch in a predetermined input port, to be resent at a specific output port. If the software switches do not have a matching rule, the SDN controller is called. The weaknesses of CacheFlow relies in its inherent architecture, as this solution requires the installation of a software switch for every hardware switch, which might need a reorganization of the network cabling and additional resources to host software switches. Secondly, the optimal number of needed software switches can be difficult to determine, due to the fact that for performance reasons, software switches must only keep forwarding rules (whose number depends on the traffic characteristics) in the kernel memory space, which is limited. Lastly, the two-layer architecture of CacheFlow (i.e., software switches over a hardware switch) increases the delay to contact the controller and install missing rules.

The third category considers reducing the size of the table by using compression techniques. Our work falls under this category. Works such as [BK14] or [KS13] propose modifications to the rule shape to reduce their size, but requires hardware modifications. To the best of our knowledge, the closest papers to our work are [Hu+15; BM14; GMP14]. In [Hu+15] the authors introduce XPath which identifies end-to-end paths using path ID and then compresses all the rules and pre-install the necessary rules into the TCAM. We compare our results with the ones of XPath in Section 5.2.2. Minnie uses fewer rules even in the case of an all-to-all traffic as XPath codes the routes for all shortest paths between sources and destinations. This is at the cost of less path redundancy which is useful for load-balancing and fault tolerance. Network operators should consider this trade-off when choosing which method to use. In [BM14] the authors suggest SDN rule compression by following the concept of longest prefix matching with priorities using the Espresso [TNW96] heuristic and show that their algorithm leads to 17% savings only. We succeed in reaching better compression ratios.

To the best of our knowledge, only [GMP14] and [Awa+17] consider the joint problem of compression and energy aware routing. While the solution proposed in [Awa+17] falls into the first category, the solution in [GMP14] uses the default rule to reduce the size of the table. We extend this solution by considering other types of compression.

Chapter 3

The Compression Problem

Contents

3.1 Introduction

In this chapter, we study the *Compression Problem* which consists in reducing the size of the forwarding tables using the default rule and aggregation rule on source and destination. This problem is the common theme of the first part of this thesis.

After a definition of the problem in Section 3.1.1, we propose several solutions such as an Integer Linear Program, an approximation algorithm, and a greedy heuristic algorithm, in Section 3.2.

We compare them in Section 3.3 using randomly generated tables and using network tables obtained from simulations on SDNlib instances.

Flow	Output port
(0, 4)	Port-4
(0, 5)	Port-5
(0, 6)	Port-5
(1, 4)	Port-6
(1, 5)	Port-4
(1, 6)	Port-6
(2, 4)	Port-4
(2, 5)	Port-5
(2, 6)	Port-6

(a) Without compression

Flow	Output port
(0, 5)	Port-5
(0, 6)	Port-5
(1, 4)	Port-6
(1, 6)	Port-6
(2, 5)	Port-5
(2, 6)	Port-6
$(*, *)$	Port-4

(b) Default port compression

Flow	Output port
(1, 5)	Port-4
(2, 6)	Port-6
$(1, *)$	Port-6
$(*, 4)$	Port-4
$(*, *)$	Port-5

(c) Multi-field compression

Table 3.1: Examples of routing tables: (a) without compression, (b) with default port compression (only $(*, *)$ rule), (c) with multi-field compression (three possible aggregation rules), giving the routing table with minimum number of rules.

3.1.1 Definition of the problem

A forwarding table is composed of multiple entries that match flows with corresponding action(s). In OpenFlow 1.3, the matching can be done on 40 fields from the packet header. For each field, the matching rule can use a particular value or a wildcard (noted $*$) that can accept any value. The action associated with a matching rule can be to drop the packet, modify the header, or forward it to a specific port.

In the following, we consider the action to be limited to a forward to outgoing ports. We also limit the use of the wildcard to the source and destination addresses. However, our solution also applies if other fields are considered such as ToS field or transport protocol. We compress a table by using either the aggregation by source (i.e., $(s, *, p)$), by destination (i.e., $(*, t, p)$) or by the default rule (i.e., $(*, *, p)$). When only the default rule is used, we talk about *default port compression*, and, when all the wildcard may be used, about *multi-field compression*.

An example is given in Table 3.1. Table 3.1a represents the original table. Table 3.1b provides the result using default port compression and Table 3.1c the one using multi-field compression, that is when all three types of wildcard rules can be used to obtain the optimal compressed table. *Since* *multiple entries can correspond to the same flow, rule priority is given from top to bottom.* Indeed, in Table 3.1c, if we exchange the priorities of the rules $(1, *, Port - 6)$ and $(*, 4, Port - 4)$, a flow from source 1 to destination 4 is no longer forwarded through Port-4 like in the original table.

3.2 Heuristics and exact formulations

We propose several solutions to solve the *Compression Problem*: First, *Comp-Default*, giving optimal solutions for the default port compression. We then provide an integer linear program, *Comp-LP*, which gives optimal solutions for the multi-field compression. However, as the problem is NP-Hard (see [GHM15; GHM16] for a proof), the program does not scale to large tables (also see Section 3.3 for compression time and a discussion). We thus provide two heuristic algorithms for the compression problem, *Comp-Greedy* and *Comp-Direction*.

3.2.1 Default Rule (Comp-Default)

When using only the default port compression, finding the optimal solution is simple. The algorithm finds the most occurring port p^* in the forwarding table, remove all the rules with p^*, and add the default rule $(*, *, p^*)$ at the end of the table.

3.2.2 Integer Linear Programming (Comp-LP)

We first define the following notations. We then formulate the problem as an Integer Linear Program.

- $\mathcal{R}$, set of rules in the forwarding table
- $\mathcal{S}$, set of sources in the forwarding table
- $\mathcal{T}$, set of destinations in the forwarding table
- $\mathcal{P}$, set of ports of the router
- $r_{stp} \in \{0, 1\}$, where $r_{stp} = 1$ if the rule (s, t, p) exists
- $s_{tp} \in \{0, 1\}$, where $s_{tp} = 1$ if the wildcard rule $(*, t, p)$ exists

- $t_{sp} \in \{0,1\}$, where $t_{sp} = 1$ if the wildcard rule $(s,*,p)$ exists
- $d_p \in \{0,1\}$, where $d_p = 1$ if p is the default port of the table
- $o_{st} \in \{0,1\}$, where $o_{st} = 1$ if the wildcard rule for the source s has higher priority than the one for the destination t.
- $\overline{o}_{ts} \in \{0,1\}$, where $\overline{o}_{ts} = 1$ if the wildcard rule for the destination t has higher priority than the one for the source s. By definition, $o_{st} = 1 - \overline{o}_{ts}$.

We want to minimize the total number of rules in the compressed table (3.1). All original rules must have a corresponding rule in it (3.2).

$$\min \quad \sum_{(s,t,p)\in\mathcal{R}} r_{stp} + \sum_{p\in P}\left(\sum_{t\in T} s_{tp} + \sum_{s\in S} t_{sp}\right) + \sum p \in \mathcal{P} d_p \tag{3.1}$$

$$r_{stp} + s_{tp} + t_{sp} + d_p \geq 1 \qquad \forall (s,t,p) \in \mathcal{R} \tag{3.2}$$

There can be at most one default port (3.3), one wildcard rule per source (3.4) and one wildcard rule per destination (3.5) in the table.

$$\sum_{p\in\mathcal{P}} d_p \leq 1 \tag{3.3}$$

$$\sum_{p\in\mathcal{P}} s_{tp} \leq 1 \qquad \forall t \in \mathcal{T} \tag{3.4}$$

$$\sum_{p\in\mathcal{P}} t_{sp} \leq 1 \qquad \forall s \in \mathcal{S} \tag{3.5}$$

For every rule (s,t,p) in the original table, if a matching wildcard rules exists with a different port, i.e., $(s,*,p' \neq p)$ or $(*,t,p' \neq p)$, either the original rule (s,t,p) or a matching wildcard rule with the right port exists with a higher priority, (3.6) to (3.9).

$$r_{stp} + s_{tp} \geq t_{sp'} \qquad \forall (s,t,p) \in \mathcal{R}, p' \in \mathcal{P} \setminus \{p\} \tag{3.6}$$

$$r_{stp} + \overline{o}_{ts} \geq t_{sp'} \qquad \forall (s,t,p) \in \mathcal{R}, p' \in \mathcal{P} \setminus \{p\} \tag{3.7}$$

$$r_{stp} + t_{sp} \geq s_{tp'} \qquad \forall (s,t,p) \in \mathcal{R}, p' \in \mathcal{P} \setminus \{p\} \tag{3.8}$$

$$r_{stp} + o_{st} \geq s_{tp'} \qquad \forall (s,t,p) \in \mathcal{R}, p' \in \mathcal{P} \setminus \{p\} \tag{3.9}$$

Finally, we remove the cyclic order dependencies between rules.

$$1 \leq o_{s_1t_1} + \overline{o}_{t_1s_2} + o_{s_2t_2} + \overline{o}_{t_2s_1} \leq 3 \qquad \forall s_1 \neq s_2 \in \mathcal{S}, t_1 \neq t_2 \in \mathcal{T} \tag{3.10}$$

As we will see in Section 3.3, the computation time of the ILP is prohibitive for larger tables. Note that a slight alteration of this formulation is used in the formulation proposed in Chapter 4.

3.2.3 Most savings heuristic (Comp-Greedy)

For this heuristic algorithm, we add wildcard rules for sources or destinations in a greedy way, based on the highest potential compression ratio. The *potential compression ratio* of a source s (or destination t) is equal to the number of rules with the most repeated port p among all the rules with s (or t) over the total number of rules with s (or t). At each step, we compute the potential compression ratio of all sources and destinations. We then add the wildcard rule corresponding to the source or destination with the highest potential compression ratio, and we remove all the rules matching the wildcard rule. Ties between ports are resolved at random. Note that at each step, the compression ratios of the other sources can be affected. We thus recompute them at each step.

3.2.4 Direction Based Heuristic (Comp-Direction)

We present a second heuristic, shown in Algorithm 1. It was studied in [GHM15; GHM16] and was proved to be efficient, as it provides a 3-approximation of the compression problem. We will demonstrate that it is also efficient in practice.

It first computes three compressed routing tables (aggregation by source (line 1-22), by destination (line 23-44) and by the default rule (line 45-52)) and then chooses the smallest one, as explained in more detail below.

Given a routing table such as the one given in Table 3.2a, the algorithm first considers an aggregation by source (Table 3.2b) using $(s, *, p^*_s)$ rules. The main principle is simple, but there is a small technicality to break ties. We first consider the sources one by one and choose (one of) the most occurring port(s) in the rules with this source. It corresponds to the port allowing to compress the most rules using a rule of aggregation by source. Then, we use the default rule to reduce the number of aggregated rules.

There is a small technicality to break ties when there are several most occurring ports. As a matter of fact, the choice taken of aggregation ports for each source affects the default port chosen. We thus postpone the choice

Flow	Output port
(0, 4)	Port-4
(0, 5)	Port-5
(0, 6)	Port-5
(1, 4)	Port-6
(1, 5)	Port-4
(1, 6)	Port-6
(2, 4)	Port-4
(2, 5)	Port-5
(2, 6)	Port-6

(a) Without Compression

Flow	Output port
$(0, 4)$	Port-4
$(1, 5)$	Port-4
$(2, 4)$	Port-4
$(2, 5)$	Port-5
$(0, *)$	Port-5
$(*, *)$	Port-6

(b) Source table

Flow	Output port
(1, 4)	Port-6
$(1, 5)$	Port-4
$(0, 6)$	Port-5
$(*, 4)$	Port-4
$(*, 5)$	Port-5
$(*, *)$	Port-6

(c) Destination table

Flow	Output port
(0, 5)	Port-5
(0, 6)	Port-5
(1, 4)	Port-6
(1, 6)	Port-6
(2, 5)	Port-5
(2, 6)	Port-6
$(*, *)$	Port-4

(d) Default only

Flow	Output port
(1, 5)	Port-4
(2, 6)	Port-6
$(1, *)$	Port-6
$(*, 4)$	Port-4
$(*, *)$	Port-5

(e) Optimal solution (ILP)

Table 3.2: Examples of routing tables: (a) without compression, (b) compression by the source, (c) compression by the destination, (d) default rule only, and (e) routing table with minimum number of rules given by Integer Linear Program. The order of the rules reads top to bottom.

of the source aggregation port in case of ties to choose the default port compressing the largest number of aggregation rules, as explained in details below.

For each source s, we need to find the port p_s^* such that we can aggregate using the rule $(s, *, p_s^*)$, and the port p^* to aggregate with the default rule $(*, *, p^*)$. First, we compute the set of most occurring ports for each source s, noted $\mathcal{P}_s^*$. The default port p^* is thus the most occurring port in all sets $\mathcal{P}_s^*$. If multiple ports can be chosen, one is selected at random. Then, for each source s, the port p_s^* is equal to p^* if $p^* \in \mathcal{P}_s^*$. Otherwise we choose at random among $\mathcal{P}_s^*$. Once the ports for the aggregated rules are chosen, we build the compressed table. First, we add rules that cannot be aggregated (line 12), i.e., $(s, t, p \neq p_s^*)$. Then, we add all the aggregation rules by source that do not use the default port p^* (line 15), i.e., $(s, *, p_s^* \neq p^*)$. Finally, we add the default rule $(*, *, p^*)$ (line 16). The order of insertion in the routing gives the order for the matching, i.e., non aggregated rules, then source aggregation rules and then default rule.

For example, the sets of the most occurring ports of sources 0, 1, and 2 in Table 3.2a are {Port-5}, {Port-6}, {Port-4, Port-5, Port-6}, respectively. Since Port-5 and Port-6 appear two times each, we choose at random Port-6 to be the default port. The ports used for the aggregation by source for 0, 1, 2 are then Port-5, Port-6, Port-6, respectively. Port-6 is chosen for the source 2, because it is the default port. We can now build the compressed table by adding all rules that have ports different than their corresponding aggregate rules: $(0, 4, 4)$, $(1, 5, 4)$, $(2, 4, 4)$, $(2, 5, 5)$. Then, we add all aggregate rules with a port different from the default port: $(0, *, 5)$. Finally, we add the default rule $(*, *, 5)$. This gives us the compressed table in Table 3.2b.

For the second compressed routing table (Table 3.2c), we do the same compression considering the aggregation by destination with $(*, t, p_t^*)$ rules. As for the third table (Table 3.2d) a single aggregation using the best default port is performed, i.e., one of the most occurring port in the routing table becomes the default port (tie broken uniformly at random). This table is equivalent to the one produced by Comp-Default. We then choose the smallest routing table among the three computed ones.

Algorithm 1: Compressing a table

```
   Input: Set of rules $R$
   Output: Compressed rules
   // Compression by source
1  list of rules $C_r$ // order of insertion = order of matching
2  foreach $s \in V$ do
3      $\mathcal{P}_s^*$, set of most occurring ports $p$ in $\{(s,t,p) \mid \forall t \in V\}$;
4  $p^*$ = most occurring port in all $\mathcal{P}_s^*$ // ties are broken at random
5  foreach $s \in V$ do
6      if $p^* \in \mathcal{P}_s^*$ then
7          $p_s^* = p^*$
8      else
9          $p_s^*$ = most occurring port in $\mathcal{P}_s^*$ // ties are broken at random
10 foreach $(s,t,p) \in R$ do
11     if $p \neq p_s^*$ then
12         add $(s,t,p)$ to $C_r$;
13 foreach $s \in V$ do
14     if $p_s^* \neq p^*$ then
15         add $(s,*,p_s^*)$ to $C_r$;
16 add $(*,*,p)$ to $C_r$;
   // Compression by destination
17 list of rules $C_c$ // order of insertion = order of matching
18 $\mathcal{P}_t^*$, set of most occurring ports $p$ in $\{(s,t,p) \mid \forall t \in V\}$, $\forall s \in V$;
19 $p^*$ = most occurring port in $\mathcal{P}_t^*$ // ties are broken at random
20 foreach $t \in V$ do
21     if $d \in \mathcal{P}_t^*$ then
22         $p_t^* = p^*$
23     else
24         $p_t^*$ = most occurring port in $\mathcal{P}_t^*$ // ties are broke at random
25 foreach $(s,t,p) \in R$ do
26     if $p \neq p_t^*$ then
27         add $(s,t,p)$ to $C_c$;
28 foreach $t \in V$ do
29     if $p_t^* \neq p^*$ then
30         add $(s,*,p_t^*)$ to $C_c$;
31 add $(*,*,d)$ to $C_c$;
   // Default port compression
32 list of rules $C_d$;
33 $p^*$ = most occurring port in R // ties are broke at random
34 foreach $(s,t,p) \in R$ do
35     if $p \neq p^*$ then
36         add $(s,t,p)$ to $C_d$;
37 add $\{(*,*,p^*)\}$ to $C_d$;
38 return smallest set of rules between $C_r$, $C_c$, and $C_d$
```

3.3 Compression of forwarding tables

We now evaluate typical compression ratios, compression times and compare the different solutions proposed in Section 3.2: the solution using default port compression (Comp-Default) and the three solutions using the multi-field compression: the optimal one from the Linear Program (Comp-LP), when it is possible to compute it, the Direction Based Heuristic (referenced as Comp-Direction or Comp-Dir in short), and last, the Most savings heuristics (Comp-Greedy). As instances, we consider randomly generated routing tables as well as network routing tables coming from simulations on SNDlib instances [Orl+10].

3.3.1 Random tables

In this section, we focus on the compression of random tables. The following parameters are used to generate the random tables studied:

- the number of sources and destinations n
- the number of ports of the switch p
- the density of the corresponding matrix $0 \leq d \leq 1$

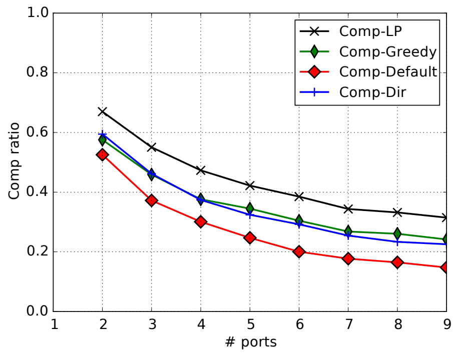
For a pair source-destination, there is an entry in the table with probability d, and in this case, the exit port is chosen uniformly at random among the p ports.

We show the average compression ratio of the solutions proposed in Section 3.2, given by

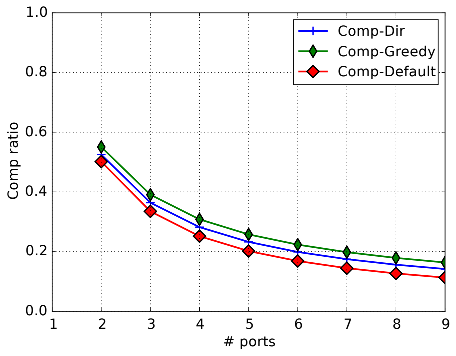
$$1 - \frac{\text{size of the compressed table}}{\text{size of the original table}}$$

, as a function of the parameters used to build the random matrices. We vary the number of ports in the experiments of Figure 3.1, the number of network nodes (corresponding to the number of sources and destinations) in Figure 3.2, and the table density in Figure 3.3. Each point represents the average of the results for 10 random forwarding tables for the comparison with the Linear Program (LP) and 20 for the heuristics.

Gap from optimal for small tables. For small routing tables, we are able to compute the optimal compressed tables using the integer linear program

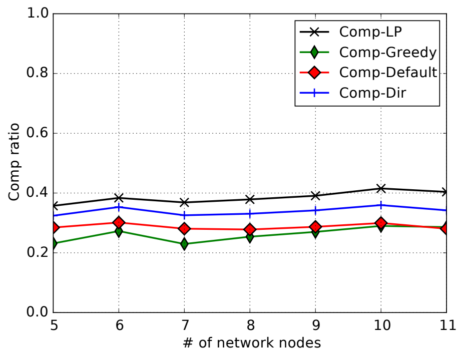


(a) ~ 112 rules $(n = 15, d = 0.5)$

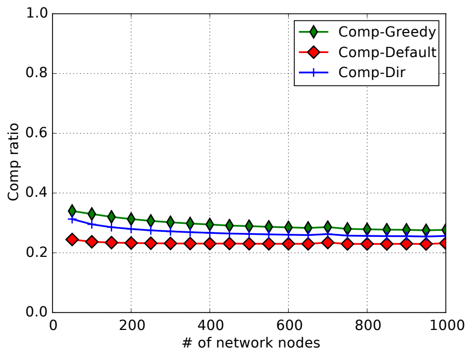


(b) $\sim 101\,250$ rules $(n = 450, d = 0.5)$

Figure 3.1: Compression ratio as a function of the number of ports for the four compression methods.



(a) All methods



(b) Without ILP

Figure 3.2: Compression ratio as a function of the number of network nodes, i.e., the number of sources and destinations, for the four compression methods.

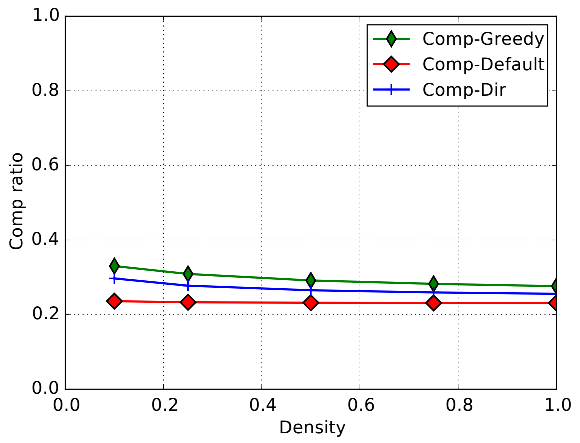


Figure 3.3: Compression ratio as a function of the forwarding tables density for the four compression methods.

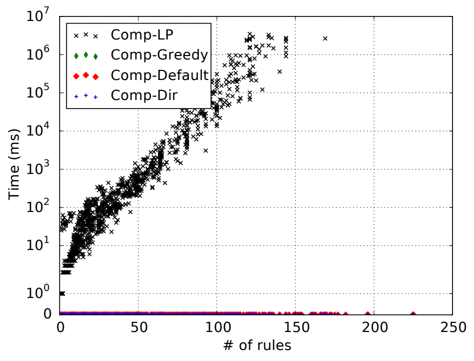
(see Figure 3.1a and Figure 3.2a). As an example, in Figure 3.1a, we compare Comp-LP and the other three solutions on a set of random tables with $n = 15$ sources/destinations, a density of 0.5 with a number of ports between 2 and 9. Without surprise, the ILP compresses better than the other 3 solutions with 68% ratio at only two ports to 32% with nine ports. The two heuristics present the same compression with a ratio of 59% at two ports and 23% at nine ports. Finally, the only use of the default port yields to the worst com pression as it compresses 53% of the rules with 2 ports and only 15% at nine ports. Similarly, the difference of compression ratio in Figure 3.2 is between 4 and 10% when comparing the optimal solution with the Comp-Greedy and Comp-Direction heuristics. Default port is the less efficient solution with a compression ratio around 23%, when the compression ratio of Comp-Greedy and Comp-Direction heuristics is around 30%. In Figure 3.2a, we vary the number of network nodes between 5 and 11. The global comparison between solutions is similar, except that, when there is a small number of network nodes, Comp-Greedy does not behave well and provides worse results than Comp-Default. The explanation is that, for small tables, Comp-Greedy adds source and destination aggregation rules that are not necessary, as a default rule works well. Because of the order between source and destination rules, most of these rules cannot be aggregated when we add the default rule, leading to an inefficiency. The problem disappears for larger numbers of network nodes (larger than 10), and thus would not appear for ISP networks which have more network nodes.

Comparison between heuristics for larger tables. However, the ILP does not scale well for larger tables. In Figure 3.1b, we only compare the two heuristics and Comp-Default on tables with $n = 450$ sources/destinations and a density of 0.5. First, we notice that the two heuristics Comp-Greedy and Comp-Direction obtain the best results. However, the Comp-Default solution is not far behind with a ratio between 49% and 11% for the random tables. We will see later that the difference is significantly higher for real network tables. Comp-Greedy behaves better than Comp-Direction, with a compression ratio between 55% and 16% to be compared to a compression ratio between 52% and 14% for Comp-Direction.

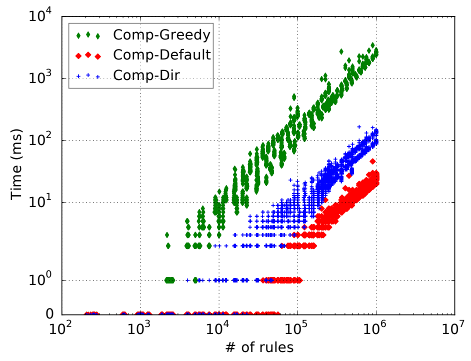
Impact of the parameters. The compression ratio is very sensitive to the number of ports, see Figure 3.1. The compression ratio varies from 55% to 18% when the number of ports ranges from 2 to 9 for a random matrix with around 100 000 rules. Similar results are observed for small tables with variations from 70% to 35%. *We observe higher compression ratio for smaller numbers of ports.* This is expected as, for example, the impact of setting a default port is higher when the number of ports is lower. For two ports, using a default port saves at least 50% of the rules.

Conversely, the *density and the size (number of network nodes) of the forwarding tables do not have an important impact on the compression ratio.* For the experiments in Figures 3.2, the compression ratio varies of only a few percents when the number of network nodes increases from 5 to 11, and then from 50 to 1000; and similarly, when the density goes from 0.1 to 1, even if it represents a 10-fold increase in the number of rules in the table (Figure 3.3). However, density and size of the forwarding tables have an impact on the compression time as discussed below.

Compression time. We study the time to compress forwarding tables. This time depends mostly on the number of entries in the forwarding table, as presented in Figure 3.4. The compression time using linear programming (Comp-LP) is a lot higher than the one using heuristic algorithm: around 1000 s for only 125 rules, when it takes a lot less than 1 ms for the heuristics. We thus had to present the results for Comp-LP independently in Figure 3.4(a) with a different log-scale ($[0, 10^7]$), compared to ($[0, 10^4]$) for Figure 3.4(b). We observe that the compressing time of Comp-LP increases exponentially with the number of rules. It reaches the limit of one hour we



(a) ILP



(b) Heuristic Algorithms

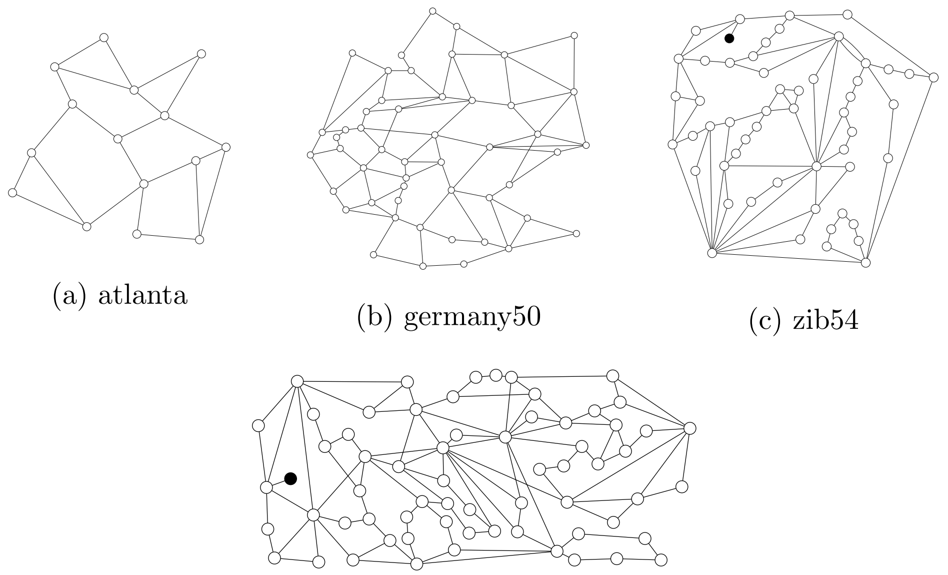
Figure 3.4: Compression times of forwarding tables as a function of the number of rules in the tables for four methods of compression (two different scales for Time).

had set for tables with a little bit more than 150 rules. Note that a network with ten nodes cannot have more than 90 entries in a routing table (in the extreme case of one central node seeing all the possible flows). Thus, we know that we can use Comp-LP for networks with some nodes reaching 10, and surely a little bit more as all traffic usually is not routed through a single node. In fact, we show in Section 4.4, that LP runs on the SNDlib Atlanta network with 15 nodes, but that it is not usable for larger networks.

On the contrary, the *compression time of the heuristic algorithms is very low* and does not increase exponentially, but linearly in the number of rules. A large network with 100 nodes cannot have more than 10 000 entries in a routing table. A forwarding table of this size is compressed in less than 10 ms (around 10 ms for Comp-Greedy, 1 ms for Comp-Direction, and less than 1 ms for Comp-Default). It is even possible to compress a routing table with a million rules (for a network of more than a thousand nodes) in a little bit more than 1 s for Comp-Greedy and less than 10 ms for Comp-Direction and Comp-Default. The heuristic algorithms for compression can thus be used for very large networks and have a very low execution time.

3.3.2 Network tables

We now compare the solutions on tables from routing on backbone networks using the routing module presented in Section 4.3.1. We use four of the



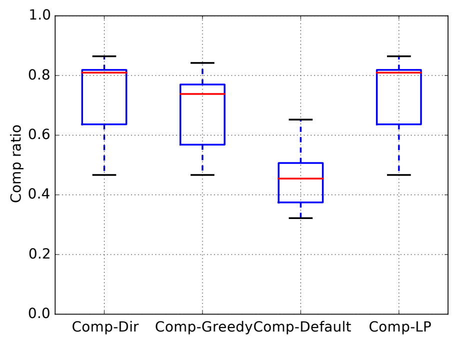
(a) atlanta

(b) germany50

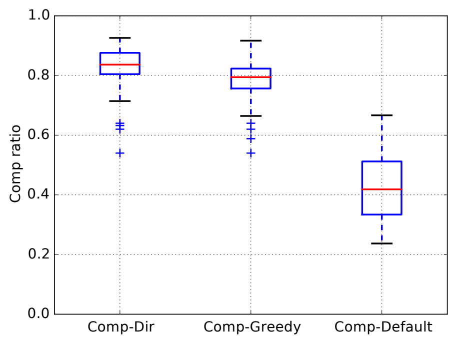
(c) zib54

(d) ta2

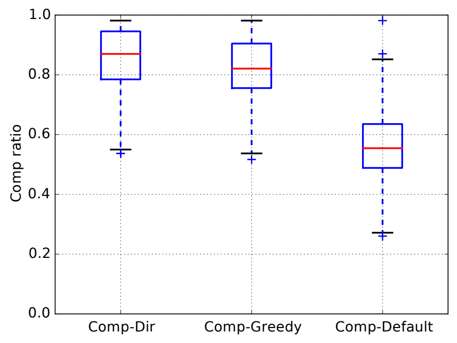
Figure 3.5: The four SNDlib topologies used. Each edge corresponds to two directional links. Black nodes are switches with only one port.



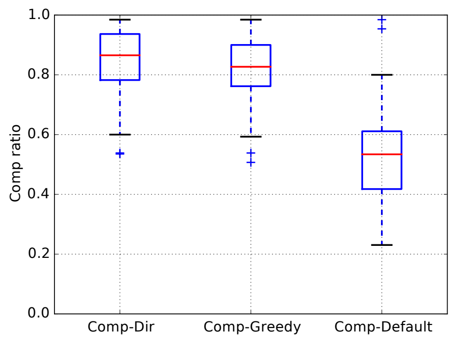
(a) atlanta



(b) germany50



(c) zib54



(d) ta2

Figure 3.6: Compression ratio of the three different heuristics on SNDlib topologies.

SNDlib instances shown in Figure 3.5:

- atlanta network with 15 nodes and 44 directed links,
- germany50 network with 50 nodes and 176 directed links,
- zib54 network with 54 nodes and 216 directed links, and
- ta2 network with 81 nodes and 162 directed links.

For each network, we compute a routing of all demands without considering a limit on the number of rules. We then extract the forwarding tables for all routers. We then compress each of them with the different compression solutions. Since the ILP does not scale, we only compare it with the other solutions on the `atlanta` network, see Figure 3.6a. On the other three networks, we compare the EARC-H-Direction, EARC-H-Greedy and EARC-H-Default solutions, see Figures 3.6b to 3.6d for germany50, ta2, and zib54 networks, respectively.

Compression rates The first global observation is that the solutions achieve *higher compression rates for network tables than for random tables*, with median values *around 80% for all networks.* This is good news as it shows the efficiency of the algorithms for practical cases. The explanation of this phenomenon is that real network tables have a larger number of repeating ports traffic originating from a source or going to a destination, than random matrices.

We remark that some tables show a compression ratio near 100% for all solutions for `zib54` and `ta2`. These tables corresponds to the two routers with only one outgoing port (the two routers in black in Figure 3.5). Thus, only the default port can be used to route all the demands.

Comparison of the solutions. In the `atlanta` network, we see that the difference in efficiency between the heuristics, Comp-Direction, and Comp-Greedy, and the linear program for compression, Comp-LP, is smaller than in the case of random tables. The compression rate of Comp-Direction is almost the same as the one of Comp-LP, with a median ratio of 81%. This is also good news: *real network tables are easier to compress than random tables.* We thus can suppose that the results of the heuristics on larger networks should be good. And, in fact, we obtain very high compression rates: the median is 83% for `germany50`, 86% for `ta2` and `zib54`.

Last, we observe that the *difference between the two levels of compression is more significant for real network tables than for random tables.* The median ratios of the Comp-Default solution are about 30% lower than the one from the Comp-Greedy heuristics. This shows the importance of considering multi-field compression.

The two heuristics using multi-field compression, *Comp-Direction and Comp-Greedy, show similar results on all networks.* While the Comp-Greedy heuristic provides better compression ratios on random tables, *the advantage for real network tables is for the Comp-Direction heuristic*: the median ratio is 4% higher for `germany50`, `ta2`, and `zib54`, and 8% for `atlanta`. We use both heuristics in the simulations of the next chapter in which we obtain results for the EARC problem on practical network instances.

3.4 Conclusion

The finer control offered by SDN comes at the cost of more complex forwarding rules and smaller space constraints for forwarding tables. Even if newer hardware are expected to increase the capacities of the forwarding tables, the high cost and high power usage of the TCAM is an obstacle to the deployment of complex routing policies in large networks.

In this chapter, we studied the *Compression Problem*, which consists of minimizing the size of a forwarding table using a default rule and possible wildcard rules, i.e., aggregating rules on the source or the destination of a network flow.

We studied several solutions to the problem on randomly generated forwarding tables as well as routing table generated from SDNlib instances. We show that multi-field compression significantly increases the compression ratio of the table. We also show that the Comp-Direction heuristic provides the best results on routing tables.

In the next two chapter, we will continue to study the *Compression Problem*, first, in the EAR context and then in the context of data centers network. As we showed that the heuristics presented in this chapter are efficient, we re-use them in both settings.

Chapter 4

Energy Aware Routing with Compression

Contents

4.1 Introduction

In this chapter, we define and explore the Energy Aware Routing with Compression (EARC) Problem. To the best of our knowledge, we are the first to study the EARC problem. As the EAR problem is known to be NP-hard [Gir+10] (and thus EARC), we propose two Integer Linear Programs for the default rule and multi-field compression (see Section 4.2) as well as an efficient heuristic algorithm, in Section 4.3. The heuristic is composed of three modules: a routing module, in charge of finding paths for each demand in the network while respecting link and node capacities; a compression module, responsible for the reduction of the table size in the network; and an energy saving module that decides which links to shutdown. We further validated the solutions proposed for the *Compression Problem* in Chapter 3 by using them in the compression module. We compare all solutions on four SNDlib instances in Section 4.4 and show that we can save almost as much power as possible without capacity constraints, by jointly routing and compressing routing tables.

4.1.1 Definition of the problem

Energy Aware Routing with Compression (EARC)

We consider a backbone network as a *directed* graph $G = (V, A)$. The nodes in V describe routers, and the arcs in A represent connections or links between those routers. The links have a limited capacity. We denote by C_{uv} the capacity of a link (u, v). The nodes have a limited memory space to store rules, and we note S_u the maximum number of rules can be installed at router u. We denote by D^{st} the demand of traffic flow from node s to node t such that $D^{st} \geq 0, s \neq t \in V$. The *objective is to find a feasible routing for all traffic flows, respecting the capacity and the rule space constraints and being minimal in energy consumption.* We name the problem *Energy Aware Routing with Compression (EARC).* Since the power consumption of routers is mostly independent of traffic load as stated in related work, the energy consumption of the network is given by the number of active links in our model. We consider that routers have to stay powered on in backbone networks as they are the points of entry and exit of traffic.

Power Model

Campaigns of measures of power consumption (see, e.g., [Cha+08]) show that a network device consumes a significant amount of its power as soon as it is switched on. Following this observation, on/off power models have been proposed and studied. Later, researchers and hardware constructors have proposed more energy proportional hardware models [Nic+12]. To encompass those different models, see [Idz+16] for a discussion, we use a hybrid power model in which the power of an active link (u, v) is expressed as

$$P_{uv}^{\text{IDLE}} + \frac{\text{BW}_{uv}}{C_{uv}} P_{uv}^{\text{LOAD}}$$

where $P^{\text{IDLE}}(u, v)$ represents the energy used when the link uv is switched on, BW_{uv} the bandwidth that is carried on uv, and $P^{\text{LOAD}}(u, v)$ the additional energy consumed by (u, v) when it is fully capacitated, i.e., when the amount of carried bandwidth equals the transport capacity (C_{uv}) of link (u, v). We assume that links can be put into sleep mode, by putting to sleep both endpoint interfaces. Routers cannot be put into sleep mode, as there are the sources/destinations of network traffic.

4.2 Integer Linear Programming

We propose two Integer Linear Programs to solve the EARC problem. In the first one, *EARC-LP-Default*, only the default port compression is allowed, while multi-field compression is used in the second, *EARC-LP-Multi*. The first program thus is less powerful but runs faster. We were able to obtain optimal solutions for small networks using both ILPs.
The following notations are used in both formulations:

- x_{uv}: binary variable to indicate if the link (u, v) is active or not.
- $\mathcal{D}$: the set of all traffic demands to be routed.
- $D^{st} \in \mathcal{D}$: demand of traffic flow from s to t.
- C_{uv}: capacity of a link (u, v).
- C_u: maximum number of rules that can be installed at router u.

4.2.1 EARC with default port Compression (EARC-LP-Default)

In this version of the problem, a flow can be routed following the FIB, that contains only *perfect match* rules, or via the default port. The following notations are used in the formulation of the ILP:

- $f_{uv}^{st} \in \{0,1\}$: a flow (s,t) that is routed on the link (u,v) by a distinct rule. We call f_{uv}^{st} as normal flow.
- $g_{uv}^{st} \in \{0,1\}$: a flow (s,t) that is routed on the link (u,v) by a default rule. g_{uv}^{st} is called default flow to distinguish from the normal flow f_{uv}^{st}.
- $f_{uv} \in \mathbb{R}^{+}$: sum of the flows routed on the link (u,v).
- $k_{uv} \in \{0,1\}$: binary variable to indicate if the default port of the router u is to go to v or not.
- $x_{uv} \in \{0,1\}$: binary variable to indicate if the link (u,v) is active or not.

We want to minimize the power consumption of the network (4.1).

$$\min \sum_{(u,v)\in A} (P_{uv}^{\text{IDLE}} x_{uv} + P_{uv}^{\text{LOAD}} \frac{f_{uv}}{C_{uv}}) \tag{4.1}$$

The flow conservation constraints (4.2) express that the total flows entering and leaving a router are equal (except the source and the destination nodes). It is noted that a normal flow entering a router can become a default flow on outgoing link and vice versa.

$$\sum_{v\in N^-(u)} f_{vu}^{st} + g_{vu}^{st} - \sum_{v\in N^+(u)} g_{uv}^{st} - f_{uv}^{st} = \begin{cases} -1 & \text{if } u = s, \\ 1 & \text{if } u = t, \\ 0 & \text{else} \end{cases}$$
$$\forall u \in V, (s,t) \in \mathcal{D} \tag{4.2}$$

A flow cannot be router as both a default flow and a normal one (4.3).

$$f_{uv}^{st} + g_{uv}^{st} \le 1 \qquad \forall (u,v) \in A, (s,t) \in \mathcal{D} \tag{4.3}$$

Link capacities are given by Equation (4.4) and no flow is allowed to be forwarded on a disabled link.

$$f_{uv} = \sum_{(s,t)\in\mathcal{D}} D^{st}(f_{uv}^{st} + g_{uv}^{st}) \le C_{uv} x_{uv} \qquad \forall (u,v) \in A \tag{4.4}$$

The total number of rules in the table of a router is equal to the sum of the normal flow forwarded from the router. It cannot exceed the table capacity C_u minus the reserved rule for the default port (4.5).

$$\sum_{(s,t)\in\mathcal{D}} \sum_{v\in N(u)} f_{uv}^{st} \leq C_u - 1 \qquad \forall u \in V \tag{4.5}$$

Finally, we limit the number of default port to one per router (4.6). A demand can only be forwarded on an edge as a default flow if is the edge of the default port (4.7).

$$\sum_{v\in N(u)} k_{uv} \leq 1 \qquad \forall u \in V \tag{4.6}$$

$$g_{uv}^{st} \leq k_{uv} \qquad \forall (u,v) \in A, (s,t) \in \mathcal{D} \tag{4.7}$$

4.2.2 EARC with multi-field Compression

In this version, we consider that the forwarding table contains several wildcard rules. These rules can match any flow that comes from a source s (i.e., $(s,*,p)$) or goes to a destination t (i.e., $(*,t,p)$). The following notations are used for the formulation of the ILP:

- $\mathcal{S}$, set of all sources
- $\mathcal{T}$, set of all destinations
- $f_{uv}^{st} \in \{0,1\}$, where $f_{uv}^{st} = 1$ if the flow (s,t) is routed on the link (u,v)
- $f_{uv} \in \mathbb{R}^+$: sum of the flows routed on the link (u,v).

For each router u, we also define the following sets of variables used for the compression of the tables, similar to the ones defined in Section 3.2.2. We use the notation p_v to define the port of the router u connected to the router v.

- $r_{stp_v}^u \in \{0,1\}$, where $r_{stp_v}^u = 1$ if the rule (s,t,p_v) exists.
- $s_{tp_v}^u \in \{0,1\}$, where $s_{tp_v}^u = 1$ if the wildcard rule $(*,t,p_v)$ exists.
- $t_{sp_v}^u \in \{0,1\}$, where $t_{sp_v}^u = 1$ if the wildcard rule $(s,*,p_v)$ exists.

- $d^u_{p_v} \in \{0,1\}$, where $d^u_{p_v} = 1$ if p_v if the default port.
- $o^u_{st} \in \{0,1\}$, where $o^u_{st} = 1$ if the wildcard rule for the source s has higher priority than the one for the destination t.
- $\overline{o^u_{ts}} \in \{0,1\}$, where $\overline{o^u_{ts}} = 1$ if the wildcard rule for the destination t has higher priority than the one for the source s. By definition, $o^u_{st} = 1 - \overline{o^u_{ts}}$.

We want to minimize the power consumption of the network (4.8).

$$\min \sum_{(u,v)\in A} \left(P^{\text{IDLE}}_{uv} x_{uv} + P^{\text{LOAD}}_{uv} \frac{f_{uv}}{C_{uv}} \right) \tag{4.8}$$

Flow conservation is ensured via the following set of constraints:

$$\sum_{v\in N^+(u)} f^{st}_{uv} - \sum_{v\in N^-(u)} f^{st}_{vu} = \begin{cases} 1 & \text{if } u = s, \\ -1 & \text{if } u = t, \\ 0 & \text{else} \end{cases} \qquad \forall u \in V, (s,t) \in \mathcal{D} \tag{4.9}$$

The sum of the flows on a link cannot exceed its capacity. Moreover, if the link is disabled, no flow can be forwarded on it (4.10).

$$f_{uv} = \sum_{(s,t)\in\mathcal{D}} D^{st} f^{st}_{uv} \quad \leq C_{uv} x_{uv} \qquad \forall (u,v) \in A \tag{4.10}$$

The following sets of constraints are similar to the ones presented for Comp-LP. The principal change is that the table is not longer an input of the problem and depend on the routing of the demands. Thus, if a demand (s,t) is forwarded on the link (u,v), there must exists a corresponding rule on the router u (4.11). Moreover, a non aggregated rule can only exists if the demand (s,t) is forwarded on the link (u,v) (4.12).

$$r^u_{stp_v} + s^u_{tp_v} + t^u_{sp_v} + d^u_{p_v} \geq f^{st}_{uv} \qquad \forall (u,v) \in A, (s,t) \in \mathcal{D} \tag{4.11}$$

$$r^u_{stp_v} \leq f^{st}_{uv} \qquad \forall (u,v) \in A, (s,t) \in \mathcal{D} \tag{4.12}$$

The total number of rules in the table of a router u cannot exceed its capacity C_u

$$\sum_{v\in N^+(u)} \left(d^u_{p_v} + \sum_{(s,t)\in\mathcal{D}} r^u_{stp_v} + \sum_{t\in\mathcal{T}} s^u_{tp_v} + \sum_{s\in\mathcal{S}} t^u_{sp_v} \right) \leq C_u \qquad \forall u \in V \tag{4.13}$$

We limit to one the number of default port (4.14), the number of wildcard rules for one source (4.15) and for one destination (4.16).

$$\sum_{v \in N(u)} d^u_{p_v} \leq 1 \qquad \forall u \in V \tag{4.14}$$

$$\sum_{v \in N(u)} s^u_{tp_v} \leq 1 \qquad \forall u \in V, t \in \mathcal{T} \tag{4.15}$$

$$\sum_{v \in N(u)} t^u_{sp_v} \leq 1 \qquad \forall u \in V, s \in \mathcal{S} \tag{4.16}$$

For a given demand (s,t) routed on a link (u,v_1), if a matching wildcard rule exists with different port than v, the table must contain the unaggregated rule (s,t,p_{v_1}) or another wildcard rule, $(s,*,p_{v_1})$ or $(*,t,p_{v_1})$, with a higher priority.

$$r^u_{stp_{v_1}} + s^u_{tp_{v_1}} \geq t^u_{sp_{v_2}} - 1 + f^{st}_{up_{v_1}} \qquad \forall u \in V, (s,t) \in \mathcal{D}, v_1 \neq v_2 \in N^+(u) \tag{4.17}$$

$$r^u_{stp_{v_1}} + \overline{o^u_{ts}} \geq t^u_{sp_{v_2}} - 1 + f^{st}_{up_{v_1}} \qquad \forall u \in V, (s,t) \in \mathcal{D}, v_1 \neq v_2 \in N^+(u) \tag{4.18}$$

$$r^u_{stp_{v_1}} + t^u_{sp_{v_1}} \geq s^u_{tp_{v_2}} - 1 + f^{st}_{up_{v_1}} \qquad \forall u \in V, (s,t) \in \mathcal{D}, v_1 \neq v_2 \in N^+(u) \tag{4.19}$$

$$r^u_{stp_{v_1}} + o^u_{st} \geq s^u_{tp_{v_2}} - 1 + f^{st}_{up_{v_1}} \qquad \forall u \in V, (s,t) \in \mathcal{D}, v_1 \neq v_2 \in N^+(u) \tag{4.20}$$

Finally, we remove cyclic order dependencies between rules.

$$1 \leq o^u_{s_1t_1} + \overline{o^u_{t_1s_2}} + o^u_{s_2t_2} + \overline{o^u_{t_2s_1}} \leq 3 \qquad \forall u \in V, s_1 \neq s_2 \in \mathcal{S}, t_1 \neq t_2 \in \mathcal{T} \tag{4.21}$$

Both linear programs run for small networks. In particular, we were able to obtain optimal solutions for the `atlanta` network from SNDLib, which has 15 nodes and 22 bi-directional links, see Section 4.4. However, the running time increases very quickly as the Energy Aware Routing problem is NP-Hard [GMM12]. Thus, we propose efficient heuristic algorithms for larger networks in the following section.

4.3 Heuristic Algorithms

As the linear programs proposed in the previous section do not run for medium and large networks, we propose here efficient heuristic algorithms. The problem can be decomposed into three sub-problems:

- First, the *compression problem* consists in reducing the size of a single table by using aggregation rules: the default rule for default port compression, and, additionally, source or destination rules for the multi-field compression, see Chapter 3.
- Second, the *routing problem* goal is to compute and assign a path in the network for each demand while respecting the link and forwarding table capacities.
- Last, the *energy saving problem* goal is to shut down a maximum number of links while maintaining a valid routing in the network for all the flows.

The heuristic algorithm is thus composed of three different modules designed to solve these subproblems. For every demand, using the routing module, we compute a path and the corresponding set of rules to install on the nodes. Whenever a node become overloaded, the compression module is called upon that node. Once all demands have been assigned to a path, the energy saving module is called.

4.3.1 Routing module

We propose an efficient routing heuristic using a weighted shortest-path algorithm with an adaptive metrics. When several routes are possible for flow, we select the one using the less loaded equipment, links, and routers, as measured by our metrics. The intuition is two-fold:(i) we want to avoid sending new flows to a router with a very loaded routing table, if there exists an alternative path using routers with less loaded routing tables (ii) load balancing the traffic over the multiple possible paths is currently done in data centers to avoid overloading links. More emphasis can be given to one or the other thanks to two parameters in the weight computation, named α and β.

For every flow (s, t, d), we first build a weighted directed graph (digraph) $G_{st} = (V, A_{st}, w)$, where, for every $(u, v) \in A_{st}$, w_{uv} is the weight of link

(u,v). G_{st} represents the residual network after having routed the previously routed flows:

- G_{st} is a subgraph of G where an arc (u,v) is removed if its capacity is less than d or if the flow table of the router u is full and does not contain any wildcard rule for (s,t,p_v) (where p_v represents the output port of u towards v). Note that, when a table is full and compressed, a node u has only one outgoing arc (to the node v), corresponding to the first existing rule of the form $(s,*,p_v)$, $(*,t,p_v)$ or to the default rule $(*,*,p_v)$. As more tables get full, the number of nodes with only one outgoing arc increases, reducing the size of the graph.

- The weight w_{uv} of a link depends on the overall flow load on the link and the table's usage of router u. We note w^c_{uv} the weight corresponding to the link capacity and w^r_{uv} the weight corresponding to the rule capacity. They are defined as follows:

  $$w^c_{uv} = \frac{\mathcal{F}_{uv}}{C_{uv}}$$

  where C_{uv} is the capacity of the link (u,v) and $\mathcal{F}_{uv}$ the total flow load on (u,v). The more the link is used, the heavier the weight is, which favors the use of lower loaded links allowing load-balancing. And

  $$w^r_{uv} = \begin{cases} \frac{|R_u|}{C_u} & \text{if } \nexists \text{ wildcard rule for } (s,t,v) \\ 0 & \text{otherwise} \end{cases}$$

  where R_u is the current set of rules for router u. Recall that S_u is the maximum number of rules which can be installed in the routing table of router u. The weight is proportional to the usage of the table. Note that $w^c_{uv} \in [0,1]$ and $w^r_{uv} \in [0,1]$. They measure the percentages of usage of link (u,v) and the routing table of router u.

  The weight w_{uv} of a link (u,v) is then given by:

  $$w_{uv} = 1 + \alpha w^c_{uv} + \beta w^r_{uv} \tag{4.22}$$

  The α and β, with $\beta = 1-\alpha$, parameters allow for a tuning between link and node capacity emphasis. The additive term 1 is used to provide the shortest path in terms of the number of hops when links and routers are not used (i.e., when $w^c_{uv} = 0$ and $w^r_{uv} = 0$ for all $(u, v) \in A_{st}$). This term could be replaced by the delay to traverse link (u, v) to obtain the shortest paths in terms of delay. When the links and routers are used, we take into account their usage. Note that $w_{uv} \leq 2$. This ensures that $l(p) \leq 2 \times l(p^*)$, where p is the path found by the routing module, p^* is the unweighted shortest path and $l(p)$ the number of hops of path p (indeed, $l(p) \leq w(p)$ as $w_{uv} \geq 1$, $w(p) \leq w(p^*)$ as p was selected, and $w(p^*) \leq 2\ l(p*)$, where $w(p)$ is the sum of the weights of the links of path p). This means that no path longer than twice the current available shortest path is selected.

When (G_{st}, w) is built, we compute a route for the flow by finding the shortest path between s and t in the digraph minimizing the weight w. If no such path exists, the module return that no feasible routing was found. We also compute the set of non-wildcard rules to install on the nodes to ensure a valid routing.

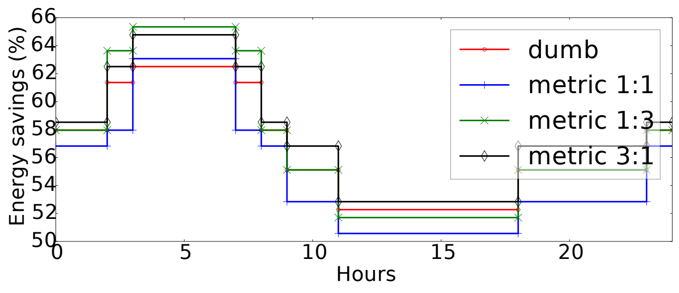
Setting the parameters of routing module. The metric Equation (4.22) to find a path for each demand combines link usage and link capacity with table capacity and table usage. The importance of links is given by the parameter α and the one for the tables by β. If α is larger than β, we give more weight to links. In Figure 4.1, we compare the effect of giving a higher priority to one or the other by changing the weight α or β. In particular, we provide results for values of $\alpha : \beta$ 1:1, 3:1, and 1:3. We tested other values which are not presented here for clarity of the plots. We also compare the metric with a simple metric, called *dumb*, where all links have a weight of one.

On `zib54`, the use of *dumb* and *metric 1:1* allow to shutdown between 40% and 50% of the links, while the use of metric 3:1 and 1:3 allow to shutdown between 48% and 56% of the network. The same behavior can be observed on the `ta2` network where between 48% and 56% of the network is shutdown with the *metric 1:1*, 52% and 56% for the dumb metric, 54% and 61% for the metric 3:1 and 56% and 60% for the metrics 1:3. We also observe that during the off peak hours, the metrics 3:1 gives better results than the metrics 1:3 while it is the other way around for the peak hours. For `germany50`, the difference between metrics is smaller, but the metric 3:1 is almost always the best one.

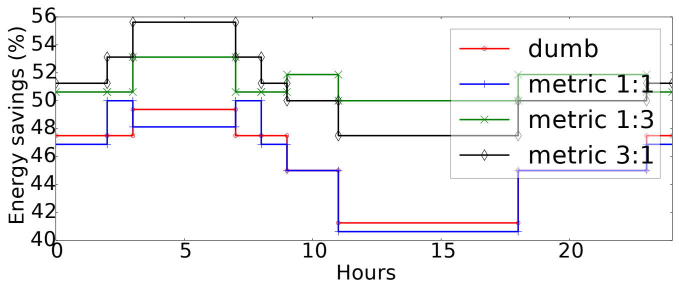
To summarize, for the three networks, the best metrics are 3:1 and 1:3. *We thus choose one of the two,* metric 3:1*, as the default metric in the*

Algorithm 2: Finding a path for a flow

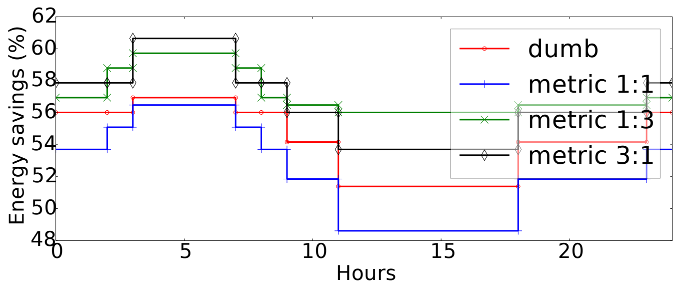
```
Input: A flow $(s, t, d)$,
a digraph $G = (V, A)$,
rule space capacity $S_u \; \forall u \in N$,
set of rules $R_u \; \forall u \in N$,
link capacity $C_{uv} \; \forall a \in A$,
flow $\mathcal{F}_{uv}, \forall a \in A$
Output: A path for $(s, t, d)$
1  Create a weighted digraph $G_{st} = (V, A_{st} = \emptyset, W)$;
2  foreach $(u, v) \in A$ do
3      if $C_{uv} - \mathcal{F}_{uv} \geq d$ then
4          if $\exists$ wildcard rule for $(s, t, p_v)$ then
5              add edge $(u, v)$ to $G_{st}$;
6              $w^r_{uv} = 0$;
7              $w^c_{uv} = \mathcal{F}_{uv}/C_{uv}$;
8              $W_{uv} = 1 + \alpha w^r_{uv} + \beta w^c_{uv}$;
9          else if $|R_u| < S_u$ then
10             add edge $(u, v)$ to $G_{st}$;
11             $w^r_{uv} = |S_u|/R_u$;
12             $w^c_{uv} = \mathcal{F}_{uv}/C_{uv}$;
13             $W_{uv} = 1 + \alpha w^r_{uv} + \beta w^c_{uv}$;
14 return weighted shortest path between $s$ and $t$ in $G_{st} = (V, A', W)$
```



(a) `germany50`



(b) `zib54`



(c) `ta2`

Figure 4.1: Energy savings for the different metrics with EARC-H-Direction. For metric $\alpha : \beta$, α represents the weight of the links and β the weight of the table (See Section 4.3.1).

remaining of the chapter.

Note that the Minnie algorithm, presented in Chapter 5, use the same routing module.

4.3.2 Energy savings module

The energy savings module uses a greedy approach to select the links to switch off. It tries to remove in priority links that are less loaded and to accommodate their traffic on other links to reduce the total number of active links.

The algorithm is simple. We start with the full network. We launch the routing module to try to find a feasible routing for all the demands. If such a routing exists, we try to remove the edge with the lowest load. We then re-launch the routing module on the network without the considered edge. If a feasible routing is found, we continue and try to switch off another edge. If no feasible routing is found, we put back the edge, and we try to remove the edge with the second lowest load. An edge, which was selected and could not be removed, is not considered anymore in the following of the algorithm. The algorithm stops when all edges have been selected once.

4.3.3 Compression module

The compression module is called whenever a table is full. Different levels of compression can be offered: default rules only, or wildcard aggregation on source or destination. Optimal and heuristic solutions are presented in Chapter 3.

4.4 Energy savings

In this section, we study the energy saved over multiple periods of time and the four following networks: `atlanta`, `germany50`, `ta2` and `zib54`. We compare the results obtained for the different solutions proposed to solve the EARC problem, the EAR problem without compression and Classical Routing (CR) (without energy).

For the power parameters, we look at the Powerlib database [Van+12] that collects representative data for major network equipment such as routers, switches, transponders. In this database, the scope of the values for the maximal power is huge, going, as an example, from 10 W to 9000 W for IP router components. Therefore, in order to present results that do not rely on a specific equipment from a specific vendor, we choose for the parameters of the power model a classical ON-OFF power model. Several other papers

Compression kind	Name	Short name in figures	Routing	Energy	Compression algo
default port	EARC-LP-Default		LP (Section 4.2.1)		
multi-field	EARC-LP-Multi		LP (Section 4.2.2)		
default port	EARC-H-Default	EARC-Default	Heur		Opt. Comp-Default (Section 3.2.1)
multi-field	EARC-H-LP		Heur		LP Comp-LP (Section 3.2.2)
multi-field	EARC-H-Greedy	EARC-Greedy	Heur		Heur Comp-Greedy (Section 3.2.3)
multi-field	EARC-H-Direction	EARC-Dir	Heur		Heur Comp-Direction (Section 3.2.4)
none	EAR		yes	yes	none (but no limit on the number of rules)
none	EAR-with-limit		yes	yes	none (with limit on the number of rules)
none	Classic Routing	CR	yes	no	none (but no limit on the number of rules)

Table 4.1: Names of the solutions to solve the EARC problem (and of the EAR problem without compression for comparison).

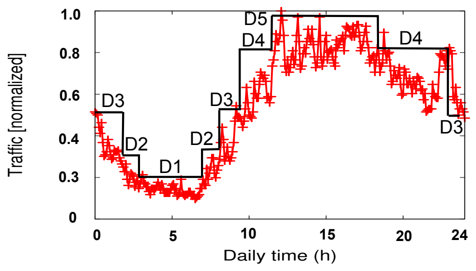
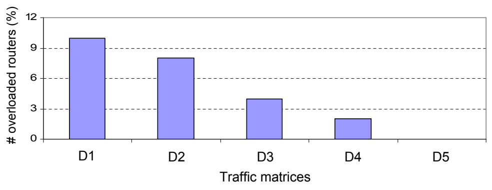


Figure 4.2: Daily traffic in multi-period

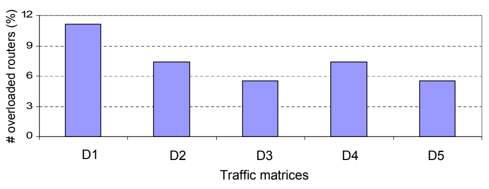
are dealing with this same power model, and among others, we can cite the very well known and most-cited paper in this area: [Cha+08].

The different solutions are summarized in Table 4.1. Unless specified, the limit of the forwarding table is 750 rules. We considered a typical daily pattern of traffic as shown in Figure 4.2. Data come from a typical France Telecom link. For each network considered, we rescale the traffic based on the traffic matrices provided by SNDib. We then divide the day into five periods, with different levels of traffic as shown in Figure 4.2. D1 represents the off peak hours with the least amount of traffic on the network and D5 the peak hours. We choose a small number of periods as network operators prefer to carry out as few as possible changes to configurations of their network equipment to minimize the chance of introducing errors or producing routing instability. Moreover, most of the energy savings can be achieved with a very small number of configurations, see for example [Ara+16]. We can not disable any router since they are all transmitters and receivers of traffic in the matrices considered. Energy savings is thus computed as the number of links to sleep divided by the total number of links of the network ($|E|$).

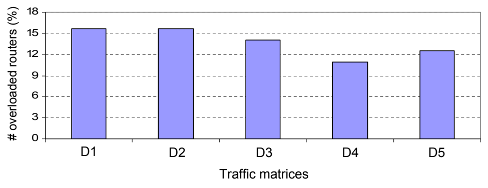
4.4.1 The need for more space



(a) `germany50`



(b) `zib54`



(c) `ta2`

Figure 4.3: Number of overloaded routers in three networks with unlimited rule-space algorithm

In Figure 4.3, we show the number of overloaded routers (with more than 750 installed rules) when applying the heuristic proposed in [GMM12] for Energy Aware Routing. This EAR heuristic does not take into account the table size constraint. As a result, we see that for almost every traffic patterns (except for D5 on `germany50`), an EAR needs more than 750 rules to be deployed. In `germany50`, up to 10% of the devices are overloaded. For `zib54`, this number goes up to 11% and 16% for `ta2`. This confirms that to be able to deploy energy policies on a SDN, the table size problem needs to be resolved.

4.4.2 Optimal vs. Heuristic solution

We compare for a small network, `atlanta` (15 links and 44 links), the solutions using linear programming and heuristic algorithms. We considered solutions for different rule capacities on routers: 100, 750 and 2000 rules.

Both linear programs, LP-Default and LP-Multi, proposed in Section 4.2 can be run on the `atlanta` network (but not on larger networks such as `germany50`, `zib54` and `ta2`). As expected, LP-Multi, which solves the problem using more complex wildcards, has a longer execution time as it has more

Table 4.2: Energy savings (in %) and computation times (in millisecond) for the ILPs and the heuristics on the `atlanta` network

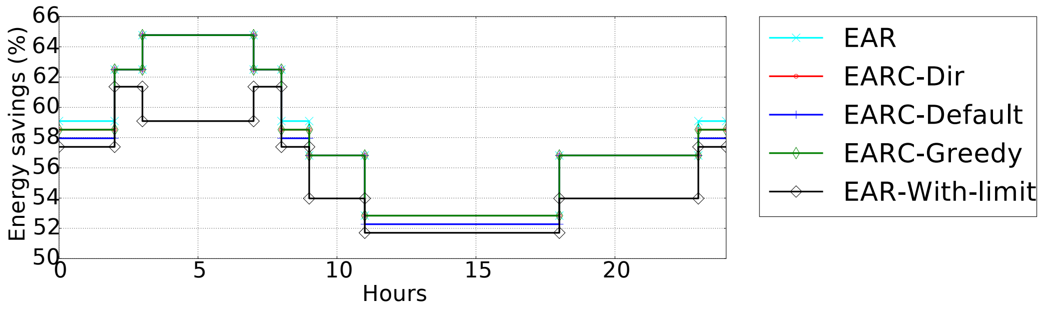
Rule capacity	EARC-LP-Default		EARC-LP-Multi		EARC-H-Direction		EARC-H-LP	
	savings	time	savings	time	savings	time	savings	time
100	52.27	641 940	52.27	694 302	40.91	~ 14	40.91	3381
750	52.27	33 830	52.27	486 759	40.91	~ 14	40.91	3311
2000	52.27	23 640	52.27	487 386	40.91	~ 14	40.91	3300

variables: around 8 min for 750 and 2000 rule capacities, to be compared with 23 s and 33 s for EARC-LP-Default. For a limit of 100 rules, both ILPs see an aggravated execution time, with a sharper increase for LP-default. Both LPs find the same optimal solution, with a savings of 52.27%. This is because `atlanta` is a small network with nodes of small degrees. The need for compression is not high, and both levels of compression achieve the same results.

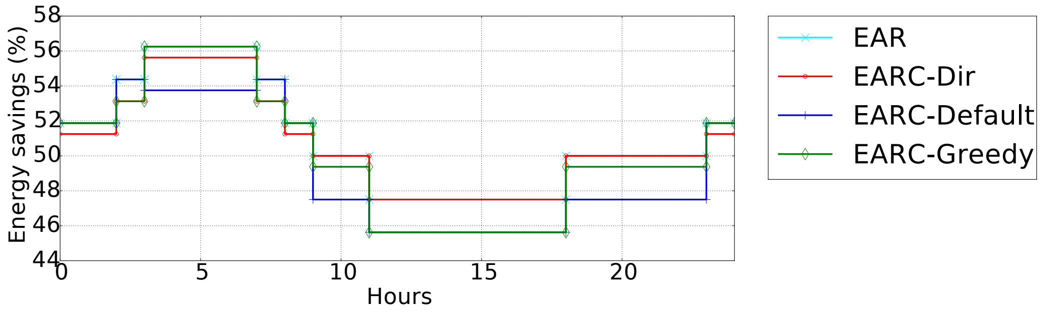
We ran two heuristic algorithms with two different compression modules proposed in Section 3.2, EARC-H-Direction with the Comp-Direction heuristic and EARC-H-LP, which solves optimally the compression problem each time a table has to be compressed when flows are routed. Both heuristics provide solutions of the same values, 40.91% of energy savings. However, the computation times are a lot higher for EARC-H-LP: more than 3 s compared to 14 ms for the heuristic. The computation time will prevent us from using EARC-H-LP on larger networks. As a matter of fact, we recall the compression time of tables using the LP increases exponentially, see Figure 3.4. However, we observe that both solutions provide the same level of energy savings for the three rule capacities. The EARC-H-Direction heuristic is very efficient, and we use it to get solutions for larger networks.

4.4.3 Energy savings during the day

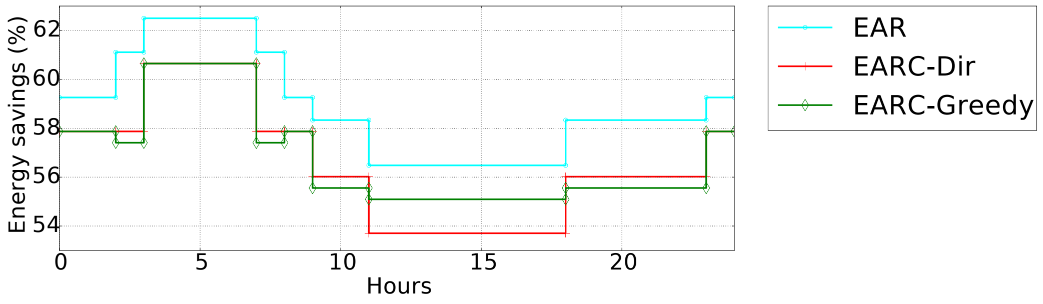
In Figure 4.4, we compare the multiple solutions proposed for the compression module. We also check the possibility of an SDN routing without compression (corresponding to a simple *EAR*). The ILP is not considered in the comparison as the networks are too big to be optimally resolved in an acceptable time.



(a) `germany50`



(b) `zib54`



(c) `ta2`

Figure 4.4: Energy savings of the different heuristics during the day with a limit of 750 rules.

Need for compression First, we see that, as the networks grow in size, not all heuristics give a valid SDN routing. No compression is needed to find a valid SDN routing on `germany50`. However, *it is impossible to find a routing satisfying the capacity constraint for* `zib54` *and* `ta2` *without using a compression algorithm*. Moreover, *multi-field compression should be used to find a valid routing for* `ta2`. Indeed, it is impossible to find a valid routing for `ta2` while using only default port compression.

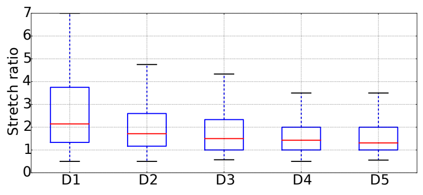
Compression impact On `germany50`, all heuristics give similar results between 52% for the peak hours and up to 65% during the night. They are all small within a margin of about 2% from one another. The EARC-H-Greedy and EARC-H-Direction heuristics show the best results and no compression gives the worst ones in all periods.

For the `zib54` network, the difference between the heuristics is a little bit more visible. Between 46% and 56% is saved during the day. Once again, either the EARC-H-Greedy or EARC-H-Direction heuristics gives the best results depending on the periods. The only exception is during the D2 periods, where the EARC-H-Default compression shut about 1% more links than the other two heuristics.

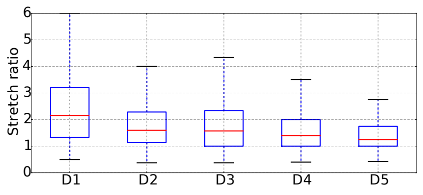
Finally, in the `ta2` network, the EARC-H-Greedy heuristic saves a little bit more energy than the EARC-H-Direction one as the former saves almost 2% more than the latter.

The amount of saved energy by the heuristics for each network is different. The explanation is that the order in which each link is extinguished depends on its charge. A small change in the routing thus can affect the total energy saved.

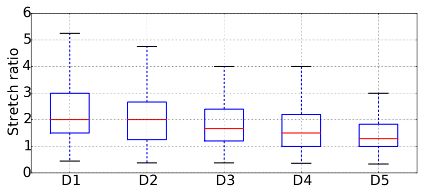
EAR vs. EARC We compare the results of the proposed solutions with the one of the classic EAR approach in which no limit on the number of rules is considered. We show that *by using an efficient way to route demands and compress forwarding tables, it is possible to save almost as much power consumption as the EAR approach* (curve named *No Limit* in Figure 4.4). Indeed, we see that for the `zib54` network, we succeeded to save the same amount of energy when using the best of all solutions. The solution EARC-H-Direction alone is very close to the EAR one. Only half a percent of energy is lost for some periods of time. On `germany50`, the results of the heuristics are almost as good. For some periods of time, no solutions can do as well as



(a) `germany50`

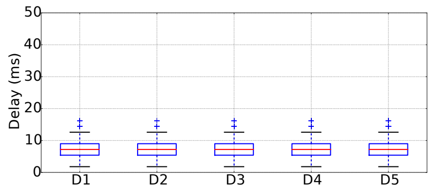


(b) `zib54`

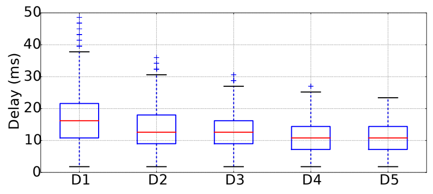


(c) `ta2`

Figure 4.5: Stretch ratio of the paths given by a EARC solution (EARC-H-Direction) compared to the one given by a classic routing (without energy savings) on the `germany50` network with different traffic matrices.



(a) Classic Routing



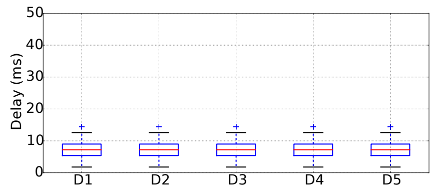
(b) EARC

Figure 4.6: Average delay by path on the `germany50` network

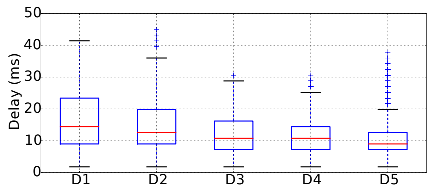
EAR, but the difference again is only of half a percent. In general, the results of EARC-H-Greedy are withing 1% of the one of EAR. For the network `ta2`, the difference between EAR and our solutions is higher but stays with 2%.

4.4.4 Path lengths

As we shut down links, we remove some shortest paths in the network and thus raise the minimum delay between nodes. To study this effect, we look at the length of the paths in our EAR solutions and compare it to a routing obtained not using the energy saving module. For these comparisons, we use

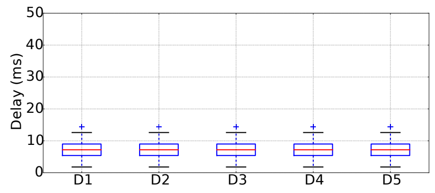


(a) Classic Routing

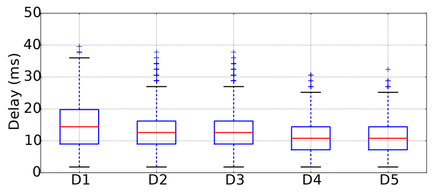


(b) EARC

Figure 4.7: Average delay by path on the `zib54` network.



(a) Classic Routing



(b) EARC

Figure 4.8: Average delay by path on the `ta2` network.

the Direction heuristic.

In Figure 4.5, we show the distribution of the stretch ratio of the path used in EARC-H-Direction compared to a CR. The first observation is that the behavior is similar for the three topologies: the median stretch is about 2 in the off-peak hour period (corresponding to the demand D1) and decreases to about 1.3 in the peak hours (demand D5). The explanation is that, as expected, in the off-peak hours, a large number of links can be switched off, and the paths are the longest. For larger demands, more links are on, and the stretch decreases.

Note that the median value is not very high. However, the third quartile value of the off-peak hours is quite high: 7, 6 and 5.25 for `germany50`, `zib54`, and `ta2`, respectively. These values are mostly due to paths of small lengths stretched all the way trough the network to attain their destination (corresponding for example to nodes linked by a switched-off edge). Nevertheless, we show below that these somehow large values of stretch do not cause a problem of too large delays on the networks.

4.4.5 Delays

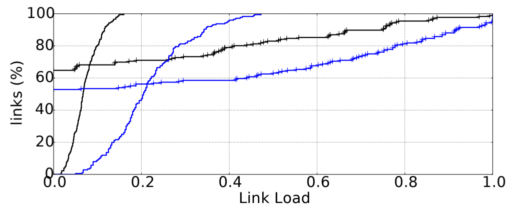
In Figures 4.6, 4.7 and 4.8, we show the delay of the paths in the three networks, for both the classical routing and an EARC solution (EARC-H-Direction). We consider an optical network in which the delay is proportional to the distance [Cho+07], and we used the distances given by the geographical coordinates in SNDlib for the `germany50` network. We got an average value of 1.8 ms per link. Since the coordinates are not given for the other two topologies, we used the same average value for `zib54` and `ta2`.
The delays for the classical routing are similar for the three networks with a median of 8 ms and a maximum of 15 ms during all periods. For the EARC solution, the values are much higher. Larger delays are shown during the off-peak hours as expected. The `germany50` network shows the biggest delays among the three topologies. The explanation is that more energy can be saved for this network. Its median delay is between 11 ms and 16 ms, and the maximum delay is below 50 ms. The delay on the two larger networks is slightly less impacted as fewer links can be turned off. The maximum delay observed on `zib54` and `ta2` is about 40 ms and the medians fluctuate between 14 ms and 9 ms for `zib54` and 14 ms and 10 ms for `ta2`.

Note, that the *maximum delay observed is always below* 50 ms. This is an important fact, as Service Level Agreement (SLA)s often choose this value as the maximum allowed delay for a route in a network [Gir+03]. Thus, even if new routes computed by our algorithms may experience sometimes a high value of stretch, this will not be a problem for network operators.

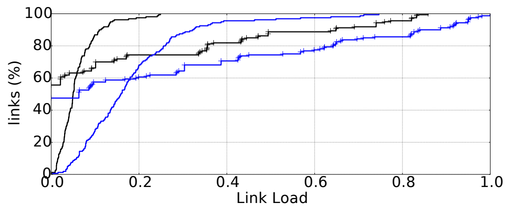
4.4.6 Link load

When we turn off links, we aggregate the flows on the remaining links. The load of them is thus increased. In Figure 4.9, we *compare the link load of all network links (switched off and switched on) for energy aware routing and classical routing*. In Figure 4.10, cumulative distributions are given considering only the switched on links. Results are provided only for off-peak traffic (D1) and rush hour traffic (D5) in the first figure for clarity reason, while all five demand matrices are considered in the other.

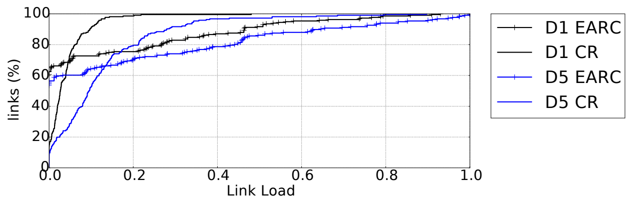
The first observation is the percentage of links with a null load (switched off links), e.g., for `germany50` 62% with the demand D1 and 54% with D5. The load on the remaining links is highly increased: for `germany50` again, we see that no link has a load higher than 15% for the CR for the D1 (higher



(a) `germany50`

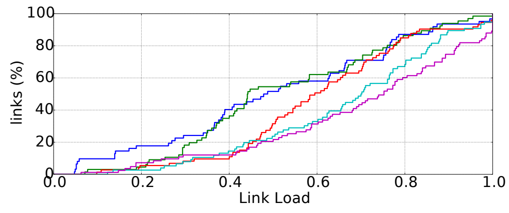


(b) zi54

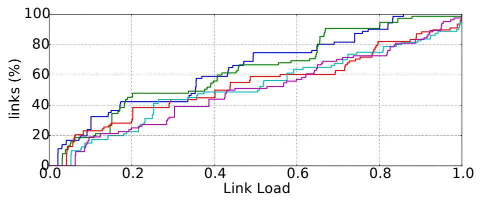


(c) `ta2`

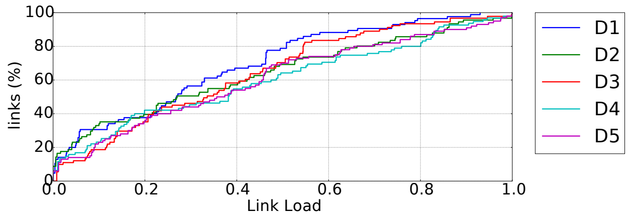
Figure 4.9: Comparison of the cumulative distribution function of the link load for EARC and CR. Results for off peak traffic (D1) and rush hour traffic (D5) are provided.



(a) `germany50`



(b) zi54



(c) `ta2`

Figure 4.10: Cumulative distribution function of the link load of the *swichted on links* using EARC for the five demand matrices (D1 is off peak traffic and D5 is rush hour traffic.)

than 50 % for D5), when 45% of the links have a load higher than this value for EARC (40% for D5). Similarly, for `zib54` and `ta2`, more than 80% of the links have a load smaller than 10% for D1 for CR and of 30% for D5, when more than 50% of the switched on links have a load higher than 50% for EARC.

Note that, in the `germany50` network, there is a very notable difference between the D1 and D5 period. In the off-peak hours, only 30% of the switched-on links are above 75% compared to 52% in the peak hours. In the other two networks, the difference between periods as the difference as the range of energy savings is smaller. For `zib54`, in the off-peak hours, the maximum link utilization is 86%, and the minimum is 2% while in the peak hours, 19% of the link are above this usage and the minimum is 6%.

4.5 Conclusion

In this chapter, we studied the impact of the table size constraints on the energy savings possible on an SDN network. To the best of our knowledge, we are the first to study the Energy Aware Routing with Compression Problem. We modeled the problem as an ILP, for both default port and multi-field compression scheme. We also proposed efficient heuristic algorithms for large networks using the heuristics solutions presented in 3. We showed, using real traffic traces, that using wildcard rules allows for almost as much energy savings as in the case of classic EAR without capacities on the forwarding tables. We also evaluate the impact of the proposed solutions on path delay. We demonstrated that, if the delay is inevitably increased, the maximum delay always stays below typical values given by SLAs.

CHAPTER 5 Minnie

Contents

5.1 Introduction

In this chapter, we look at the *Compression Problem* in the context of classical routing in data center networks. We present MINNIE, a routing solution using multi-field compression to circumvent the table size constraints imposed by TCAMs. It can be seen as a dynamic variant of the heuristic presented in Chapter 4 without the energy saving module. It routes the traffic and compresses routing tables to satisfy link capacity and routing table size of the different forwarding devices. We validate our algorithm on multiple well-known data center topologies in Section 5.2. We show that one can deploy networks with up to 1 million flows using MINNIE by carefully choosing a threshold for compression. We further validated MINNIE using a testbed emulating a $k = 4$ Fat-Tree data center topology in Section 5.3. We demonstrate on the one hand that even with a small number of clients the limit concerning

		Compression ratio	Compression frequency	# Rule	Delay on packets		Execution time of Minnie in average (ms)		Additional results
Simulations		Section 5.2.2 Figure 5.3	Section 5.2.2 Figure 5.4	Section 5.2.2 Figure 5.5 Section 5.2.2 Table 5.2			Routing Section 5.2.2 Figure 5.6a	Compression Section 5.2.2 Figure 5.6b Figure 5.7	Comparison with XPath Section 5.2.2 Table 5.3
Experiments	Scenario 1 (LLS) Section 5.3.2	Table 5.5	Table 5.6	Table 5.4 Figure 5.10	First Figure 5.14 Figure 5.15	Subsequent Figure 5.16 Figure 5.17	Compression Figure 5.11		SDN control path Section 5.3.2 Figure 5.13
	Scenario 2 (HLS) Section 5.3.2	Table 5.7		Figure 5.19	Figure 5.18a	Figure 5.18b Figure 5.20 Section 5.3.2			Software vs. hardware Section 5.3.2 Figure 5.21

Table 5.1: Summary of the results for Minnie

the number of rules is reached if no compression is performed, increasing the delay of new incoming flows. MINNIE, on the other hand, reduces the number of rules needed, with no packet losses, nor noticeable additional delays using TCAM. A summary of the simulation and experimental results of this section can be found in Table 5.1

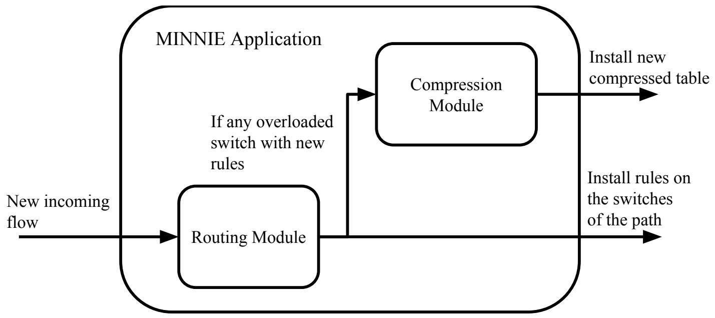

MINNIE Application
Compression Module
Install new compressed table
If any overloaded switch with new rules
Install rules on the switches of the path
New incoming flow
Routing Module

Figure 5.1: MINNIE algorithm

5.1.1 Minnie

MINNIE, shown in Figure 5.1, is composed of two modules: a routing module using the routing heuristic presented in Section 4.3.1, and a compression module using the Comp-Direction approximation algorithm presented in Section 3.2.4.

Every time the controller detects a new flow, the routing module computes a path on the network along a set of rules to install on the switches of the path. Once the rules are installed, the compression module is called on any switch reaching its maximum capacity. Delaying the compression offers a reduced impact on the delay onto new packets, as we will see in Section 5.3.2. The details of deploying MINNIE in an SDN controller can be found in Section 5.3.1.

5.2 Simulations on data center topologies

In this section, we study the behavior of MINNIE through simulations for a wide variety of data center architectures. We first present the different scenarios, performance metrics, and data center architectures in Section 5.2.1.

We then demonstrate that Minnie works well for topologies of various sizes and structures in Section 5.2.2.

5.2.1 Simulation settings

We present in this section the different scenarios studied via simulations, the traffic patterns and metrics that will be evaluated. All simulations were carried out on a computer equipped with a 3.2GHz 8 Core Intel Xeon Central Processing Unit (CPU) and 64 GB of Random Access Memory (RAM).

Scenarios

We ran simulations under three different scenarios:

- **Scenario 1: No compression**. We only use the routing module of Minnie and fill up the routing tables without compressing them. This scenario serves as a baseline for measuring the efficiency of Minnie.
- **Scenario 2: Compression at the end of the simulation**. We compress the routing tables of every switch once at the end of the simulation, when all the forwarding rules have been stored assuming an unlimited capacity of the routing table. We use it to test the efficiency of the compression module of Minnie.
- **Scenario 3: Minnie (Dynamic compression at a fixed threshold)**. We validate Minnie with a threshold of 1000 rules, which represents the routing table limit. This scenario aims at testing Minnie in a scenario closer to real life. The capacity of 1000 rules has been chosen as it corresponds to the number of entries supported by the TCAM of typical switches such as Apollo 2 and Triumph 2 [Mod16]. The actual number ranges around couple of thousands to tens of thousands [Ste+12].

Traffic patterns

For all scenarios, we consider an all-to-all traffic in which every single server establishes a connection to all other servers. Each flow is constantly sending traffic. We consider this situation to test Minnie in the most extreme scenario in terms of number of flows, and thus, in terms of number of rules. Each flow is represented by a unique source-destination pair.

Data center architectures

To test the efficiency of Minnie, we considered state-of-the-art data center architectures: Fat-Tree [ALV08], VL2 [Gre+09], BCube [Guo+09] and DCell [Guo+08]. For each family of architecture, we considered topologies of different sizes hosting from few units to about 3000 end points. These end points can be either servers or IP subnets, grouping thousands of different machines. In the following, for simplicity, we often use the term *server* for both cases. The number of flows routed in the topologies can thus reach a few million.

The architectures considered during these simulations can be classified into two different groups:

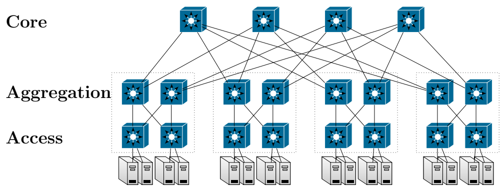
- **Group 1**, in which servers only act as end hosts includes Fat-Tree and VL2.
- **Group 2**, in which servers also act as forwarding devices (similarly to switches) includes BCube and DCell.

We detail below how we chose the different set of parameters to build these topologies like the number of switches or level of recursion.

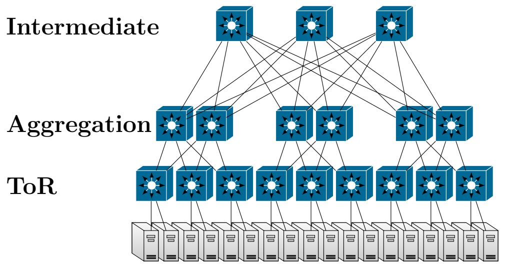
Fat-Tree. The Fat-Tree is one of the most well-known architectures. The switches are divided into three categories: core, aggregation, and access (or Top of the Rack (ToR)) switches. A k Fat-Tree is composed of k pods of k switches and $k^2/4$ core switches. Every switch possesses k ports. Inside a pod, aggregation and edge switches form a complete bipartite graph. Each core switch is connected to every pod via one of the $k/2$ aggregation switches. Every ToR switch has a rack composed of $k/2$ servers. A $k = 4$ Fat-Tree is shown as example in Figure 5.2a.

For our simulations, to build Fat-Trees with up to 3000 servers, we considered k values between 4 and 22.

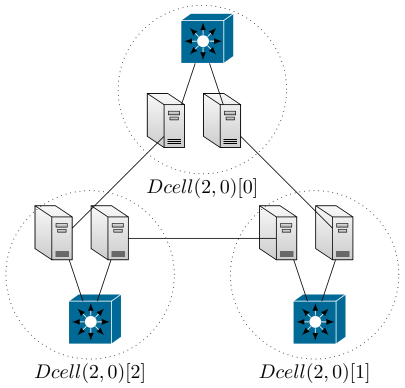
VL2. The VL2 architecture is also composed of three layers of switches: intermediate, aggregation and ToR switches. The intermediate and aggregation switches are connected to form a complete bipartite graph. Each ToR is connected to two different aggregation switches. Three parameters control the number of switches of each layer and the number of servers of the architecture: D_a represents the number of ports of an aggregation switch, D_i the number of ports of an intermediate switch and T the number of servers in



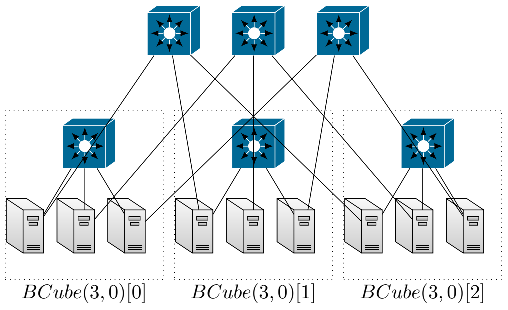
(a) Group 1: A Fat-Tree with $k = 4$ pods



(b) Group 1: A VL2 network with $D_i =$ 6-ports intermediate swiches, $D_a = 6$-ports aggregation switches and $T = 2$ servers per ToR



(c) Group 2: A DCell(2, 1), composed of 3 DCell(2,0)



(d) Group 2: A BCube(3, 1), composed of 3 $BCube(3,0)$

Figure 5.2: Example of topologies studied.

the rack of a ToR switch. Figure 5.2b shows a VL2$(D_a = 6, D_i = 6, T = 2)$. The topology has $D_a/2$ (3 in the example) aggregation switches, D_i (6 in the example) intermediate switches, $D_aD_i/4$ (9 in the example) ToR switches and $TD_aD_i/4$ (18 in the example) servers.

For our simulations, we chose the parameters of the topologies to ensure that every switch has the same number of ports, that is VL2$(2k, 2k, 2k - 2)$ for k between 2 and 11.

DCell. The DCell architecture is a topology in which both servers and switches act as forwarding devices. The topology is built recursively. The basic block is the level-0 DCell, DCell(n,0), where n servers are connected to a unique switch. From a DCell(n, l-1), composed of $s(n, l-1)$ servers, a DCell(n, l) can be built by connecting each server of a DCell(n, l-1) to a different DCell(n, l-1). This builds a DCell(n, l) containing $(s(n, l) + 1)$ DCell(n, l-1). For example, a DCell(2, 0) is composed of 2 servers ($s(n, 0) = n$) and to create a DCell(2, 1), as shown in Figure 5.2c, 3 DCell(2, 0) are interconnected.

In our simulations, we compare topologies with one level of recursion (referenced as $DCell(l = 1)$), with n between 1 and 54, and topologies with two levels of recursion (referenced as DCell(l=2)), with n between 1 and 7.

BCube. BCube is another architecture in which the servers also act as forwarding devices. Again, it is a recursive construction. The building block is a $BCube(n, 0)$, composed of n servers connected to a single switch. The level l being composed of multiple $l - 1$ levels. Unlike in the construction of DCell, in which the recursion connect servers together, the construction of BCube is done by connecting the servers via new switches. The number of switches added to make a BCube of level l is equal to the number of servers in a BCube of level $l-1$. Each switch is then connected to one server of every BCube of level $l - 1$ and each servers to $l + 1$ switches – see the BCube$(3, 1)$ in Figure 5.2d.

Like for DCell topologies, the same number of servers can be obtained with different levels of recursion. We consider levels of recursion up to 3.

5.2.2 Simulation results

In this section, we validate Minnie through simulations over the set of topologies described in Section 5.2.1. We demonstrate in this section that Minnie works well for different topologies and different sizes of data centers.

We first analyze the compression rates that can be obtained by compressing large tables. Then, we show that if tables are compressed all along the simulation as soon as the limit is reached, then the compression module is much more efficient and the compression ratio reaches 90% for some topologies. We then investigate the efficiency of MINNIE when considering around 1000 servers in multiple topologies. We show the efficiency of our method by comparing the results of MINNIE with XPath [Hu+15]. Finally, we present the routing and compression time of these different topologies.

Metrics To assess the efficiency of MINNIE, we measure the following metrics:

- Average compression ratio of compressed tables: *compression ratio* =

  $$1 - \frac{\textit{number of rules of a switch}}{\textit{number of flows passing through the switch}}$$

  Note that the compression ratio measures the efficiency of the compression algorithm. We thus do not consider tables, on which no compression event was performed (in particular empty tables), when we compute the average compression ratio.

- Number of compression events performed by a switch during the simulation.

- Number of flows passing through a switch (maximum and average over all switches).

- Number of rules per switch (maximum and average over all switches).

- Computation time for compressing a table and for routing a flow.

- Maximum number of servers which can be installed on a data center topology without going beyond a forwarding table size of 1000 rules.

For each family of topologies, we present the results for the three scenarios described in Section 5.2.1, referenced respectively as *No compression*, *Compression at the end* and MINNIE.

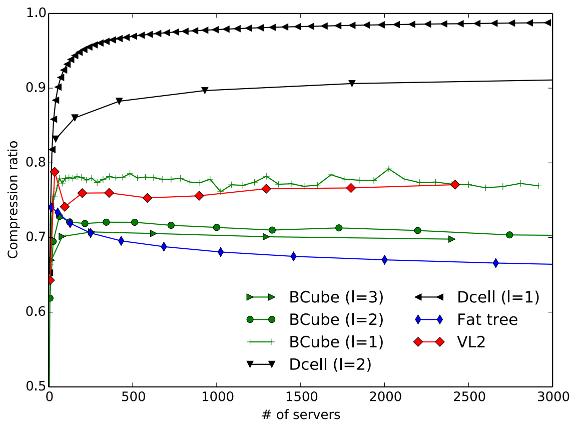


Figure 5.3: Average compression ratio on the different topologies in Scenario 2.

Efficiency of the compression module. The efficiency of the *compression module* of Minnie can be assessed from Figure 5.3 where we look at the average compression ratios of the *Compression at the end* scenarios. In this figure we observe that DCell, BCube and VL2 topologies follow a similar phenomenon. They all feature a sharp increase of the compression ratio when the number of servers is between 0 and 100: for example, the ratio raises from 62% to 84% for DCell(l=2). Then, for larger number of servers, the compression ratio levels off. On the other hand, Fat-Tree topologies have a different behavior and do not experience the increase phase ; the curve is almost flat all along the simulation. The higher ratio shown on DCell topologies is explained by the aggregation of flows on the few switches available in the topology. Combined with a few number of outgoing ports, the compression module can attain a very high compression ratio.

In the flat phase, compression ratios are between 60% and 80% for the three families BCube, VL2 and Fat-Tree, and even reach values between 85% and 99.9% for DCell. **In summary, the compression module of Minnie features a minimum of 60% savings in memory.**

Compression event frequency. In Figure 5.4, we observe the total number of compression events executed for the different topologies. Group 1 topologies reach a maximum of 516 compressions for the $k = 18$ Fat-Tree

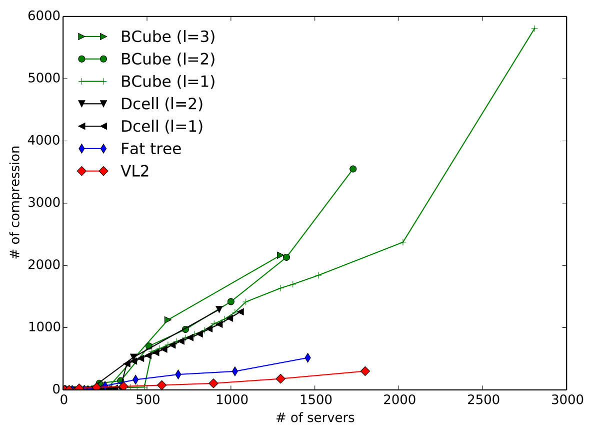
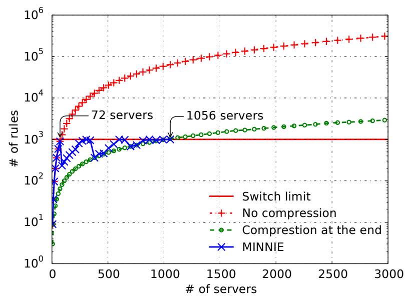


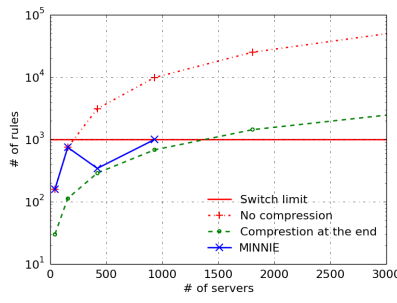
Figure 5.4: Number of compression executed for different topologies

(and 301 for VL2(20, 20, 18)). This represents an average of about one compression event per switch for the Fat-Tree topology and less than six compression events for VL2. However, Group 2 shows a higher number of compression events, with a maximum of almost 6000 compression events for a $BCube(53, 1)$ (in average, 54 compression events per forwarding device). This difference is due to the near saturation of most of the switches in Group 2 topologies. In these nearly saturated tables, the compression leaves a table that is close to the 1000 limit and thus, the table is compressed after the addition of a few new flows.

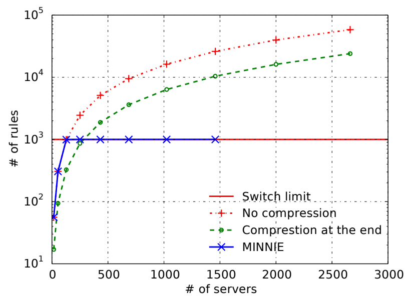
Efficiency of Minnie MINNIE is composed of a routing and a compression module. When the number of rules reaches the 1000 limit, MINNIE triggers the compression module. This dynamic behavior allows to efficiently route traffic *without overloading the routing tables* on topologies where the number of servers increases. Figure 5.5 presents the maximum number of rules on a device (a router or a server depending on the family of topology) as a function of the number of servers for the different families of topologies. We remark that the curve for MINNIE first follows the *No compression* one until reaching the 1000 limit. Indeed, during this first phase, MINNIE performs no compression at all as the limit is not attained. Then, MINNIE triggers compression regularly and manages to keep all routers' table below the limit of 1000. When performing compression, MINNIE has introduced wildcard



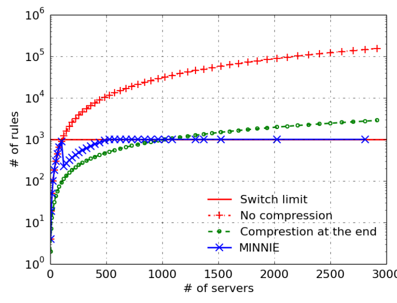
(a) DCell Topologies (l=1)



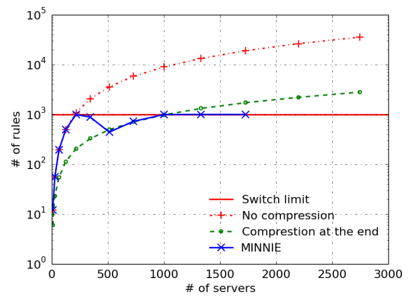
(b) DCell Topologies (l=2)



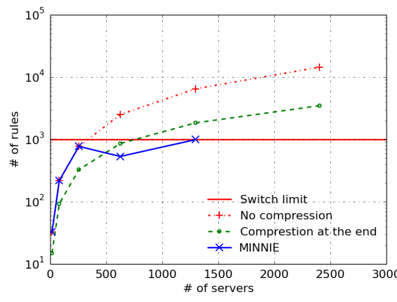
(c) Fat-Tree topologies



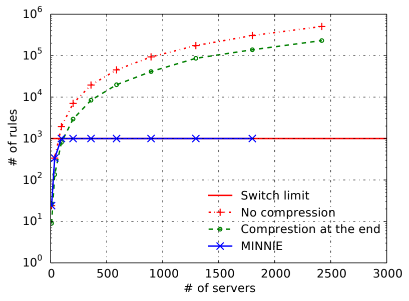
(d) BCube Topologies (l=1)



(e) BCube Topologies (l=2)]



(f) BCube Topologies (l=3)



(g) VL2 Topologies

Figure 5.5: Maximum number of rules on a forwarding device as a function of the number of servers for different data center architectures.

rules in the routing tables, and the new incoming flows will follow these paths in priority. Therefore, MINNIE deals with the same number of flows as *No Compression* with less than 1000 entries while *No Compression* needs between 10^4 and 10^6 entries. Note that some points for MINNIE are not depicted. Indeed, in Figure 5.5, we present only the results in which all the flows are routed without overloading the routing tables. As soon as one request cannot be routed and when the routing tables cannot be further compressed, the simulations are stopped.

This phenomenon can be clearly seen for DCell(l=1) topologies in Figures 5.5a. Without compression, only 72 servers can be deployed in a DCell$(8, 1)$ without overloading tables while MINNIE allows deploying 1056 servers with a DCell$(32, 1)$. This represents a 15 fold increase compared to *No compression*. The number of servers which can be deployed with DCell topologies having two levels of recursion (Figure 5.5b) is similar: 930 with a DCell(5, 2) when running MINNIE and less than 200 with *No compression*.

Another key observation is that Minnie can reach or even outperforms *Compression at the end* without exceeding the limit of number of rules. Indeed, if we consider for example Fat-Tree topologies in Figure 5.5c, without compression, the largest Fat-Tree which can be deployed with a rule limit of 1000 is a $k = 8$ Fat-Tree with 128 servers and 992 rules. With compression at the end, the number of servers which can be deployed would be around 256. However, we see that MINNIE succeeds in deploying a $k = 18$ Fat-Tree with 1458 servers without having overloading issues. This is a 6 fold increase compared to *Compression at the end*. **This is due to the fact that by compressing online, i.e., when flows are introduced, Minnie impacts the routing of the following flows**. Because of the metrics used in the routing module, the algorithm will prefer to select shortest paths using wildcards as they do not increase the number of rules. This allows better compression ratios.

The phenomenon also appears for BCube topologies (Figures 5.5d, 5.5e, 5.5f) and with a striking intensity for VL2 topologies (Figure 5.5g). When compressing at the end, up to 96 servers can be deployed without reaching the table size limit (and only 36 without compression). With MINNIE, this number can be pushed up to 1800 servers which represents **36 fold increase**.

Difference of behavior inside a family of topologies. We notice in Figure 5.3 and 5.5 a difference of behavior inside a family of topologies. For a given family of data centers, different topologies can host a similar number of servers. For example, DCell(32,1) and DCell(5,2) host around

1000 servers, as well as BCube(32,1), BCube(10,2) and BCube(6,3). But the behavior of these topologies is significantly different: for example, the average number of rules is 113 for a DCell(32,1) compared to 642 for a DCell(5,2). We see that the compression ratio of the family DCell(l=1) is higher (more than 95% when the number of servers is greater than 200) than the one of DCell(l=2) (more than 85% when the number of servers is greater than 200). **Hence, the choice of the best set of parameters for a given family of topologies is very important.** In order to answer this question, we study in the following section all these topologies with similar number of servers (around 1000).

Comparison of Minnie effect on topologies with 1000 servers. Table 5.2 sums up the effect of MINNIE on the different topologies with a similar number of servers (around 1000), hence a similar number of flows to route. We detail below the different parts of the table, highlighting the key conclusions to draw.

Topology characteristics. The first part of the table provides basic information about the topologies. **Even with a similar number of servers, the topologies are very different** in terms of number of switches (between 20 and 903), links (between 1056 and 5184) and average number of ports per switch (between 2.9 and 54.4).

Flows in the network. The second part of the table reports the number of flows introduced in the network during the simulation. These topologies behave very differently in terms of number of flows per device: the average number of rules ranges from 3734 to 216 000 and the maximum number of rules ranges from 7800 to 650 000. Two explanations can be given for these differences. First, the topologies have very different numbers of switches (from 20 to 864). Secondly, in the topologies of Group 2, servers also act as switches, and thus also host some rules, leading to a lower average number per device.

Compressing with Minnie. The third part of the table represents the effect of using MINNIE on the number of rules, average compression ratio and computation time. **Minnie succeeds to route the traffic on all the topologies without exceeding the limit of** 1000 **rules per device**

Topology	servers #	switches #	links #	Avg ports #	# flow per switch		Rule w/ comp #		Average Comp. Ratio	Computation time in average (ms)	
					Max	Average	Max	Average		Paths	Comp.
Group 1											
$k = 4$ Fat-Tree (64)	1024	20	1056	54.4	454 244	216 268	999	446	$\sim$ 99.60	0.17	13
$k = 8$ Fat-Tree (8)	1024	80	1280	19.2	649 044	61 030	999	323	$\sim$ 99.61	0.21	7
$k = 16$ Fat-Tree (1)	1024	320	3072	16	630 998	15 897	999	303	$\sim$ 98.42	0.30	5
VL2(16, 16, 14)	896	88	384	16	261 266	42 906	1000	673	$\sim$ 97.90	0.15	4
VL2(8, 8, 64)	1024	28	612	$\sim$ 41.1	423 752	161 499	1000	799	$\sim$ 99.45	0.19	11
VL2(16, 16, 16)	1024	88	1152	$\sim$ 17.5	276 575	56 040	1000	648	$\sim$ 98.39	0.18	4
Group 2											
DCell(32, 1)	1056	33	1584	$\sim$ 2.91	63 787	4893	1000	113	$\sim$ 97.23	0.09	2
DCell(5, 2)	930	186	1860	$\sim$ 3.33	11 995	5716	994	642	$\sim$ 87.84	0.19	2
BCube(32, 1)	1024	64	2048	$\sim$ 3.77	37 738	3734	999	329	$\sim$ 86.04	0.19	2
BCube(10, 2)	1000	300	3000	$\sim$ 4.62	10 683	4153	998	653	$\sim$ 80.85	0.25	2
BCube(6, 3)	1296	864	5184	4.8	7852	5184	991	831	$\sim$ 83.18	0.49	4

Table 5.2: Comparison of the behavior of MINNIE for different families of topologies with around 1000 servers each. For the Fat-Tree topologies, we tweak the number of clients per server to obtain 1024 "servers".

(maximum number of rules between 989 and 1000).

We also observe that with 1000 servers **Minnie allows to attain an average compression ratio higher than 80%**. This shows that considering the state of the forwarding table when routing increases the compression done by the wildcard rules. Compared with the *Compression at the end* scenario, we see a ratio increase between 20% and 30% for the Fat-Tree and VL2 topologies, and a smaller increase between 5 and 10% for BCube. This difference comes from the smaller amount of shortest path available in BCube compared to the Group 1 topologies. DCell topologies display close to no differences since flows were already highly aggregated in the other scenario.

As for the computation time we notice that **Minnie dynamically computes the route with a sub-millisecond delays** as the maximum average routing computation time is 0.49 ms for BCube(6,3). And finally, we can observe that **compressing the rules with Minnie will cost less than** 13 ms **delay in all topologies**.

Comparison with XPath. We compare MINNIE with another compression method of the literature, XPath [Hu+15]. XPath combines re-labeling and aggregation of paths. Each path is assigned an ID. Two paths can share the same ID if they are either convergent or disjoint but not if they are divergent. The assignment of IDs is then based on prefix aggregation. This method requires that, for every request in the data center, an application contacts the controller to acquire the corresponding ID of the path to its destination.

Topology	Number of rules	
	XPath	MINNIE
BCube(4, 2)	108	56
BCube(8, 2)	522	443

(a) Comparison with MINNIE for paths between servers

Topology	Number of rules		
	ToR to ToR		Server to Server
	XPath	MINNIE	MINNIE
$k = 8$ Fat-Tree	116	27	272
$k = 16$ Fat-Tree	968	116	6351
$k = 32$ Fat-Tree	7952	482	113 040
$k = 64$ Fat-Tree	64 544	1925	-
VL2(20, 8, 40)	310	135	138 354
VL2(40, 16, 60)	2820	1252	-
VL2(80, 64, 80)	49 640	22 957	-

(b) Comparison with MINNIE: for paths between servers and paths between level 1 switches

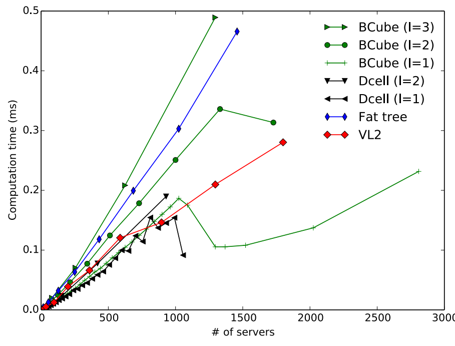
Table 5.3: Comparison of the maximum number of rules on a switch between XPath and Minnie (between servers or ToRs).

In Table 5.3, we compare the maximum number of rules installed on a forwarding device between XPath and MINNIE for a larger variety of topologies. Numbers reported in the table for XPath are directly extracted from [Hu+15]. In MINNIE, we consider all the flows between servers even if they act only as end hosts but in XPath, only the path between ToRs are considered for the standard architecture (VL2, Fat-Tree). So for an accurate comparison, we apply the same principle to MINNIE by only considering flows between ToRs. Since in [Hu+15], they also consider a bigger table size of 144 000 entries, the limit is set to 144 000 for MINNIE too. MINNIE requires a lower number of rules to be installed than XPath on every architecture while both dealing with all possible (source,destination) flows. This can be explained by the fact that XPath installs rules for all possible paths for every source/destination pair before compressing while MINNIE only considers one path per flow.

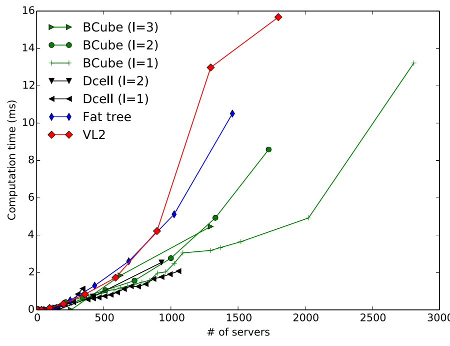
Execution time of Minnie Finally, we study the execution time of MINNIE in order to assess if it is a viable solution in practice. We discuss here the software running time. It represents the time of execution of the algorithm in the controller. In Section 5.3, we then study the additional network delay induced by our method for a flow using our testbed experiment.

Routing time. When a new flow arrives, the controller has to compute its path in the network and the set of rules to be installed in the switches along the path. We plot in Figure 5.6a the average time for this operation. Recall that, to compute the paths we used Dijkstra algorithm with the metrics w_{uv} and residual graph G_{st} described in Section 4.3.1. The longest average time is about 0.42 ms which corresponds to the $k = 18$ Fat-Tree (with 1458 servers and 405 switches), whereas the shortest routing time happens for VL2, DCell(l=1) and BCube(l=1) which have a small number of shortest paths between two routers. On the contrary, the Fat-Tree and BCube(l=3) experience a longer routing time explained by the large number of possible paths between two servers. Note that even if Fat-Tree and VL2 have a similar shape, the latter topology has significantly fewer switches and edges, which explain the smaller number of possible paths and therefore the smaller routing time. Nevertheless, **for all of the studied topologies, the routing time is small and we will see in Section 5.3.2 that the delays of the packets are not significantly impacted.**

Moreover, we observe a surprising behavior for some topologies. In most cases, the computation time is globally increasing with the size of the topologies. However, DCell(l=1), BCube(l=1), and BCube(l=2) experience a drop in computation time: For example, the computation time for BCube(l=1) topologies increases to 0.18 ms for 1024 servers, then drops to 0.10 ms for 1350 servers to increase again to 0.22 ms for 2800 servers. This behavior is caused by the saturation of a large number of switches of the topology when the number of flows becomes high during the simulation. A switch is *saturated* when the compression module can no longer reduce the size of the table below the 1000 limit. However, a saturated switch can still forward a new flow (say between server s and server t) using the first wildcard rule in the routing table of the form $(s, *, p)$, $(*, t, p)$, or $(*, *, p)$. The degree of this switch is one in the residual graph used by MINNIE to compute Dijkstra. This decreases the computation time and the routing becomes very fast when the number of saturated switches is large (as the number of possible paths is then small). This is helpful as it may decrease the routing time of large topologies with high number of flows.



(a) Average routing time over all flows



(b) Average compression time per forwarding device

Figure 5.6: Computation time for the compression and routing phases for different topologies.

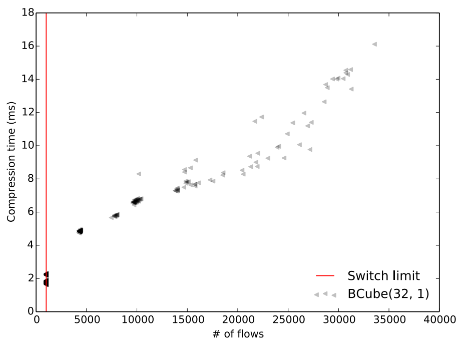
Compression time. After having determined the path of a new flow and installed the rules along the path, we check if the size of one of the corresponding routing tables reaches the limit of 1000 rules. If so, MINNIE carries out a compression of the routing table. We plot in Figure 5.6b the average time to compress a routing table during the simulations for each group of topology. We see that even for the simulations of the largest topologies (pushed to their maximum with an all-to-all traffic of 6 million flow), the average compression time is below 16 ms. This corresponds to large (uncompressed) routing tables dealing with 20 000 flows. A topology with around 1000 servers (1 million of flows in total) experiences an average compression time between 2 and 4 ms. As a typical example, we provide in Figure 5.7 the time needed to compress a switch for a BCube(32,1) and a $k = 12$ Fat-Tree (432 servers) in function of the number of flows passing through it. For the $k = 12$ Fat-Tree, the average compression time is 1.29 ms. For any switch, the first compression is done when reaching 1000 flows corresponding to 1000 forwarding rules (as aggregation rules are only introduced at the first compression). We then see that the second compression for a switch is done for around 2500 flows followed by compression when reaching 3000 to 4000 flows. These compression results show that previous compressions were efficient and that a large number of new flows are routed via aggregated rules. As for the two exceptions observed of tables compressed with around 18 000 flows[1], they correspond to one or two switches on which the paths are concentrated.

These time results allow to assume that the **impact of Minnie on the controller load and on the flow delay will be limited** for these sizes of topologies. Note also that, when a new flow arrives, we choose to apply the compression module when the routing table size reaches the rule limit, but only after the new flow is routed. Thanks to this strategy, the delays experienced by the packets of the flow are not impacted by the compression carried out by the controller. These results are furthermore validated by running Minnie on a data center testbed in the following sections.

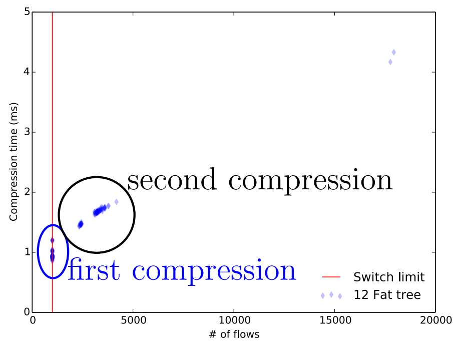
5.3 Experiments on an SDN testbed

In this section, we demonstrate the effectiveness of Minnie using an SDN testbed. The characteristics of our experimental network is described in Subsection 5.3.1. More specifically, we explain how with a single hardware switch and Open vSwitches (OVSes) we deploy a full k=4 Fat-Tree topology with enough clients to exceed the routing table size of the hardware switch,

[1]Beware to distinguish the number of flows in the network from the number of rules. Here the number of rules per router is always below 1000 while the number of flows can be way higher.



(a) BCube(32, 1)



(b) $k = 12$ Fat-Tree

Figure 5.7: Scatter plot of the time to compress a table as a function of the number of flows passing through the forwarding device.

as well as the traffic pattern that fits the needs of our cases of study. A few details about the implementation of MINNIE within a Beacon controller is also provided. The obtained results are shown in Section 5.3.2, where we discuss the impact of MINNIE over the traffic delay, loss rate and the impact of using software rather than hardware rules. A table of content of the results obtained through simulations and experiments is available in Table 5.1.

5.3.1 Experimental settings

Testbed description

Our testbed consists of an HP 5400zl SDN capable switch with four modules of 24 GigaEthernet ports, and 4 DELL servers. Each server has 6 quad-core processors, 32 GB of RAM and 12 GigaEthernet ports. On each server, we deployed four Virtual Machines (VMs) with eight network interfaces each. Each VM is connected to a dedicated OVS. Each OVS is further connected using one physical port (of the server's 12 ports) to the HP switch.

The topology of our data center network is a full k=4 Fat-Tree topology (see Figure 5.8), which consists of 20 SDN hardware switches. Those switches are emulated with 20 Virtual Local Area Networks (VLANs) on the physical switch (referred to as vSwitches). Since each VLAN possesses an independent OpenFlow instance, it behaves as an independent SDN-based switch with its proper isolated set of ports and MAC addresses. The VLAN configuration and ports isolation prevent the physical switch from routing traffic among VLAN through the backplane. The vSwitches are then interconnected on the HP switch using Ethernet cables.

The HP SDN switch can support a total of 65 536 rules, both software and hardware. Software rules are handled in the RAM and processed by the general-purpose CPU (slow path) while hardware rules are stored in the TCAM (fast path) of the switch. The number of hardware rules that can be stored per module in our switch being equal to 750, the total switch capacity is equal to 3000 hardware rules maximum. Those 65 536 entries are shared among all 20 switches and are distributed to the vSwitches in the order of arrival of the flows.

In one of the physical servers, we also deployed an additional VM hosting a Beacon [Eri13] controller to manage all the switches (HP vSwitches or OVSes) in the data center. According to [Sha+13], Beacon features high performance regarding throughput and ensures a high level of reliability and security. To prevent the controller from becoming the bottleneck during our experiments, we configured it with 15 CPU cores and 16 GB of RAM.

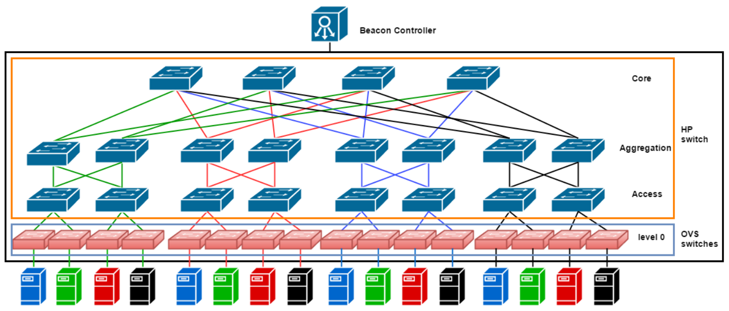


Figure 5.8: Our k=4 Fat-Tree architecture with 16 OVS, 8 access, 8 aggregate, and 4 core switches.

Detecting new flows

When a switch does not find a match for a packet to a flow entry in its forwarding table, its default action is to contact and ask the controller the action to perform. However, by introducing wildcard and default rules in the tables, we override this default action, which can introduce routing errors. Indeed, let us consider a brand new flow between s_1 and d_1 on the network. If the access switch of s_1 contains a matching wildcard rule, e.g., $(s_1, *, p_2)$, due to a previous compression of its table, the switch will forward it through the port p_2 without contacting the controller. In the best case scenario, the switch connected via p_2 does not have a corresponding wildcard rule and will contact the controller. In the worst case, the packet is forwarded to its destination without ever being detected by the controller. In both cases, it can cause inconsistent and inefficient routing in the network and can impact the compression of the table since the controller is missing information.

In [Rif+15], we circumvented the problem by not compressing the table of the access switches, at the cost of lower compression ratios and overloading of these switches. The problem can also be solved by deploying OVSes between the servers and the access switches. The forwarding tables of the virtual switches are not compressed to allow compression on the access switches without missing detection of new flows.

This added layer could be seen as a mere shift of the problem from the edge devices to the physical servers. We believe however that this architecture represents a major step towards the solution of limited TCAM space because of the following reasons:

1. While the TCAM size is a real problem for physical SDN-capable devices, placing one OVS per server, even without compressing the flow table, should not introduce major performance problems. Indeed the number of rules to be processed by each OVS should remain modest[2] while an OVS can handle 1000 rules at the kernel space, and up to a maximum of 200 000 rules [Pfa+15].

2. Virtualization is a common service in modern data centers. Hence, virtual switches are routinely used to provide network access to the virtual machines. OVS is natively supported by Xen 4.3 and newer releases. VMware offers support to OVS through the NSX service for Multi-Hypervisor, which is the natural choice for large data centers. KVM, due to its native integration in Linux environments, can easily be deployed using OVSes.

[2] As reported in [Cur+11], a typical rack of 40 servers generate around 1300 flows per second. Therefore, each server is generating on average around 32.5 flows per second. Assuming a worst case where every rule per flow is unique and that the expiration interval for unused rules is the default value of 10 seconds of inactivity, then, a OVS in a single server will need to store 325 forwarding rules roughly (plus the default route to reach the local VM). This value is pretty small compared to the 1000 rules in the fast path of an OVS.

Number of clients

In our Fat-Tree architecture, we can easily deduce the number of rules corresponding to a valid routing assuming that each VM talks to all other VMs not in its IP subnet, i.e., VMs not connected to the same ToR. Without compressing the rules, a flow requires one rule on each switch along the path from its source to its destination. The set of flows that a switch forwards depends on its level in the Fat-Tree. Note that here, a flow is identified by the couple IP source and IP destination addresses. Hence, for every pair of nodes A and B, there are two unidirectional flows: $A \rightarrow B$ and $B \rightarrow A$, i.e., two rules per switch on the path from A to B.

For any flow between two servers, the path first goes through the access switches to which the servers are connected. Assuming n servers per access switch ($n = 2$ in Figure 5.8), then each of the n servers connected to an access switch communicates with the other $7 \times n$ servers in other subnets via outgoing and incoming flows. Overall, this represents $14n^2$ flows going through any access switch.

Using the same argument to find the number of flows for switches at the higher levels, we have a total of $13n^2$ flows at each aggregation switch and $12n^2$ flows for a core switch. In total, $264n^2$ rules are needed for the entire network.

In Figure 5.9, we compare the total number of rules with no compression at all, and with compression (obtained via simulation) on all switches. Without compression, we can only deploy 15 clients per subnet without running out of space in the forwarding table of our entire data center (65 536 entries), while up to 36 clients can be deployed when compressing at the end. Therefore, Figure 5.9 explains our choice of installing 16 clients per subnet. Indeed, it is the first value for which the number of rules exceeds our total number of rules' limit (67584 rules) when without using compression.

Experimental scenarios

We aim at assessing the performance of Minnie with a high number of rules and a high load. Unfortunately, those two objectives are contradictory in our testbed. Indeed, stressing the SDN switch in terms of rules, i.e., getting close to the limit of 65536 entries, imposes to have software rules. As software rules are handled, by definition, by the general purpose CPU of the switch (the so-called slow path), a safety mechanism has been implemented by HP to

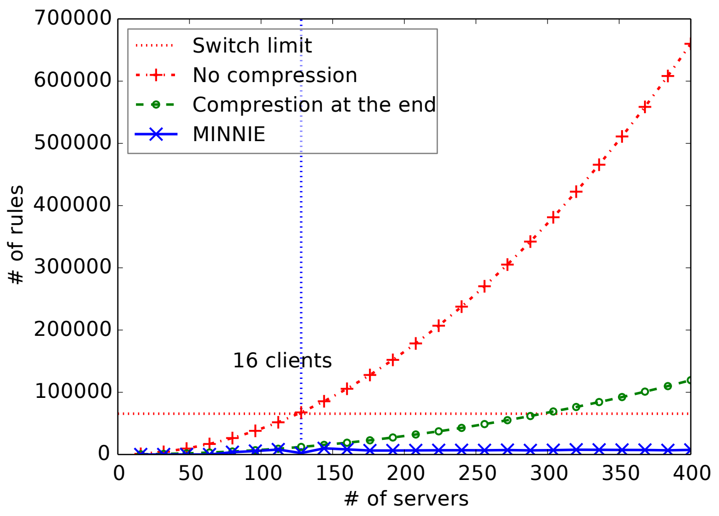


Figure 5.9: Total number of rules installed as a function of the number of servers, in a $k = 4$ Fat-Tree configuration.

limit the processing speed to only 10 000 packets/s per VLAN. Assuming an Maximum Transmission Unit (MTU) of 1500 bytes, we could not go beyond 120 MB/s, shared between all ports in a VLAN. This is why we designed a second scenario where only hardware rules are used. In this scenario, we can fully use the 1 GB/s link but we are limited to the 3000 hardware rules that have to be shared among the 20 switches. We thus consider two scenarios to assess the performance and the feasibility of deploying Minnie in real networks:

- **Scenario 1: Low load with (large number of) software rules** *(LLS)*. This scenario enables to test the behavior of the switch when the flow table is full.
- **Scenario 2: High load with (small number of) hardware rules** *(HLS)*. This scenario enables us to demonstrate that the impact of Minnie remains negligible even when the switch transfers a load close to the line rate.

For each scenario, we consider three compression cases, which are similar to the ones presented in Section 5.2.1:

- **Case 1: No compression**. Tables are never compressed. This case provides the baseline against which we compare results obtained with MINNIE.
- **Case 2: Compression at the end** In this case, we compress the table once the physical forwarding table is full or when all forwarding rules have been installed. This scenario illustrates the worst case and provides insights about the maximum stress introduced by MINNIE in the network. Indeed, by delaying the compression, we end up with the largest amount of rules possible to remove and the biggest compressed table to install.
- **Case 3: Minnie** Individual tables are compressed once they reach a set threshold. Three different thresholds values are considered in both scenarios: 500, 1000 and 2000 rules for *LLS*, and 15, 20 and 30 rules for *HLS*.

While *LLS* allows to test the scalability of MINNIE in terms of number of rules in real SDN equipments, this scenario might introduce, by default, an important jitter in the network because of the usage of the general-purpose CPU to process the traffic.

HLS helps to better understand the impact of the compression and forwarding table replacement over the traffic. Since the traffic rate fills up to 75% of the access links, which is not enough to introduce congestion, and packets are processed by the Application-Specific Integrated Circuit (ASIC), we expect to have a low jitter. Hence, any sudden increase of this last will immediately suggest an important impact of the compression mechanisms over the network stability.

Traffic pattern

We further detail the actual implementation of the two scenarios in our testbed.

Low Load with software rules Scenario - *LLS*. In this scenario, the traffic is generated as follows: each client pings all other clients in every other subnet. This means that for each access switch, each of the 16 clients pings 112 other clients. As we explained in Section 5.3.1, pings between hosts in the same subnet are not routed using IP and thus are not using any TCAM space.

We start with an initial client transmitting five ping packets to one other client. This train of five Internet Control Message Protocol (ICMP) requests forms a single flow from the SDN viewpoint. We wait for this ping to terminate before sending five other different ping packets to another client, and so on until all the 112 clients are pinged. When the first client finishes its pings series, a second client (hosted in the same VM) starts the same operation. Hence, the traffic is generated during the whole experiment in a round-robin manner, among the eight clients of each VM. Moreover, VMs do not start injecting traffic at the same time. We impose an inter-arrival period of 10 minutes between them. Hence, VM 1 starts sending traffic at time zero, while VM 2 starts at minute 10, VM 3 at minute 20, and so on. This smooth arrival of traffic in the testbed is motivated by the fact that we do not wish to overload the physical switch with OpenFlow events. Indeed, as stated in [Kre+15], commercial OpenFlow switches can handle up to 200 events/s. Since our testbed has 20 switches that handle its *flow_mod* (message for sending rules), *packet_out* (message with the packet to send) and other events, the critical number of events can be easily reached.

An experiment for this scenario lasts up to three and a half hour, and all rules are installed after the first two hours and forty-five minutes.

High Load with hardware rules Scenario - *HLS* In this scenario, we used one client per VM so that the total number of rules installed (1056 rules in total) is less than the hardware limit (3000 rules). Each VM starts a 50 MB/s ICMP traffic with the other clients in a round robin manner. We wait a period of 75 s before starting the outgoing traffic of a new VM. In total, we have a load of 800 MB/s on each 1 GB/s link.

This scenario lasts up to 1 hour, and all the rules are installed, as hardware rules, after the first 20 min.

Deploying Minnie in the SDN controller

When the controller is asked about a new packet in the network, the Minnie SDN application will first execute the routing module and then, if necessary, the compression module. The compression module is called on a table when a new rule is installed, and the table reaches its maximum threshold. The rule is first installed in the switch table to allow the packets to be forwarded and the compression module is executed. Once the compression is launched at the controller, a single OpenFlow command is used to remove the entire routing table from the switch. Then the compressed table is sent immediately to limit the *downtime* period, defined as the period between the old rules removal and the installation of the compressed table. When two or more switches need to be compressed at the same time, the compression is executed sequentially.

As stated in Chapter 3, OpenFlow entries can specify a 16-bit priority field in the case where a flow matches multiple entries in the table. The switch executes the action of the rules with the highest priority.

The compressed table created using MINNIE only contains three types of rules: (*i*) *Normal forwarding rules* which match on source and destination (*ii*) *Aggregated forwarding rules* that match on either source or destination (*iii*) and the *default rule*. Note that a compressed table, given by MINNIE, can not contain both types of aggregation rules. Hence, three priorities are sufficient, with the normal rules having the highest and the default rule having the lowest.

To minimize the downtime when compressing and pushing its compressed table to an SDN device, we decided to delete all the rules and install the new rules instead of updating existing rules. This decision was motivated by the fact that updating the SDN rules in TCAM is time-consuming and an update operation is considered as two operations (delete + insert) [KPK14]. Our methodology leads to a single delete action for the whole table and then a batch of rule insertions. These rules are going to be inserted without waiting for the barrier reply message in order not to provoke large delay (see [KPK14] for details). In case one rule was not installed in the SDN switch, the controller will be notified of this problem, and it will then reinstall the required rule. As we will see in Section 5.3.2, this strategy did not have any negative impact on the network traffic delay or packet loss.

5.3.2 Experimental results

Scenario 1: compression with *LLS*

Number of rules As explained in Section 5.3.1, in this scenario and without compression, the limit of 65 536 entries in our HP switch is reached. On the other hand, compressing the table with MINNIE allows installing all the required rules without reaching the limit, when compressing at a given threshold (500, 1000 or 2000 entries) or when the flow table is full. Indeed,

Level	No Comp	Comp 500	Comp 1000	Comp 2000	Comp full
access	3452	752	761	790	802
aggregation	3233	618	649	672	717
core	3014	97	97	97	97
total	65 535	11 346	11 667	12 087	12 542

Table 5.4: Average number of SDN rules installed in a virtual switch at each level

Level	Comp 500	Comp 1000	Comp 2000	Comp full
access (8 switches)	79%	78.75%	77.95%	77.61%
aggregation (8 switches)	81.43%	80.51%	82.14%	78.45%
core (4 switches)	96.84%	96.84%	96.84%	96.83%
total (20 switches)	83.21%	82.19%	81.55%	81.44%

Table 5.5: Average compression ratios for all compression threshold.

as shown in Table 5.4, the total number of installed rules does not exceed 13 000 in all compression cases. This represents a total saving higher than 80% of the total forwarding table capacity (Table 5.5) with a saving larger than 96% at the third level and a minimal saving over 76%.

Figure 5.10 depicts how the number of rules evolves with and without compression. Please, note that this figure takes into account the total number of forwarding rules in the network, including both OVSes and the HP switch. The number of rules increases at the same pace in all three scenarios during the first 30 minutes. When the compression is triggered, the number of rules decreases. Later, for compression at 500 and 1000 entries, the number of rules increases at a lower pace than in the non-compression case. This is because (*i*) the controller has installed some wildcard rules and so fewer rules at level 1, 2 or 3 need to be installed for new flows, and (*ii*) other compression events are triggered. We further notice here that the presence of wildcard rules also explains the difference between the compression when the forwarding table is full and the compression with fixed thresholds. This is in line with the results of Section 5.2.1 where we observed that the presence of wildcard rules in the routing tables influences the routing as the new incoming flows will follow these paths in priority. Even though the difference between dynamic compression and compression at the end is more pronounced for networks with a larger number of servers (see Figure 5.5), the phenomenon can already

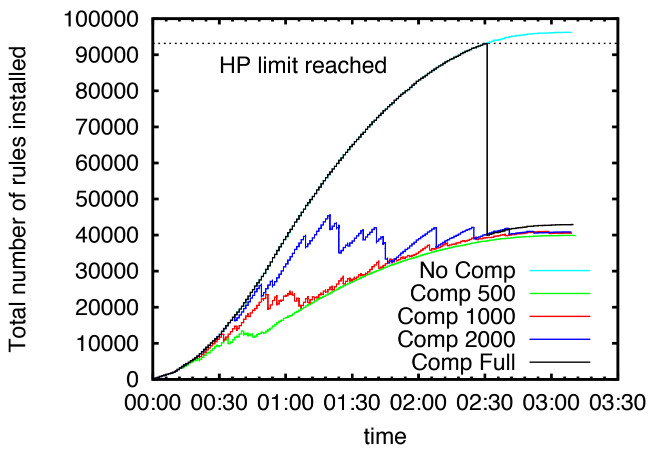


Figure 5.10: Total number of rules installed in the network for all compression threshold

be observed in the testbed.

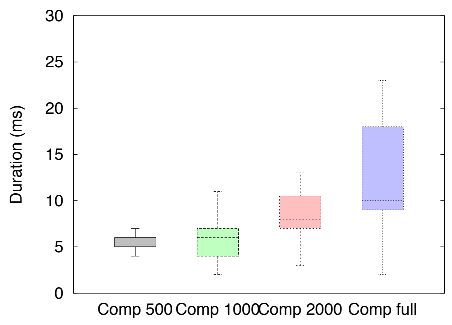


Figure 5.11: Duration of a compression event for all compression threshold

Compression time Figure 5.11 shows the compression time seen by the controller. This time comprises the computation time of compressed rules (already analyzed in Section 5.2.2), the removal of the current forwarding

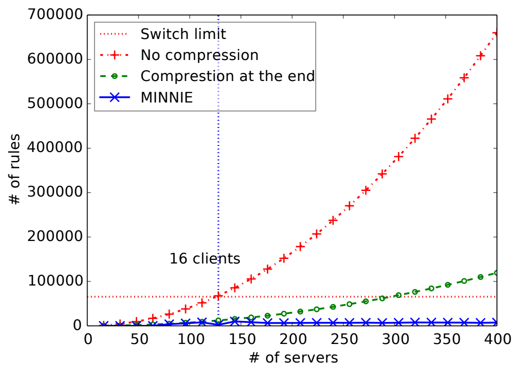


Figure 5.12: Total number of rules installed as a function of the number of servers, in a $k = 4$ Fat-Tree configuration.

table, the formatting of the compressed rules to the OpenFlow standards, and the injection of the new rules to the switch.

We notice that the compression time per switch remains in the order of a few milliseconds. Indeed, compression takes about 5 ms (resp. 7 ms) for compression at 500 and 1000 entries (resp. 2000 entries). Even the worst case – compressing when the table is full – represents less than 18 ms for most of the switches with a median at 9 ms. Moreover, in this latter case, sequentially compressing all switches requires no more than 152 ms. This compression period is mainly due to the time needed to delete all the routing table using one delete request and install all the new rules in the switch. Indeed, the time needed to compute the compressed routing table is negligible as noted in Section 5.2.2 (Figure 5.6). It is important to note here that the code used to compute the compressed tables is the same in our simulations and experiments. These results are in line with the results shown in [KPK14] (Figure 3). We have smaller delays because, as stated before, we do not wait for the barrier reply before sending the next flow insertion rule (hence we ignore barrier reply message time). Moreover, we delete all the rules using a single action instead of deleting each rule one by one.

SDN control path In the SDN paradigm, the controller-to-switch link is a sensitive component as the switch is CPU bounded and cannot handle events at a too high rate. Figure 5.13 represents the network traffic between the switch and the controller in the different scenarios. We can observe

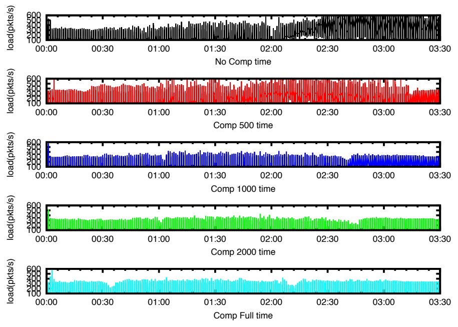


Figure 5.13: Network traffic between the switches and the controller for all compression threshold.

that the load increases highly when the switch reaches its limit in terms of the number of software+hardware rules and we do not compress the routing tables. After time t=2:30, the limit is reached, and for every packet of every new flow, each switch along the path has to ask the controller for the output port. These traffic peaks vanish when we compress the routing tables for the 1000, 2000 limits, and for the case of compression when full. As for the *compression at 500* scenario, we notice the occurrence of high peaks after the first hour. They result from successive compression events (over 16 000 in our experiments as can be seen in Table 5.6) that are triggered by any new packet arrival. Indeed, in this scenario, most of the switches will perform a compression for every new flow, since the total number of rules after compression remains higher than the threshold.

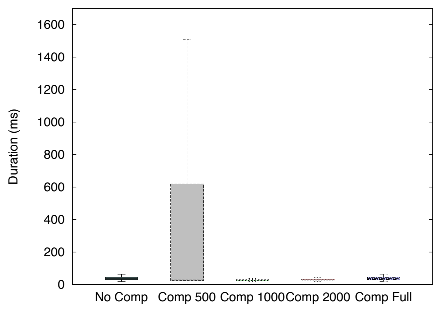
To understand the impact of the control plane on the data plane, we look at three key metrics that we detail in the following sections. We first consider (i) the loss rate for all scenarios, followed by (ii) the delay of the first packet of new flows and (iii) the delay of subsequent packets (packets 2 to 5). Note that the first packet's delay should be higher when there is no compression (at least after t=2:30) or for the *compression at 500* scenario. We ruled out a precise study of the loss rate as the load in this section is low. We report in Table 5.6 the loss rates observed for all scenarios. Though there exist some significant differences between the different scenarios, the absolute values are fairly small. We, therefore, focus on delays hereafter.

	Compression threshold				
	None	500	1000	2000	When full
# of compressions	0	16 594	95	28	20
% packet loss	6.25×10^{-6}	0.003	5.65×10^{-4}	2.83×10^{-5}	3.7×10^{-4}

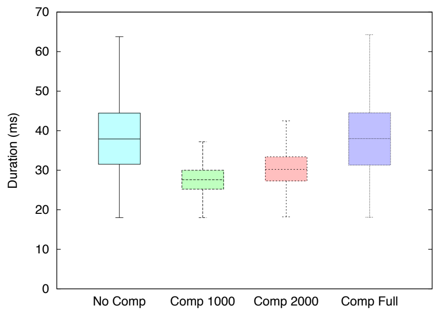
Table 5.6: Total number of compressions and packet loss rate for all compression threshold.

New rules installations: Impact on first packet delay The first packet delay provides insights on the time needed to contact the controller and install the rules when a new flow arrives. Indeed, the round trip delay seen by the first packet of a new flow includes the network propagation delay, the queuing delay, and the time needed for a switch to obtain a new rule.

We observe in Figure 5.14 that for the scenarios with compression at 1000 rules and compression at 2000 rules, the first packet delay ranges from 25 ms to 35 ms. This increase as compared to subsequent packets of the same flow - which can reach a factor of 10 as we will see in the next section - highlights the price to pay to obtain and install a forwarding rule in software. The results can significantly worsen if the controller is frequently modifying the forwarding rule, like in the compression at 500 rules case. Indeed, for that special case, the third quantile reaches up to 600 ms for the first packet delay.



(a) All threshold



(b) Without *compression at 500 rules*

Figure 5.14: First packet delay for different compression threshold (boxplot)

Surprisingly, the cases without compression and compression at the table limit lead to similar results. Compressing when the table is full should intuitively result in better performance as in many cases, a limited number of new rules are needed and can be installed as compared to the no compression case. However, in our tests, the table becomes full after 2 hours and 30 min of the experiment (out of 3 hours). Hence the similarity of results in Figure 5.14. In fact, when the table is full, the impact is striking, as can be seen in the time series of Figure 5.15a, which shows the evolution of the first packet delay per new flow when no compression is executed. Indeed, after 2:30 hours - when the table is full- we can observe a jump in the delay for no compression while when compressing at the table limit the trend is the opposite and the delay decreases (Figure 5.15e) after compression. As for the case of compression at 500, the first packet delay features a chaotic behavior (Figure 5.15b) due to its high compression frequency as expected. Regarding the scenarios of compression at 1000 (Figure 5.15c) and compression at 2000 (Figure 5.15d), the benefits of compressing periodically are striking: the first packet delay shows a constant trend during the whole experiment.

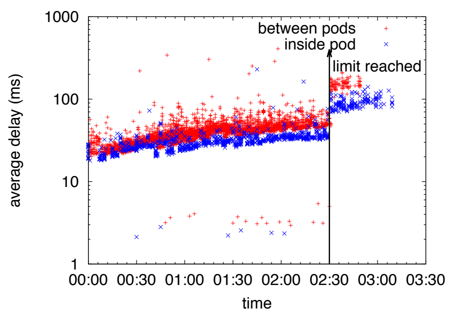
Eventually, note that the results obtained here are impacted by the fact that we use software rules, which increases the delay to install rules. Results of the experiments using TCAM exclusively are provided in Section 5.3.2.

Subsequent packets' delay As explained previously, we expect to observe higher delays for subsequent packets for the case of no compression (when the table is full) and also possibly for the case of compression at 500 as the switches have to reinstall new rules at a high frequency.

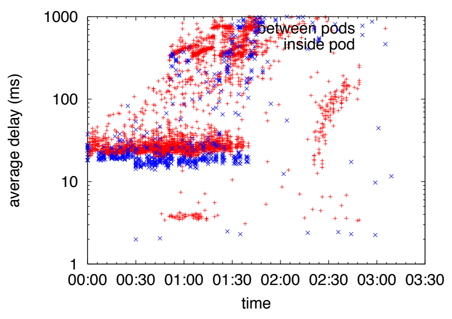
In our experiments, the delay seen by packets 2 to 5 of each flow is shorter than 4 ms most of the time for scenarios without compression, compression at 1000, compression at 2000 and compression at the forwarding table size limit, as we can see in Figure 5.16. Compression at 500 is slightly different (the third quartile reaches up to 5 ms), highlighting the negative impact of the high frequency of compression events on the data path of the switches.

Figure 5.16 aggregates all the results together, and we have again to resort on the time series to observe specific effects. When all needed forwarding rules are successfully installed and the compression frequency is low (which is the case for compression at the limit, compression at 1000 and compression at 2000), the delay of packets 2 to 5 is consistently comprised between 2 ms and 6 ms (Figures 5.17d to 5.17f).

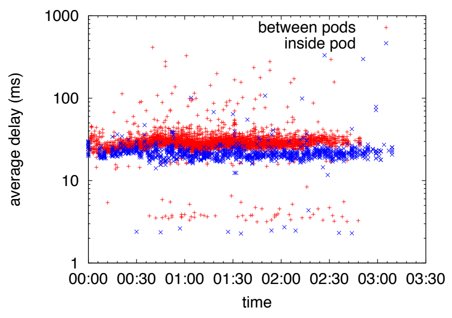
Without compression, while most of the packets experience a delay be-



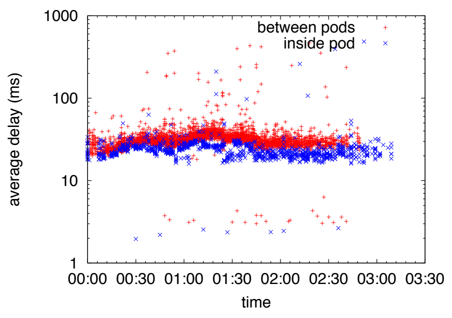
(a) Without compression



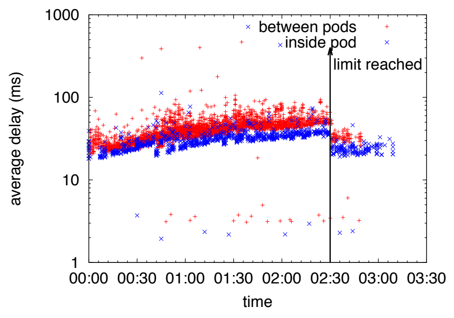
(b) Compression at 500



(c) Compression at 1000



(d) Compression at 2000



(e) Compression when full

Figure 5.15: First packet delay for different compression threshold (over time)

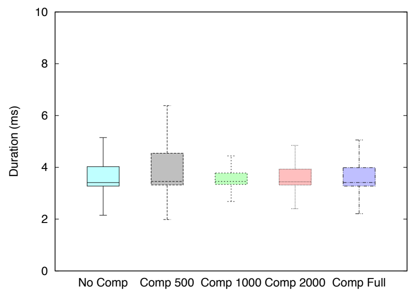


Figure 5.16: Average packet delay for subsequent packets (boxplot)

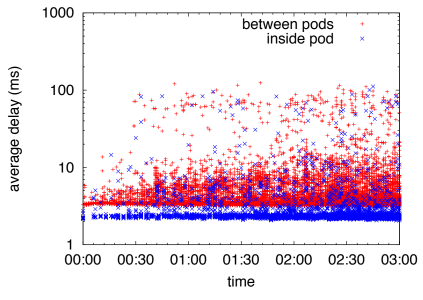
tween 2 ms and 6 ms before the table limit is reached, all new incoming packets will see a delay equal or higher than 40 ms afterward (Figure 5.17b). As for the case of compression with small table limit (500 rules), we remark in Figure 5.17c a time interval between 1:45 hour and 2:15 hour, where the delay increases suddenly from 2 ms to 100 ms. This is because some switches are unable to reach a forwarding table smaller than 500 rules even after compression, and hence, the controller executes a compression after every new flow arrival. After time 2:15, the frequency of new incoming flows that need to be installed decreases (Figure 5.15b), leading to a stabilization of the delay.

From all the results shown above, we notice that putting a low table limit (e.g., 500) has a negative impact on the traffic passing through the network, whereas setting it to 1000 and 2000 provided enhances performance for network traffic. This is due to the fact that in our scenarios, the compressed table had a size larger than 500 while it was always less than 1000. Hence, to leverage always the benefit of Minnie we advise to set a dynamic threshold that will change based on the compressed table size - see Section 5.3.3.

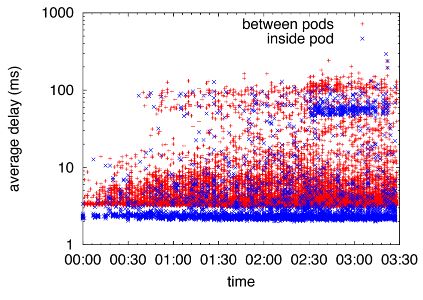
Scenario 2: compression with *HLS*

We have so far investigated the behavior of Minnie in an environment where the flow table can be full. The latter scenario involves the use of software rules and thus the slow path of our HP switch.

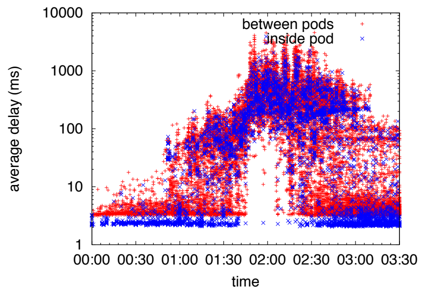
We now turn our attention to the case where the load on the data plane is as high as 80%. This entails using hardware rules only, and we are limited



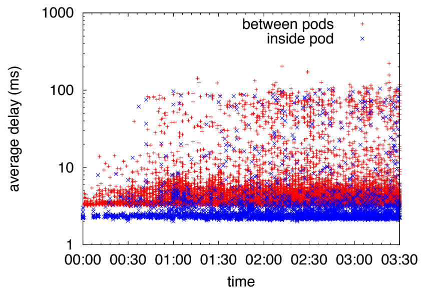
(a) Without compression 7 IPs



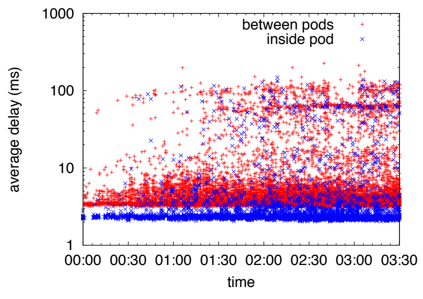
(b) Without compression 8 IPs



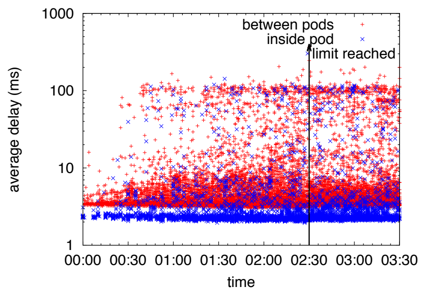
(c) Compression 500



(d) Compression 1000



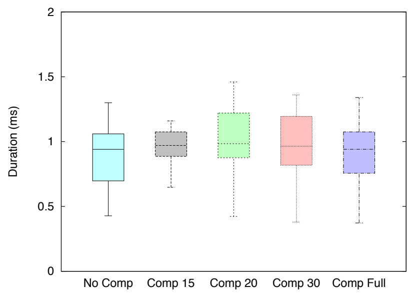
(e) Compression 2000



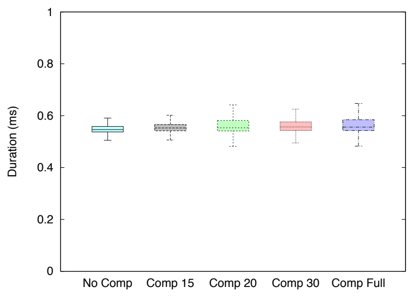
(f) Compression when full

Figure 5.17: Average packet delay for subsequent packets (over time)

to 3000 such rules with our HP switch, shared among the 20 switches of our k=4 Fat-Tree topology. The experiments in this section are consequently performed with one client per access switch (16 clients in total) and an all-to-all traffic pattern with 50Mb/s per flow.



(a) First packet delay



(b) Subsequent packets' delay

Figure 5.18: Packet delays in the *HLS* (boxplot)

As expected, the first packet round trip delay decreases to around 1 ms, while packets 2 to 5 experience a round trip delay of around 0.55 ms[3]. The compression duration, in all scenarios, is equal to 1 ms only, which is understandable given the small total number of flows. **More importantly, we noticed no packet losses and no drastic effects on delay even during compression events, which proves that Minnie is a viable and realistic solution**. Indeed, the maximum variation of delay between the delays of no compression and all compression scenarios is less than 0.1 ms, a value which might be observed even in non-SDN networks (see Figure 5.18).

The compression ratio in Table 5.7 demonstrates that even with a low number of rules, MINNIE can achieve a high compression ratio, over 70%. Figure 5.19 which represents the evolution of the forwarding table size for all cases – no compression, compression at 15, 20, 30 and when full (after installing all the needed rules)– highlights that MINNIE maintains a similar low number of rules in all compression scenarios.

The last question that we aim at investigating is the impact of compression on TCP connections. The high load scenario is especially relevant as

[3]A direct comparison between these delays and the one for the low load and software rules scenario is not straightforward. Section 5.3.2 will present a fair comparison of these two modes.

	Compression threshold			
	15 rules	20 rules	30 rules	when full
level 1 (8 switches)	76.56%	75.66%	75.00%	72.76 %
level 2 (8 switches)	75.48%	73.31%	71.87%	69.71 %
level 3 (4 switches)	76.04%	76.56%	74.47%	73.95 %
total (20 switches)	76.04%	74.9%	73.67%	71.78%

Table 5.7: Average percentage of SDN rules savings at each level

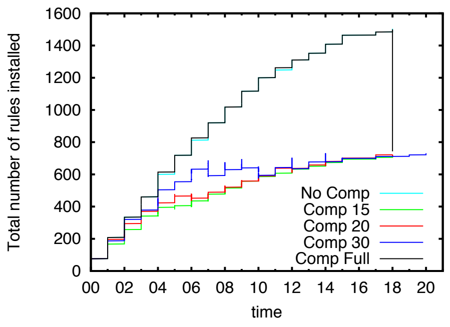


Figure 5.19: Total number of rules installed in the network

data centers are in general operated at high loads. The variation of the round trip delay of most of the packets is less than 0.1 ms (Figure 5.18) for compression at 20 entries with the highest variability. For compression at 20 entries and during the first 20 minutes of the experiment (compression events occur during that period), the minimum and maximum round trip delays between servers in the same pod is around 0.4 ms and 0.6 ms respectively. The minimum and maximum round trip delays between servers in different pods is around 0.55 ms and 0.8 ms respectively (see Figure 5.20). Those observed delays will not produce any problem to TCP connections. Indeed, the minimum Retransmission Timeout (RTO) value (the time needed to trigger a TCP timeout and retransmit a non-acknowledged packet), is equal to 200 ms on Linux systems (and defined to be 1 second in the Request for Comments (RFC) 2988 [PA00]), which is far from our observed delays (lower than a millisecond). A recent draft submitted to the TCPM Working Group [Ben+15] appeals for a decrease of the minimum RTO value to 10 ms. Once again, the maximum delay observed during the compression events is still far from that proposed minimum RTO. Hence compression operations should not lead to any spurious TCP time out. Note that results obtained in the simulations on the computational time (Figure 5.6 of Section 5.2.2) confirm that the impact of Minnie on the delay experienced by the packets of the flow should be limited in general.

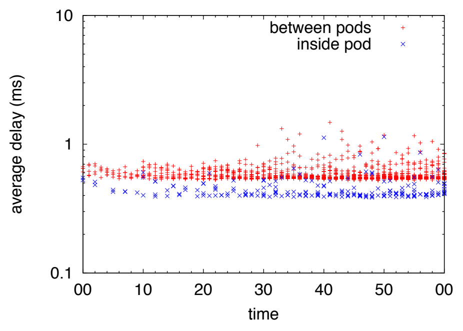


Figure 5.20: Subsequent packets' delay with *Compression at 20 entries*

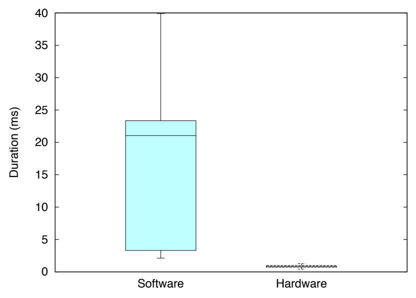
Comparison between software and hardware rules

So far, we have seen that relying only on the ASIC of the switch to forward the traffic provides better results, regarding delay and jitter, than using the general purpose CPU for such a task. Hence, the question of the impact of the *slow path* on the switch performance arises.

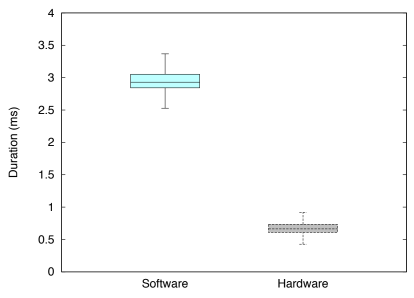
Assessing the difference between using hardware and software rules by comparing the results of the previous sections is difficult as the number of rules is different from one scenario to the other. For this reason, we devised a third scenario where we compare the performance of software and hardware rules using both, the same number rules and the same traffic load, in all cases.

In this experiment, we have one client per access switch, and each flow is composed by a train of 5 ICMP request/reply packets, which is the default behavior of the `ping` command. With this configuration, we can observe in Figure 5.21a that installing rules in software increases the first packet delay by a factor of 20 from a median of 1 ms to 20 ms as compared to hardware rules. The average matching delay of the remaining packets (Figure 5.21b) features a 6-fold increase in software as compared to hardware (3 ms compared to 0.5 ms).

The results obtained with these experiments thus confirm the large discrepancy in terms of average delay results between the two scenarios. **They further justify the necessity of using only TCAM, which can better be exploited thanks to the compression executed by Minnie.**



(a) First packet



(b) Subsequent packets

Figure 5.21: Comparison of the delay between hardware and software rules

5.3.3 Discussion

The results obtained via simulation, in Section 5.2.2, and experimentation, in Section 5.3.2, demonstrate the feasibility and efficiency of Minnie. We discuss here several practical points and possible extensions of our algorithm.

Dealing with different workloads.

We have used an all-to-all traffic pattern, which constitutes a worst case in terms of traffic workload that an application could generate in the network. This was however achieved with 16 IPs per server in the experimentation part and 1 IP per server in the simulation part, which might seem relatively limited. However, in an operational network deployment, it is reasonable to admit that SDN rules are mainly installed on an IP subnet basis, while flow-based rules (created with the matching of all or several fields of the OpenFlow standard) might rarely be employed. Our results can thus be interpreted as routing all-to-all traffic between several IP subnets per server, as one expects to observe in a typical data center where virtualization is used. This means that Minnie can deal with a worst case traffic scenario involving a large number of end hosts.

Rule deletion.

All scenarios studied in this work considered flows with an unlimited lifetime in order to obtain a worst case scenario regarding the total number of rules involved. However, in practice, flows are active for a limited amount of time as they come and go. We discuss here a possible extension of Minnie that would handle the departure of flows.

OpenFlow enables the use of idle or hard timeouts to remove rules if no more packets are seen (idle) or after a fixed time interval (hard). Timeouts could be set on the level-0 switches, allowing the detection of inactive flows by Minnie. Hard timeouts enable the controller to know the exact state of each level-0 switch without any feedback from the switch. With idle timeouts, the controller can specify (in OpenFlow) when a rule is inserted, that the switch must notify the controller when the rule expires. With the exact information of the currently active rules, Minnie, which keeps an uncompressed version of all the rules in all switches, can delete any unused aggregated rules. As more and more rules are removed, the compression module could also be called to produce a smaller table to insert in place of the current one.

Impact of compression on rule update.

We discussed the impact of rule compression on the performance of rule update in several parts of the paper. We summarize here the findings. We have 3 cases of rule update in the compressed tables:

- Addition of a new simple rule (assuming the table sizes are below the compression threshold). This event is due to the arrival of a new flow. In this case, there is no impact of compression on rule update. Note that, thanks to aggregated rules, a new flow arrival will require a new entry at the level-0 OVSes, but might require no new entries at the access switches or higher switch levels, if the new flow is routed by already existing aggregated rules. In this case, we do not have to update the routing table.

- Deletion of a rule. This is done in particular when a flow finishes. This operation is discussed above and was not tested yet. However, the controller knows which flow uses which rule (simple or aggregated), and thus may easily know which rule to delete (or not) when an entry expires at the level-0 OVS switches, which is a fast operation.

- Compression event. If a table is full, we compress the table in its totality, and we send the new compressed table to the corresponding switch. We then update the switch table by doing, first, a delete operation to remove the old table, and then, we send the new rules to be inserted in the fewest number of packets[4]. We measured the duration of these operations experimentally and tested its impact on delay and packets losses. We first evaluated the time needed to carry out a compression event (compression time, time to send a new table to the switch, and time of updates). We show that this time is in the order of a few milliseconds, as presented in Figure 5.11. Recall that, if a compression event is needed when a new flow arrives, we first send the forwarding rules for the new flow, and we compress only afterward, thus avoiding additional delay for a new flow due to a compression event. We also

[4]We have observed that several flow_mod operations are encapsulated in only a few TCP packets

evaluated the impact of rule compression on the network thanks to our experiments. We report packet delay and loss rates in our experiments and compare scenarios with compression and without compression. We show that even for the High Load Scenario (HLS), the loss rate and the delay are not impacted, see, e.g., Table 5.7 and Figure 5.19.

Dynamic compression limit.

Early compression helps to maintain the routing tables small. However, the threshold should not be set smaller than the actual compressed table size, as exemplified by the case of compressing at 500 entries in the experimentation part. To work around this potential issue and reap the full benefit of compression we advise to set a dynamic compression limit. We can start for example with a low limit, e.g., 100 rules, and once a certain percentage of our limit is reached, e.g., at 80%, to trigger Minnie to compress the routing table. We then increase this compression limit whenever the resulting compressed table is higher than the actual limit, e.g., to 150% of the current compressed table size.

Dealing with burstiness of traffic.

A dimension that we have not explored during our tests is the burstiness of the arrival of flows that could lead to stress the switch-controller communication and hit the limit of a few hundreds events/s that the switch can sustain. This could be the case of an application that generates a lot of requests towards a large set of servers at a high rate. In this situation, Minnie could help alleviate the load on the controller. Indeed, the sooner one compresses the flow table, the more likely we are to install rules that will prevent the switch from querying the controller for a rule for every new connection. One could argue that compressing entails complete modification of the flow table at the switch, i.e., a large number of events (deletion, insertion) related to the management of the table. However, in OpenFlow, those events can be grouped together: all insertions can be sent at once to the switch. **In summary, Minnie should also help to alleviate the stress of the switch-controller channel in case of flash-crowds of new connections.**

Security.

Eventually, note that Minnie does not alter the security level of the SDN network. Indeed, rules are not compressed in the level-0 switches that connect the VMs to the network. This means that there is no possibility for a packet that belongs to one tenant to be seen or to be inserted in the network of another tenant, provided that the SDN rules at the edge are correctly written. Compressing at the edge could indeed give the opportunity to the traffic of one tenant to enter another tenant's network thanks to some wildcard effect. Note however that we do not compress at the edge not because of any security concern, but to prevent any misbehavior in the routing process.

5.4 Conclusion

In this chapter, we introduced Minnie, which aims at routing flows while respecting link and SDN routing table capacity constraints, using table compression. We have investigated through numerical experiment the versatility of Minnie on a variety of data center topologies and demonstrate that it can handle close to a million flows with no more that 1000 rules per switch.

We also complemented our results with experiments on a testbed emulating a $k = 4$ Fat-Tree. They have confirmed the ability of Minnie to drastically reduce the number of rules to manage with no noticeable adverse effect on delay or packets losses.

Bibliography

[Moy98] J. Moy. *OSPF Version 2*. RFC 2328. Ascend Communications, Inc., Apr. 1998. URL: https://www.rfc-editor.org/rfc/rfc2328.txt (cit. on p. 42).

[Ste+12] Brent Stephens, Alan Cox, Wes Felter, Colin Dixon, and John Carter. “PAST: Scalable Ethernet for Data Centers”. In: *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*. CoNEXT 1́2. New York, NY, USA: ACM, 2012, pp. 49–60 (cit. on pp. 42, 90).

[Bos+13] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. “Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN”. In: *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM - SIGCOMM ’13* (2013) (cit. on p. 42).

[GMP14] Frédéric Giroire, Joanna Moulierac, and T Khoa Phan. “Optimizing rule placement in software-defined networks for energy-aware routing”. In: *GLOBECOM*. IEEE, 2014, pp. 1–6 (cit. on pp. 43, 45, 46).

[Hu+15] Shuihai Hu, Kai Chen, Haitao Wu, Wei Bai, Chang Lan, Hao Wang, Hongze Zhao, and Chuanxiong Guo. “Explicit Path Control in Commodity Data Centers: Design and Applications”. In: *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*. NSDI1́5. Berkeley, CA, USA: USENIX Association, 2015, pp. 15–28 (cit. on pp. 43, 45, 94, 100, 102).

[Coh+14] R. Cohen, L. Lewin-Eytan, J.S. Naor, and D. Raz. “On the effect of forwarding table size on SDN network utilization”. In: *INFOCOM*. IEEE, 2014, pp. 1734–1742 (cit. on pp. 43, 44).

[Gir+10] Frederic Giroire, Dorian Mazauric, Joanna Moulierac, and Brice Onfroy. “Minimizing Routing Energy Consumption: From Theoretical to Practical Results”. In: *2010 IEEE/ACM International Conference on Green Computing and Communications and International Conference on Cyber, Physical and Social Computing* (Dec. 2010) (cit. on pp. 43, 65).

[Cas+09] M. Casado, M.J. Freedman, J. Pettit, Jianying Luo, N. Gude, N. McKeown, and S. Shenker. “Rethinking Enterprise Network Control”. In: *IEEE/ACM Transactions on Networking* 17.4 (Aug. 2009), pp. 1270–1283. ISSN: 1558-2566. DOI: 10.1109/tnet.2009.2026415. URL: http://dx.doi.org/10.1109/TNET.2009.2026415 (cit. on p. 44).

[RDJ11] R. Wang, D. Butnariu, and J. Rexford. “OpenFlow-based Server Load Balancing Gone Wild”. In: *Hot-ICE*. 2011, pp. 12–12 (cit. on p. 44).

[Ngu+16] X. N. Nguyen, D. Saucez, C. Barakat, and T. Turletti. “Rules Placement Problem in OpenFlow Networks: A Survey”. In: *IEEE Communications Surveys Tutorials* 18.2 (2016), pp. 1273–1286 (cit. on p. 44).

[KHK13] Yossi Kanizo, David Hay, and Isaac Keslassy. “Palette: Distributing tables in software-defined networks”. In: *INFOCOM, 2013 Proceedings IEEE*. IEEE. 2013, pp. 545–549 (cit. on p. 44).

[Kan+13] Nanxi Kang, Zhenming Liu, Jennifer Rexford, and David Walker. “Optimizing the One Big Switch Abstraction in Software-defined Networks”. In: *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT 13́. New York, NY, USA: ACM, 2013, pp. 13–24 (cit. on p. 44).

[Ngu+15] Xuan-Nam Nguyen, Damien Saucez, Chadi Barakat, and Thierry Turletti. “OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement”. In: *INFOCOM*. IEEE, Apr. 2015, pp. 478–486 (cit. on pp. 44, 45).

[Kat+16] Naga Katta, Omid Alipourfard, Jennifer Rexford, and David Walker. “CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks”. In: *Proceedings of the Symposium on SDN Research*. SOSR 1́6. New York, NY, USA: ACM, 2016, 6:1–6:12 (cit. on p. 45).

[BK14] S. Banerjee and K. Kannan. “Tag-In-Tag: Efficient flow table management in SDN switches”. In: *CNSM*. 2014, pp. 109–117 (cit. on p. 45).

[KS13] K. Kannan and S. Banerjee. “Compact TCAM: Flow Entry Compaction in TCAM for Power Aware SDN”. In: *ICDCN*. 2013, pp. 439–444 (cit. on p. 45).

[BM14] W. Braun and M. Menth. “Wildcard Compression of Inter-Domain Routing Tables for OpenFlow-Based Software-Defined Networking”. In: *Software Defined Networks (EWSDN), 2014 Third European Workshop on*. Sept. 2014, pp. 25–30 (cit. on pp. 45, 46).

[TNW96] Michael Theobald, Steven M. Nowick, and Tao Wu. “Espresso-HF: A Heuristic Hazard-free Minimizer for Two-level Logic”. In: *Proceedings of the 33rd Annual Design Automation Conference*. DAC 9́6. New York, NY, USA: ACM, 1996, pp. 71–76 (cit. on p. 46).

[Awa+17] Mohamad Khattar Awad, Mohammed El-Shafei, Tassos Dimitriou, Yousef Rafique, Mohammed Baidas, and Ammar Alhusaini. “Power-efficient routing for SDN with discrete link rates and size-limited flow tables: A tree-based particle swarm optimization approach”. In: *International Journal of Network Management* (2017), e1972–n/a (cit. on p. 46).

[GHM15] Frédéric Giroire, Frédéric Havet, and Joanna Moulierac. “Compressing Two-dimensional Routing Tables with Order”. In: *INOC*. 2015, pp. 1–8 (cit. on pp. 49, 51).

[GHM16] Frédéric Giroire, Frédéric Havet, and Joanna Moulierac. “On the Complexity of Compressing Two Dimensional Routing Tables with Order”. In: *Algorithmica* (Nov. 2016). ISSN: 1432-0541. DOI: 10.1007/s00453-016-0243-7. URL: http://dx.doi.org/10.1007/s00453-016-0243-7 (cit. on pp. 49, 51).

[Orl+10] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. "SNDlib 1.0–Survivable Network Design Library". In: *Networks* 55.3 (2010), pp. 276–286 (cit. on p. 55).

[Cha+08] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. "Power Awareness in Network Design and Routing". In: *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*. Apr. 2008 (cit. on pp. 65, 77).

[Nic+12] Luca Niccolini, Gianluca Iannaccone, Sylvia Ratnasamy, Jaideep Chandrashekar, and Luigi Rizzo. "Building a power-proportional software router". In: *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*. USENIX ATC'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 8–8 (cit. on p. 66).

[Idz+16] Filip Idzikowski, Luca Chiaraviglio, Antonio Cianfrani, Jorge López Vizcaı'no, Marco Polverini, and Yabin Ye. "A Survey on Energy-Aware Design and Operation of Core Networks". In: *IEEE Communications Surveys & Tutorials* 18.2 (2016) (cit. on p. 66).

[GMM12] F. Giroire, D. Mazauric, and J. Moulierac. "Energy Efficient Routing by Switching-Off Network Interfaces". In: ed. by Naima Kaabouch and Wen-Chen Hu. IGI Global, June 2012. Chap. 10 - Energy-Aware Systems and Networking for Sustainable Initiatives, pp. 207–236 (cit. on pp. 70, 78).

[Van+12] Ward Van Heddeghem, Filip Idzikowski, Willem Vereecken, Didier Colle, Mario Pickavet, and Piet Demeester. "Power consumption modeling in optical multilayer networks". In: *Photonic Network Communications* 24.2 (2012), pp. 86–102 (cit. on p. 76).

[Ara+16] Julio Araujo, Frédéric Giroire, Joanna Moulierac, Yaning Liu, and Modrzejewski Remigiusz. "Energy Efficient Content Distribution". In: *Computer Journal* (2016), pp. 1–18 (cit. on p. 77).

[Cho+07] Baek-Young Choi, Sue Moon, Zhi-Li Zhang, Konstantina Papagiannaki, and Christophe Diot. "Analysis of point-to-point packet delay in an operational network". In: *Computer networks* 51.13 (2007), pp. 3812–3827 (cit. on p. 84).

[Gir+03] Frédéric Giroire, Antonio Nucci, Nina Taft, and Christophe Diot. "Increasing the robustness of IP backbones in the absence of optical level protection". In: *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies.* Vol. 1. IEEE. 2003, pp. 1–11 (cit. on p. 84).

[Mod16] Sudhir Modali. Apr. 2016. URL: http://www.infoworld.com/article/3061152/networking/break-scalability-barriers-in-openflow-sdn.html (cit. on p. 90).

[ALV08] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. "A Scalable, Commodity Data Center Network Architecture". In: *SIGCOMM Comput. Commun. Rev.* 38.4 (Aug. 2008), pp. 63–74 (cit. on p. 91).

[Gre+09] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. "VL2: a scalable and flexible data center network". In: *ACM SIGCOMM computer communication review.* Vol. 39:4. ACM. 2009, pp. 51–62 (cit. on p. 91).

[Guo+09] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers". In: *SIGCOMM Comput. Commun. Rev.* 39.4 (Aug. 2009), pp. 63–74 (cit. on p. 91).

[Guo+08] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. "Dcell: A Scalable and Fault-tolerant Network Structure for Data Centers". In: *SIGCOMM Comput. Commun. Rev.* 38.4 (Aug. 2008), pp. 75–86 (cit. on p. 91).

[Eri13] David Erickson. "The Beacon Openflow Controller". In: *HotSDN.* ACM, 2013, pp. 13–18 (cit. on p. 106).

[Sha+13] Alexander Shalimov, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov, and Ruslan Smeliansky. "Advanced Study of SDN/OpenFlow Controllers". In: *CEE-SECR.* ACM, 2013, 1:1–1:6 (cit. on p. 106).

[Rif+15] M. Rifai, N. Huin, C. Caillouet, F. Giroire, D. Lopez-Pacheco, J. Moulierac, and G. Urvoy-Keller. “Too Many SDN Rules? Compress Them with MINNIE”. In: *2015 IEEE Global Communications Conference (GLOBECOM)* (Dec. 2015) (cit. on p. 107).

[Cur+11] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. “DevoFlow: Scaling Flow Management for High-performance Networks”. In: *SIGCOMM Comput. Commun. Rev.* 41.4 (Aug. 2011), pp. 254–265 (cit. on p. 107).

[Pfa+15] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. “The Design and Implementation of Open vSwitch.” In: *NSDI*. 2015, pp. 117–130 (cit. on p. 107).

[Kre+15] D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. “Software-Defined Networking: A Comprehensive Survey”. In: *Proceedings of the IEEE* 103.1 (Jan. 2015), pp. 14–76 (cit. on p. 111).

[KPK14] Maciej Kuzniar, P Perešıni, and Dejan Kostic. “What you need to know about SDN control and data planes”. In: *EPFL, TR* 199497 (2014) (cit. on pp. 112, 115).

[PA00] V. Paxson and M. Allman. *Computing TCP’s Retransmission Timer*. RFC 2988 (Proposed Standard). Nov. 2000 (cit. on p. 124).

[Ben+15] S. Bensley, L. Eggert, D. Thalerand P. Balasubramanian, and G. Judd. *Microsoft’s Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters draft-bensley-tcpm-dctcp-05*. Internet-Draft. July 2015 (cit. on p. 124).

Part II

Hybrid networks

CHAPTER 6

Energy Aware Routing for Hybrid Networks

Contents

6.1 Introduction

In the previous chapter, we saw that the SDN paradigm bears the promise of enabling green policies in networks, even with added constraints such as TCAM size. While it is easy to consider networks built from the ground up with SDN in mind, many existing networks are still using legacy equipment, and thus unable to deploy these kinds of solutions. Operators desiring to implement green policies must then consider transitioning their network to SDN-capable equipment.

Different scenarios may be envisioned for the transition from legacy to SDN networks [VVB14]. One of the most realistic is a progressive migration, where legacy hardware is replaced over an extended period of time by SDN hardware. There is thus a coexistence of legacy and SDN, hardware and protocols, in the network. As an example, to route packets inside the network, legacy nodes have to follow legacy protocols, such as OSPF, while SDN nodes may choose the next hops of the packets using an optimization algorithm running in the controller.

In this chapter, we consider the *Energy Aware Routing in a hybrid SDN network* (hEAR) problem which consists in reducing the power usage of a network where both legacy and SDN equipment coexist. To provide energy optimization in hybrid networks, we introduce SENAtoR (Smooth ENergy Aware Routing). The main idea is that the controller first chooses the set of routes that minimizes the number of used network equipment for the current traffic, and then we put SDN nodes in sleep mode (i.e., power save mode which turns off network interfaces). We consider a typical dynamic traffic of an operator and, hence, our solution adapts the numbers of active and inactive network equipment during the day.

When the SDN nodes are put to sleep, and their links are turned off, traffic has to be rerouted, while avoiding packet loss. It is thus impossible to wait for the convergence of legacy protocols (e.g., OSPF). Moreover, if Internet Service Provider (ISP) network traffic usually shows smooth variations of throughput, it also experiences sudden changes which may correspond to (link or node) failures or to flash crowds [Rou+02].Thus, to avoid packet loss, we propose three mechanisms:

1. **Tunneling.** This first mechanism is inspired by the solution proposed in [Chu+15] to handle single link failure. The goal was to avoid waiting for the convergence of legacy routing protocols by using tunnels from a

node with a failing link to an SDN node which can reach an alternative OSPF shortest path in one hop. We reused this idea to reroute using pre-set *tunnels* from any node, with a turned off link, to any other node with a direct path towards the destination which does not include a disabled link.

2. **Turning off links smoothly.** To prevent OSPF routers from sending packets towards a node which was just put into sleep mode by the energy saving mechanism, we propose to force OSPF re-convergence before the Network Interface Card (NIC) at the SDN is turned off. The idea is that the SDN controller discards any OSPF packet sent on the node to be disabled to simulate a node failure while any other data packet must be properly processed and forwarded. After a period of time, greater than the link failure detection period and than the convergence of OSPF (which can be estimated with the OSPF timer values), and if no more traffic is received, the SDN router will effectively turn off the appropriate NICs. Note that while OSPF has not converged yet, packets can be rerouted through the pre-set tunnels; and since the link and node are still on, packets are not lost during the routing transition.

3. **Traffic Spike and link failure mitigation** Network capacity over-provisioning is exploited by energy aware algorithms to save energy. Indeed, networks are oversized, in particular, to handle traffic variations due, e.g., to link failures or flash crowds. It is thus of crucial importance for energy saving mechanisms, which turn off pieces of equipment, to not impact the failure tolerance of networks. We exploit the metrology data received by the controller from SDN nodes to detect significant traffic variations and react to them.

Our contributions are the following:

- We propose several mechanisms to bring energy aware solutions closer to reality in ISP networks to avoid packet losses when putting network devices into sleep mode: tunneling, smooth shutdown of links, and detection of traffic variations.

- We model and formulate the Energy Aware Routing in a hybrid SDN network (hEAR) problem as an ILP.

- To validate the solutions, we carried out extensive simulations on several network topologies and show the energy savings for different levels of SDN penetration.
- The mechanisms were implemented and tested on a small SDN platform. The results of the experimentations show that it is *possible to implement energy saving solutions while reducing packet losses* compared to legacy protocols.

6.2 Related work

6.2.1 Hybrid SDN Networks

As the most realistic scenario for the introduction of the SDN paradigm is a gradual migration, we focus on hybrid networks. In these networks, legacy and SDN hardware stand alongside. The difficulty is to make different protocols coexist. Opportunities and research challenges of Hybrid SDN networks are discussed in [VVB14]. Routing efficiently in hybrid networks has been studied in [AKL13]. The authors show how to leverage SDN to improve link utilization, reduce packet losses and delays. We extend this work by considering energy efficiency.

6.2.2 Handling Failures and Flash Crowds

Turning off SDN devices in hybrid IP-SDN networks can be interpreted as link or node failures by legacy network devices and might decrease the network ability to drain sudden, yet not malicious, traffic surges (due, for instance, to exceptional events such as earthquakes). Consequently, our energy-aware solution implements some features to cope with link failures and flash crowds correctly. The network community has addressed such problems, with the help of SDN, as follows.

Link Failure Detection and Mitigation

As in legacy devices, SDN devices can rely on the legacy Bidirectional Forwarding Detection (BFD) algorithm to detect link failures [KW15]. Once the link failure has been detected, OpenFlow already offers a link failure mitigation technique through the notion of FAST-FAILOVER group rules, where several rules per flow can be installed. Protection of the link and control channel of OpenFlow requires, however, more complex solutions, like the one proposed in [Sha+16]. To avoid losses in case of link failures in hybrid networks, [Chu+15] proposes to introduce pre-set tunnels from a legacy router towards an SDN router, which form backup paths. Later, SDN nodes reroute traffic through non-damaged paths. We borrow this idea and propose to use pre-set tunnels when a node is turned down. This is an adaptation and a generalization of the solution offered in [Chu+15] to handle a link failure. Indeed, we use it for energy efficiency when multiple links are turned off. We also allow tunnels to be set between any (OSPF or SDN) pair of nodes and we carry out practical experimentations to validate the method.

Detecting Traffic Variations in SDN Networks

Traffic variations of backbone networks are usually smooth as the network traffic is an aggregation of multiple flows [Rou+02; Ian+01]. However, abrupt variations happen in case of link failures or flash crowd [LCD04]. Methods have been proposed to detect them in legacy networks, see for example [LCD05; Ari+03]. Netfuse [Wan+13] has been proposed in SDN-based data centers to mitigate the effect of traffic variations.

6.3 Energy Aware Routing for Hybrid Networks

6.3.1 Routing in a Hybrid Network

A network is modeled as a directed graph $D = (V, A)$ where a node represents a Point of Presence (PoP) and an arc represents a link between two PoPs. A PoP consists of several routers linked together in full mesh [Gir+03]. Each link $(u, v) \in A$ is connected to a specific router in PoP u and in PoP v, see Figure 6.1. A link (u, v) has a maximum capacity C_{uv}. In each PoP, it exists a switch that is connected to exactly one other PoP. We consider hybrid networks in which SDN capable equipments are deployed alongside legacy routers. This kind of networks can be found in the transition from legacy networks to a complete Software Defined Network. We consider a scenario in which PoPs do not contain heterogeneous equipments, i.e, all routers of a PoP are either SDN capable or not. Legacy routers follow a legacy routing

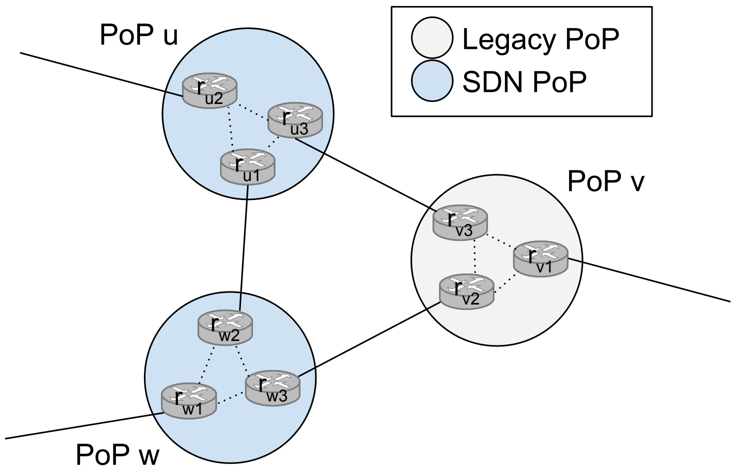


Figure 6.1: 3 PoPs interconnected in an hybrid network.

protocol, such as OSPF. We denote the next hop to the destination t on a legacy router u by $n^t(u)$. SDN switches are controlled by one or several central controllers and can be configured, dynamically, to route to any of its neighbors.

6.3.2 Traffic Estimation

We assume that an ISP can estimate the traffic matrix of its network using (sampled) NetFlow measurements [B C04] or, in the case of hybrid networks, by combining SDN and OSPF-TE data [AKL13]. Therefore, our solution monitors traffic and continuously calculate the set of nodes or links to turn off.

6.3.3 Power Model and Energy Aware Mechanisms

To model the power consumption of a link, we use a hybrid model comprised of a baseline cost, representing the power used when the link is active, and a linear cost depending on its throughput. This allows, by changing the value of the parameters, to express different power models (between ON-OFF and proportional to the load) found in the literature, see [Idz+16] for a discussion. The power usage of a link is expressed as follows

$$P_l(u,v) = x_{uv} P_{uv}^{\text{IDLE}} + \frac{\mathcal{F}_{uv}}{C_{uv}} P_{uv}^{\text{LOAD}}$$

where x_{uv} represents the state of the link (ON or OFF), P_{uv}^{IDLE} is the baseline power consumption of an active link, $\mathcal{F}_{uv}$ the total amount of bandwidth on the link, and P_{uv}^{LOAD} the additional power consumption of the link when at full capacity. Routers have two power states: active or sleep, and their total consumption $P_n(u)$ is given by

$$P_n(u) = B_u + r_{uv}A_u + \sum_{v \in N^+(u)} P_l(u,v)$$

where r_{uv} represents the state of the router (ON or SLEEP), B_u is the sleep state power usage and A_u the additional power used when the equipment is active.

To save energy, we must power down links and put routers to sleep. We can only put SDN switches into sleep mode without negative impact on the network. As it should be done dynamically according to the network traffic, the decision is taken by the SDN controller. Thus, only links with a SDN switch as one of its end points can be shut down. Since PoPs are interconnected using dedicated routers inside their infrastructure, if a link between two PoPs is shut down, then each router of the link can be shutdown, if it is SDN capable.

6.4 Integer Linear Program

We propose the following Integer Linear Program to solve the hEAR-with-tunnel-selection problem. A summary of the notations is found in Table 6.1. The formulation presents several difficulties. First, legacy nodes have to route flows through shortest paths following legacy protocols, when SDN nodes can route a flow freely to any neighbors. Second, tunnels have to be set in a way there exists a path for each flow, even when several network equipment are put into sleep mode.

We want to minimize the power consumption of the network (6.1) with at most k SDN PoPs (6.2).

$$\min \quad \sum_{u \in V} P_n(u) \tag{6.1}$$

$$\sum_{u \in V} s_u \leq k \tag{6.2}$$

D^{st}	charge of the demand between s and t.
C_{uv}	capacity of link (u,v)
$\mathcal{P}$	set of all paths
$p(s,t)$	set of path between s and t
$\mathcal{T}$	set of all destinations
$n^t(u)$	OSPF next hop on node u for destination t

(a) Input parameters

$f^{st}_{uv} \in \{0,1\}$	demand between s and t is forwarded without tunnels between u and v
$g^{st}_{p} \in \{0,1\}$	demand between s and t is forwarded on the tunnel p
$h^{t}_{ux} \in \{0,1\}$	a tunnel to x from u for packets with destination to t is used
$n^{st}_{uv} \in \{0,1\}$	v is the next hop on u for the demand between s and t
$x_{uv} \in \{0,1\}$	link (u,v) is on
$s_u \in \{0,1\}$	u is an SDN PoP
$e^{st}_{u} \in \{0,1\}$	next hop on u for demand between s and t is inactive.
$r_{uv} \in \{0,1\}$	router in PoP u connected to PoP v is on

(b) Decision variables

Table 6.1: Notations used for the ILP

The flow conservation constraints are given by Equation (6.3) and the link capacity constraints by Equation (6.4).

$$\sum_{\forall w \in V \setminus \{u\} p(u,w) \in \mathcal{P}} g^{st}_p - \sum_{\forall w \in V \setminus \{u\} p(w,u) \in \mathcal{P}} g^{st}_p + \sum_{v \in N^+(u)} f^{st}_{uv} - \sum_{v \in N^-(u)} f^{st}_{vu} = \begin{cases} 1 & \text{if } u = s, \\ -1 & \text{if } u = t, \\ 0 & \text{else} \end{cases} \quad \forall (s,t) \in \mathcal{D}, u \in V \tag{6.3}$$

$$\sum_{(s,t) \in \mathcal{D}} D^{st} \left(f^{st}_{uv} + \sum_{\{p \in \mathcal{P} | (u,v) \in p\}} g^{st}_p \right) \leq x_{uv} C_{uv} \qquad \forall (u,v) \in A \tag{6.4}$$

At each node, only one tunnel destination can be defined for a destination t

$$\sum_{w \in V} h^t_{uw} \leq 1 \qquad \forall u \neq t \in V \tag{6.5}$$

Each demand (s,t) can be forwarded through a tunnel from u to $w, \forall w \in V \setminus \{u\}$ if all links of the tunnel are not turned off (6.6), the link to the next hop of u to destination t is off (6.7) and the tunnel destination allowed on u for the destination t is w (6.8).

$$g_p^{st} \leq n_{v_1 v_2}^{uw} \qquad \forall p(u,w) \in \mathcal{P}, (v_1, v_2) \in p, (s,t) \in \mathcal{D} \tag{6.6}$$

$$g_p^{st} \leq e_u^{st} \qquad \forall p(u,w) \in \mathcal{P}, (s,t) \in \mathcal{D} \tag{6.7}$$

$$g_p^{st} \leq h_{uw}^{t} \qquad \forall p(u,w) \in \mathcal{P}, (s,t) \in \mathcal{D} \tag{6.8}$$

If the next hop of u for the demand (s,t) is v and the link (u,v) is off, $e_u^{st} = 1$, i.e., the next hop link is inactive (6.9). However, if the next hop of u for the demand (s,t) is v and the link (u,v) is on, $e_u^{st} = 0$, the next hop link is active (6.10).

$$n_{uv}^{st} - x_{uv} \leq e_u^{st} \qquad \forall (u,v) \in A, (s,t) \in \mathcal{D} \tag{6.9}$$

$$e_u^{st} \leq 2 - n_{uv}^{st} - x_{uv} \qquad \forall (u,v) \in A, (s,t) \in \mathcal{D} \tag{6.10}$$

Each node can only have one next hop per destination (or per demand if the node is SDN) (6.11). Only SDN nodes can have a different next hop than the OSPF next hop $n^t(u)$ (6.12).

$$\sum_{v \in N^+(u)} n_{uv}^{st} \leq 1 \qquad \forall u \in V, (s,t) \in D \tag{6.11}$$

$$n_{uv}^{st} \leq s_u \qquad \forall (u, v \neq n^t(u)) \in A, (s,t) \in \mathcal{D} \tag{6.12}$$

Both arcs of a link share the same state (6.13) and a link can only shutdown if one of its endpoints is an SDN nodes (6.14).

$$x_{uv} = x_{vu} \qquad \forall (u,v) \in A \tag{6.13}$$

$$x_{uv} \geq 1 - s_u - s_v \qquad \forall (u,v) \in A \tag{6.14}$$

A router u connected to a router v can only be put into sleep mode if it is an SDN switch (6.15) and if the link (u,v) is off (6.16).

$$r_{uv} \geq 1 - s_u \qquad \forall (u,v) \in A \tag{6.15}$$

$$r_{uv} \geq x_{uv} \qquad \forall (u,v) \in A \tag{6.16}$$

The ILP can be used to find good solutions for small-sized instances, see Section 6.6. The computation time is however prohibitive to find an optimal solution as the problem is NP-complete (indeed, it comprises as subproblem the EAR problem which is NP-complete [Gir+10]). For larger instances, it is even impossible to find feasible solutions using the ILP. We thus propose an efficient heuristic algorithm to solve the hEAR problem.

6.5 Smooth ENergy Aware Routing (SENAtoR)

6.5.1 Mechanisms

SENAtoR uses three mechanisms to avoid packet losses in the network due to shutting down equipment and limit packet losses due to link failures or flash crowds.

Tunneling

Shutting down a link with the SDN controller results in a failure detection by OSPF and a convergence period. To avoid losing packets during the re-convergence phase, we use pre-set tunnel backup paths to redirect traffic that would otherwise be lost. The idea is to reroute the traffic that would use this down link or node to an intermediate node whose shortest path to destination does not use down links.

As with most legacy network mechanisms, tunnels cannot be deployed dynamically during the operation of the network. They have thus to be pre-set statically. We thus consider two variants of the problem: (*i*) with tunnel selection, (*ii*) with a pre-configured set of tunnels.

Turning off links smoothly

Before putting an SDN PoP switch in power-save mode, the SDN controller stops sending any OSPF packet to its neighbors. This allows neighboring OSPF routers to converge to a network view excluding this node. Indeed, after the default *dead interval* of 3 $\times$ *hello interval* without receiving any Hello packet, an OSPF router declares its neighbor as dead and stops using the link. However, until the end of the dead interval, the link is considered to be active, and traffic flows over this link. After the dead interval, plus a safety margin of 10 additional seconds, and if no traffic is received through its links (that we define as the OSPF expected convergence period), the SDN

PoP switch is put in power-save mode. This simple strategy prevents any additional packet loss.

Traffic spikes and link failures mitigation

Sudden traffic spikes are relatively rare due to the high statistical multiplexing in the backbone of ISPs. However, exceptional events (such as earthquakes) can lead to flash crowds. Therefore, we complement SENAtoR with a safeguard mechanism that aims at reactivating inactive SDN PoP switches in case of a sudden traffic spike, in a similar fashion to [Car+16]. The latter event is defined on a per link basis as follows: the controller is collecting the traffic load on each interface of every SDN active switch at a small time scale (in our experiments, once per minute). We then compare the real traffic level received at interface i, $E_i(t)$, to the estimated rate, $E_i^{ES}(t)$, at the last epoch where SENAtoR took its decision of turning off some links. In case $E_i(t) \geq 1.5 \times E_i^{ES}(t)$, for any interface i, all inactive SDN routers are re-enabled. The value of 50% was chosen conservatively, since, in general, ISP networks are over-provisioned. After the OSPF expected convergence period, the controller reruns SENAtoR to obtain a new green architecture if possible.

We employ a similar mechanism in case of link failures. When a link connected to an active SDN router or a link between OSPF nodes fails, SENAtoR turns on again any inactive SDN node. It also directly reroutes the traffic through a different path if possible (including the pre-set tunnels).

Nearby SDN nodes can detect a link failure in between OSPF nodes due to the traffic variation at their network links. A downstream link, in regards to a failed link, will indeed observe a decrease of the rate of one interface as compared to what the traffic matrix predicts. We benefit from the fact that in typical ISP networks, traffic is all-to-all, i.e., from one PoP to any other PoP. Hence, any SDN router in the network is likely to detect the link loss, as a fraction of the traffic it handles is affected by the failure. Again we use a conservative threshold of 50%, i.e., an SDN switch must detect a decrease of 50% of any of its links' load to trigger the link failure mitigation mechanism. Once again, after the OSPF convergence expected period, the controller re-uses SENAtoR to obtain a new green architecture eventually.

6.5.2 Heuristic

SENAtoR's heuristic has two steps: first, it assigns routes to the flows using eventual tunnels, then it selects the equipment to turn-off. During the second step, we may need to find a new path for demands that were using the device that we try to shutdown. Note that two possibilities for the configuration of the tunnels are considered, (i) with dynamic tunnel selection, (ii) with a pre-configured set of tunnels.

Path assignment

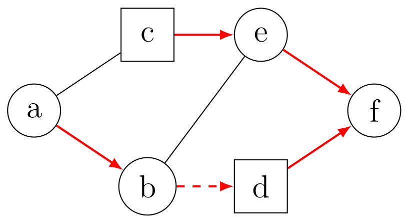
To assign a path to a demand, we build a weighted residual graph $H_{st} = (V, A')$ and then search for the shortest path between s and t in H_{st}. Nodes in H_{st} are the ones of D and correspond to network routers. We only consider links and tunnels which have enough residual capacities to satisfy the demand D_{st}. For each node u, its set of out-neighbors is constructed as follows:

If u is a legacy node, the routing is done by the legacy routing protocol towards next hop $n^t(u)$ if the link to $n^t(u)$ is active. In this case, the only neighbor of u in H_{st} is $n^t(u)$. Otherwise, if the link to $n^t(u)$ is inactive, the routing is done through a tunnel. (1) If a tunnel from u is already defined for the destination t, the neighbor of u in H_{st} is set as the tunnel endpoint. (2) If no tunnel is defined, the next step depends on the variant of the problem. (2i) In the tunnel selection variant, we have to set a tunnel. We thus add all the potential tunnels by adding any node that can reach the destination t, using direct forwarding (OSPF or OpenFlow) or existing tunnels. (2ii) With pre-configured set of tunnels, u has no neighbor in H_{st}.

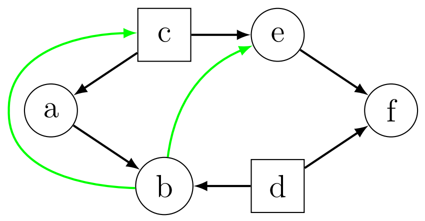
If u is an SDN node, the routing is done by OpenFlow rules installed by the controller. We have two cases: (1) if no OpenFlow rule is set for the demand in node u, any neighbor can be the next hop. The neighbors of u in H_{st} are the same as in the original digraph D. (2), we only add as a neighbor of u in H_{st} the node designed as the next hop by OpenFlow. Similar to a legacy node, if the link to the next hop given by OpenFlow is inactive, we consider tunnels in the same way described above.

We then compute a weighted shortest path from s to t in the residual graph H_{st} leading to the decision of which tunnel will be selected and whether we need to install or not a new OpenFlow rule for the SDN node.

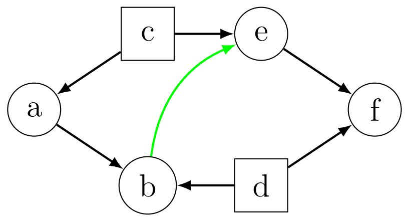
We present in Figure 6.2 an example of the construction, for the two variant, of the residual graph H_{af} needed to find the path of the demand



(a) Original network



(b) H_{af} for the dynamic tunnel selection variant



(c) H_{af} for the pre-configured set of tunnels variant. e is already set as the tunnel for destination f on node b

Figure 6.2: Example of the Path Assignment step of the heuristic for a demand between a and f when the link (b, d) is inactive. Red arcs represent the OSPF next hops to f and tunnels are represented by green arcs. Circle nodes represent OSPF nodes and square nodes, SDN nodes.

between a and f. For both variant, all OSPF nodes in H_{af} (a, e, f) are only connected to their next hop except for b, since the link (b, d) is off. Both SDN nodes (c, d) are connected to all of their neighbors in the network since no matching rule exists for the demand a and f. The set of arcs on node b corresponding to tunnels is different between the two variant. In Figure 6.2b, since no tunnel end point has yet to be set for destination f, we add all possible valid tunnel end points. Node a is not a valid end-point as it cannot reach f using non-tunnels arcs. Nodes d and f are not valid end point because they cannot be reached from b since the link (b, d) is off. In Figure 6.2b, only the tunnel to e is present since it was pre-set.

Off link Selection

When the routing module assigns a path to all demands, the energy savings module tries to power off links to save energy. We consider SDN links one by one, i.e., links with at least one SDN endpoint. We select the active link with the smallest amount of traffic on both arcs. We then try to reroute all the demands flowing through that link. If the routing module finds no valid routing, we set the link as *non-removable*, and restore the former routing. Otherwise, we set the link as *inactive* and powered off. We then consider the remaining active links. The heuristic stops when all SDN links are either powered off or *non-removable*.

6.6 Numerical results

In this section, we evaluate the solutions proposed on different ISP topologies. We first compare the performances of the ILP and the heuristic algorithm on a small topology. We then use SENAtoR on larger networks of SNDLib. We show that *SENAtoR obtains energy savings that range from 5% up to 35% for different levels of SDN hardware installation.*

For the parameters of the power model, we considered the cases of two different hardware: our HP5412zl SDN switch and an *ideal energy efficient* SDN switch as discussed in [Vu+14]. In the first case, we measured the power consumption using a wattmeter: the switch uses 95 W when in sleep mode and 150 W if it is active ($B_u = 95, A_u = 55$). According to Cisco specifications [Cis14], links are using 30 W as a baseline and go up to 40 W when at full capacity ($P_{uv}^{\text{IDLE}} = 30, P_{uv}^{\text{LOAD}} = 10$). To have a fast recovery

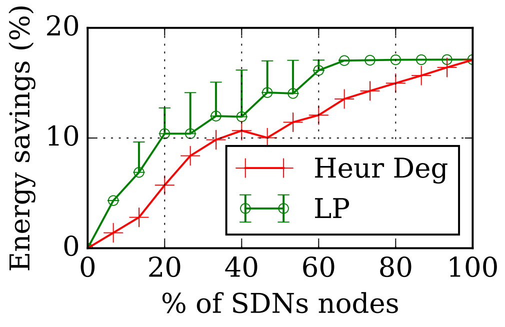


Figure 6.3: Energy Savings for the ILP and the heuristic on `atlanta`

from sleep mode, the TCAM must be kept under power to preserve the forwarding rules. According to [Con+14], TCAM represents 30% of the consumption of a high-end router, and considering results from [Vu+14], we can safely assume that an *ideal energy efficient* switch could save up to 60% of energy in sleep mode.

For the choice of SDN nodes in the networks, we tested and evaluated different methods such as node degree, centrality, and covering (*betweenness centrality*, *closeness centrality* and MAX k-VERTEX COVER). Finally, we chose the simplest one in terms of computation that gives similar results: the highest *node degree*. The resulting selection is first sort all nodes according to their degree; second, choose the k first nodes. This method has the advantages of being simple and allowing a good incremental upgrade to SDN hardware.

6.6.1 ILP vs. SENAtoR

We use the `atlanta` network (composed of 15 nodes and 22 links) and the traffic matrices provided by SNDLib to compute the energy savings for different numbers of SDN nodes. We solve the ILP with CPLEX and set a time limit of one hour (as the ILP is complex). The results presented correspond to the best solution found by the solver within the time limit. Note that for percentages of SDN nodes below 13% and greater than 73%, the ILP solves the problem optimally in less than one hour. The heuristic takes at most 5 ms to find a solution in all settings.

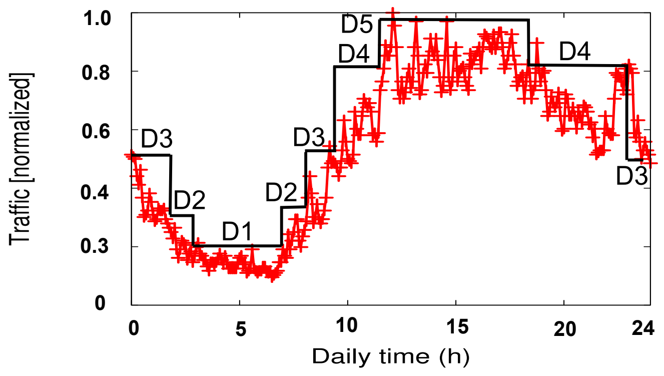


Figure 6.4: Daily traffic in multi-period.

6.6.2 Simulations on Larger Networks

We further look at the performance of the heuristic on `atlanta` and on larger networks such as `germany50` (50 nodes and 88 links), `zib54` (54 nodes and 81 links) and `ta2` (65 nodes and 108 links).

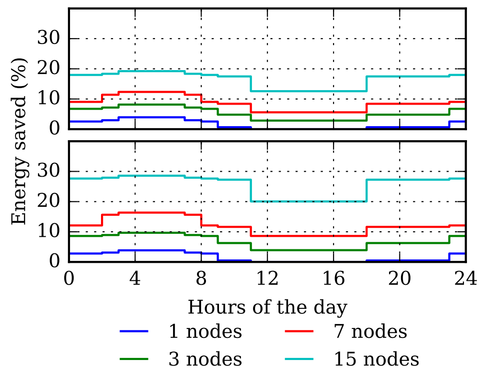
Traffic model

Since ISP traffic is roughly stable over time with clear daily patterns, a few traffic matrices are enough to cover a whole day period. Consequently, a relatively small number of routing reconfigurations allows operators to obtain most of the energy savings [Idz+16] and avoid making frequent reconfigurations. Indeed, as exemplified by the daily variations for a typical link in the Orange ISP network (Figure 6.4), five traffic matrices (labeled D1 to D5) are sufficient. These matrices are normalized and adapted to the size of each studied topology.

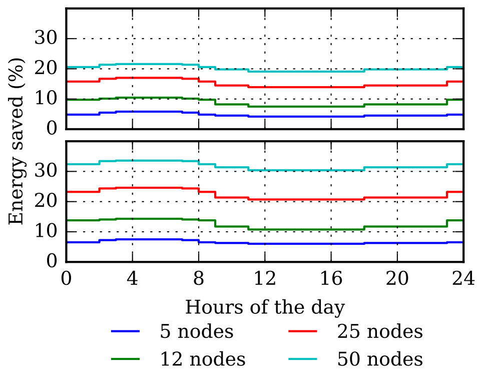
Then, we compute the best hybrid energy aware routing for each matrix and adapt the routing when the traffic changed.

Daily savings

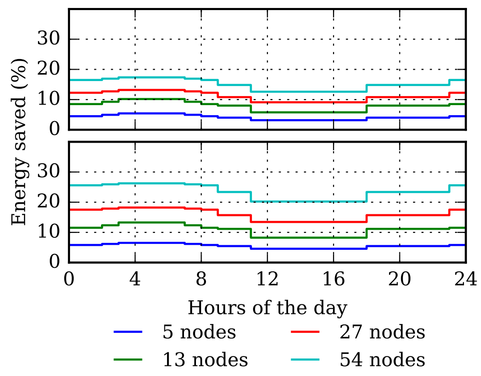
In Figure 6.5, we compare the energy savings during the day for the four topologies. The top figures represent the savings with HP switches and the bottom ones the savings with *ideal energy efficient* switches. We look at four different levels of SDN deployment: 10%, 25%, 50% and 100% of upgraded nodes in the network. For each period, we compare the energy used to the



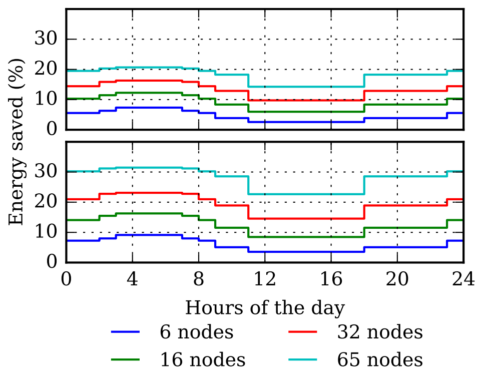
(a) `atlanta`



(b) `germany50`



(c) `zib54`

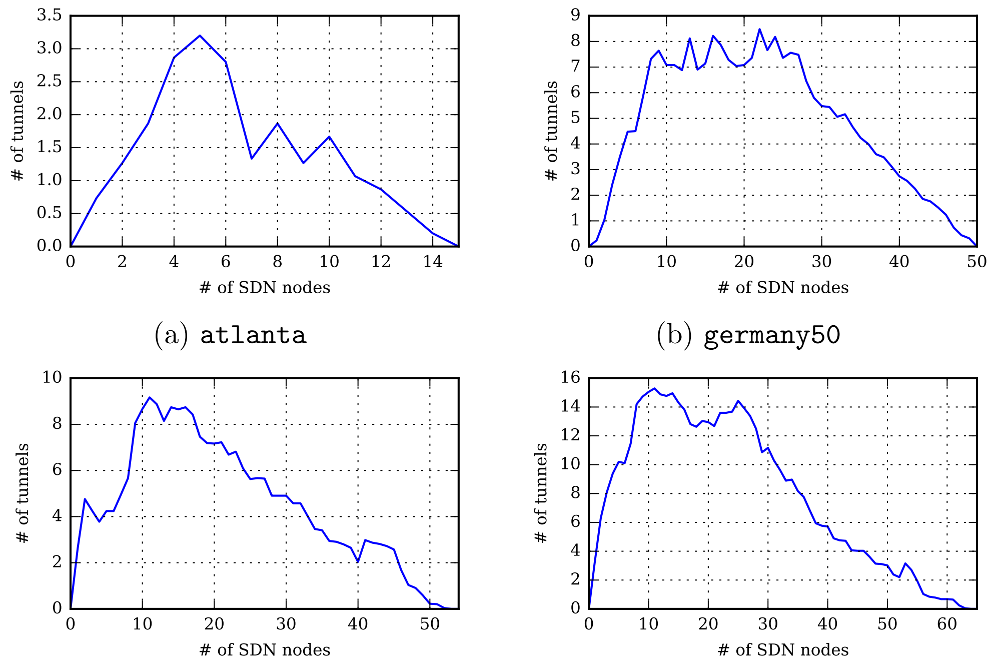


(d) `ta2`

Figure 6.5: Daily energy savings over the day for the (a) `atlanta`, (b) `germany50`, (c) `zib54` and (d) `ta2` networks. with 10, 25, 50 and 100% SDN nodes deployment. Top plots: power model of the HP switch. Bottom plots: power model of an ideal energy efficient SDN switch.

one of a legacy network at the same period. On a full SDN network, the difference between night and day energy savings is between 2% and 7% (3.5 and 9% with *ideal switches*). With HP switches, we can save up to 19% on `atlanta`, 22% on `germany50`, 17% on `zib54` and 21% on `ta2` with a full SDN networks. With *ideal* switches, we obtain higher savings, between 25% and 35%.

Number of tunnels



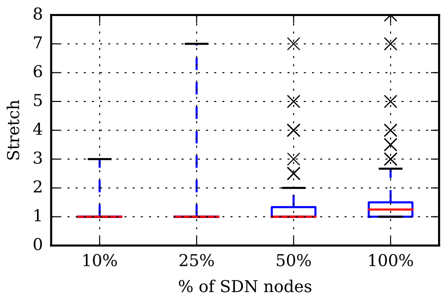
(a) `atlanta` (b) `germany50`

(c) `zib54` (d) `ta2`

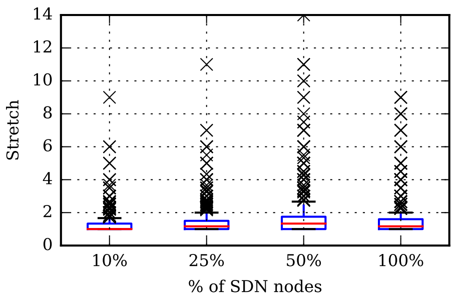
Figure 6.6: Number of average tunnels installed per node on the (a) `atlanta`, (b) `germany50`, (c)`zib54` and (d) `ta2` networks

We look at the number of tunnels used in Figure 6.6. For small SDN budgets (up to 30% of the network for `atlanta`, 20% for larger networks), the average number of tunnels significantly increases with the number of SDN nodes. The reason is that more network links may be turned off, and thus, more backup tunnels are needed. The number of tunnels then levels off and decreases. Indeed, with a high penetration of SDN in the network, SDN nodes can dynamically forward the traffic regardless of OSPF, and the traffic can be rerouted before arriving at the turned-off link. Thus, fewer backup tunnels are needed. The maximum average number of tunnels required per node is proportional to the size of the network (3 for `atlanta`, 8 for `germany50`, 9 for `zib54` and 15 for `ta2`). Finally, while the number of tunnels needed may seem high, we see in the next section that the impact of this overhead on the network performance (packet loss or delay) is not noticeable.

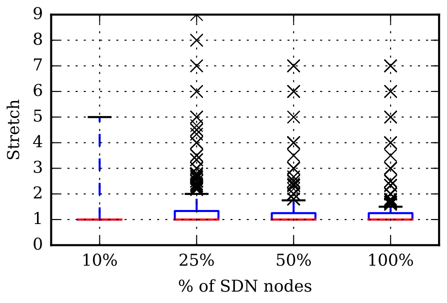
Stretch and delay



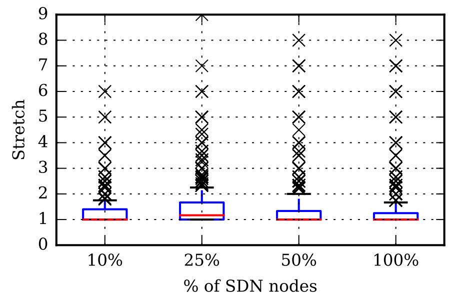
(a) `atlanta`



(b) `germany50`



(c) `zib54`

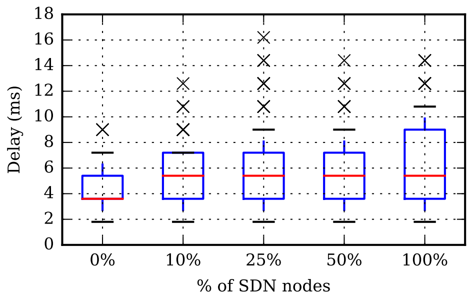


(d) `ta2`

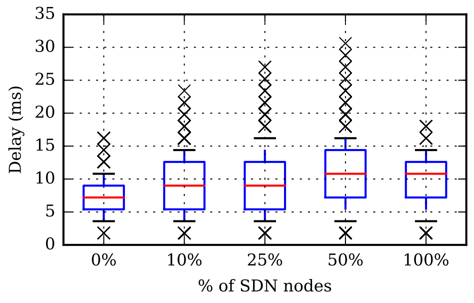
Figure 6.7: Stretch ratio for four different levels of SDN deployment on (a) `atlanta`, (b) `germany50`, (c) `zib54` and (b) `ta2` networks. The box represents the first and third quartiles and whiskers the first and ninth deciles.

By nature, Energy Aware Routing has an impact on the length of the route in the network. As we turn off links, we remove shortest paths. Moreover, tunnels can also increase the path length. In Figure 6.7, we show the stretch ratio of the paths for four levels of SDN deployment. We only show the stretch for the period with the lowest amount of traffic, as it is the period with the largest number of turned off links and thus the one with the largest stretch.

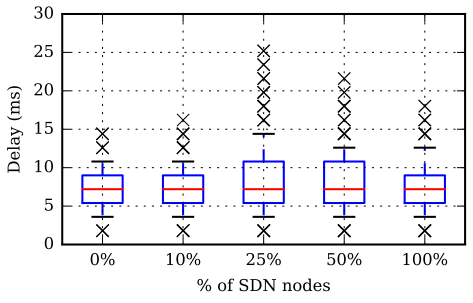
Most of the demands are barely affected by SENAtoR. The median stays around a ratio of 1 with a maximum of 1.25 for `atlanta` at 100% deployment, 1.25 for `germany50` at 50% deployment, 1.33 for `zib54` at 10%, and 1.25 for `ta2` at 25%. 90% of the paths have at most a ratio less than or equal to 3. The stretch of the paths follows the same behavior as the number of tunnels needed for a valid hEAR. Below a 50% deployment, we need an increased number of tunnels to forward the traffic, and thus, we also increase the length of the paths. On a full SDN network, we only see the stretch due to powered off links.



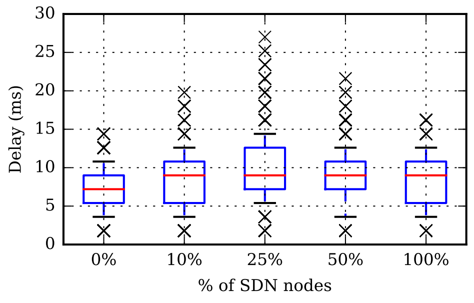
(a) `atlanta`



(b) `germany50`



(c) `zib54`



(d) `ta2`

Figure 6.8: Delays for the demands in the (a) `atlanta`, (b) `germany50`, (c) `zib54` and (b) `ta2` networks.

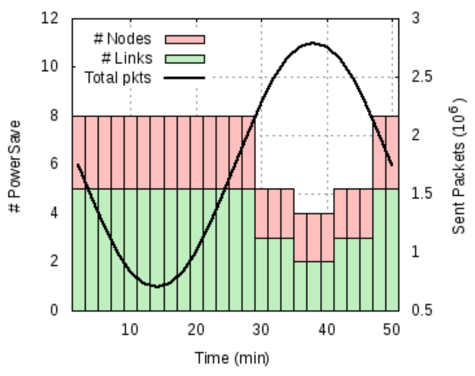
Even though some paths reach a stretch ratio of 14 on `germany50` and 9 on `zib54`, we can see in Figure 6.8 that the network delay stays relatively small. Indeed, the paths with a big stretch are mostly one-hop paths that used to be on currently inactive links. To compute the delays, as the delay is proportional to the distance in an optical network [Cho+07], we use the distances given by the geographical coordinates in SNDlib for the `germany50` network. We got an average value of 1.8 ms per link. Since the coordinates are not available for the other topologies, we used the same average value for `atlanta`, `zib54`, and `ta2`. The median delay rarely goes above 10 ms for all four networks. The `zib54` network experiences the worst delay with a half SDN deployment, with almost 35ms of delay. *The bottom line is that using SENAtoR, we stay below a delay of 50ms.* This is important, as this value is often chosen by SLAs as the maximum delay allowed for a route in a network [Faw+04]. Thus, even if new routes computed by our algorithms may sometimes experience a high value of stretch, this will not be a problem for network operators.

6.7 Experimentations

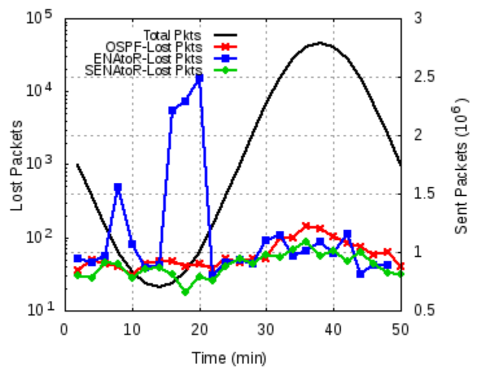
In this section, we present results obtained on a Mininet testbed with the SENAtoR solution. Our objective is to demonstrate that SENAtoR can indeed turn off links and put SDN switches in power save mode without losing packets thanks to our smooth integration with OSPF to anticipate link shutdown.

6.7.1 Testbed Description

We built a hybrid SDN testbed using Mininet [Min] and a remote Floodlight [Flo] controller. The Mininet network topology is based on `atlanta` with 50% SDN deployment. OSPF routers are materialized as host nodes in Mininet and run the Quagga software [Qua] while OVSes [Pfa+15] act as SDN switches. Our Floodlight controller is able to parse and respond to OSPF hello packets received and forwarded by the SDN OVSes (through adequate OpenFlow rules installed in the SDN switches) ; hence ensuring the correct functioning of the adjacent OSPF routers. We use Generic Routing Encapsulation (GRE) tunnels, and tune the administrative distance, so that regular interfaces have a higher priority, to control the interplay between the tunnel interface and the regular interfaces. When SENAtoR notifies an SDN PoP switch to get into sleep mode, we turn off all of its interfaces and disconnect it from the rest of the network. During this power-save mode,



(a) Number of turned off links



(b) Packet loss

Figure 6.9: `atlanta` topology

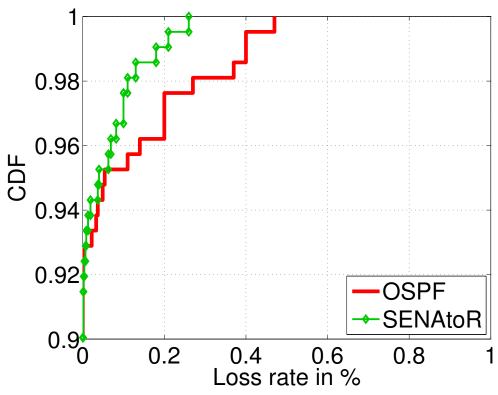
the memory keeps the set of rules previously installed by the controller to perform a quick recovery back to normal active mode.

6.7.2 Experimental Results

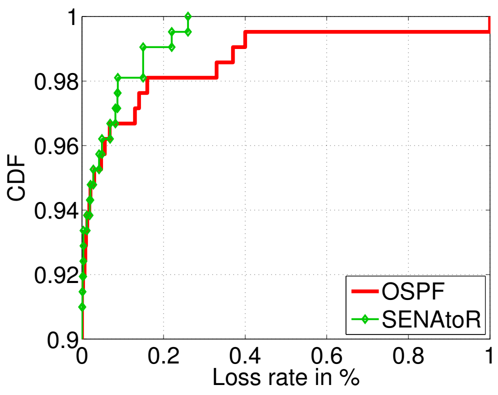
Lossless link-turn-off

In Figure 6.9a we vary the traffic over time to simulate smooth variations on the average rate. This is achieved by taking one traffic matrix and scaling it using the same sinusoidal function as in Figure 6.4.

The energy saving results in Figure 6.9a are in line with the ones presented in Section 6.6, i.e., the same number of links and nodes are turned off in all cases. The added value of the experiment is to assess if our smooth link shutdown approach to enable EAR is effective. Figure 6.9b portrays the time series of packet losses with pure OSPF (OSPF operates the complete network, and no link is turned off in this case), SENAtoR and ENAtoR (SENAtoR without the smooth link shutdown). This shows the importance of SENAtoR anticipation of links shutdown resulting from putting SDN switches in sleep mode. Without it, losses explode to 10^4 packets (see ENAtoR). In this case, the high loss rate of ENATOR is proportional to the amount of time it takes for OSPF to declare the link down multiplied by the traffic intensity. In contrast, SENAtoR manages to maintain the same packet loss as a full OSPF network without any links shutdown, with negligible loss rates (10^{-4}%), even though it is using fewer links and nodes in the network.



(a) On an SDN-OSPF link



(b) On an OSPF-OSPF link

Figure 6.10: Loss rate percentage during a traffic spike in the `atlanta` topology

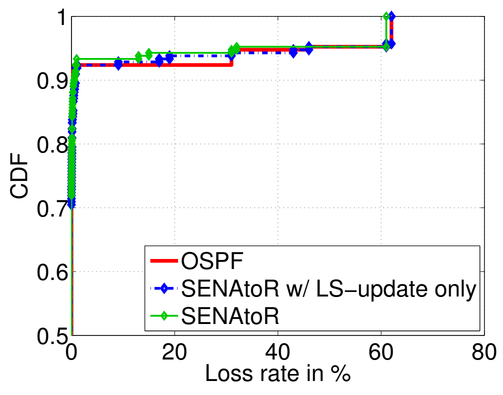
Traffic spikes

To illustrate the traffic spike mitigation mechanism, we consider a fixed traffic matrix (no scaling) and we induce a traffic spike either at an OSPF node directly connected to an SDN switch (Figure 6.10a) or between OSPF nodes (Figure 6.10b). We report the Cumulative Distribution Function (CDF) of loss rates of all connections. The spike detection algorithm of SENAtoR allows it to outperform OSPF. One of the reasons for such a phenomenon is that regular OSPF nodes have no mechanisms to load balance automatically packets in case of traffic spikes.

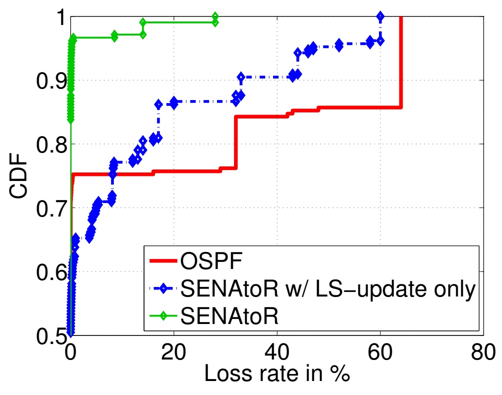
Link failure

We consider again a fixed traffic matrix (no scaling) and we induce a link failure either between an SDN switch and an OSPF router or in between two OSPF routers and report the corresponding loss rates on Figures 6.11a and 6.11b. We compare three cases: (*i*) the legacy OSPF scenario, in which the link failure is handled with a long convergence time, (*ii*) SENAtoR using OSPF Link State (LS) Updates only to detect network changes; and (*iii*) SENAtoR with its *Link failure* detection and mitigation mechanism.

We first observe that even in case (*ii*), *SENAtoR does not experience higher loss rates than case (i)* (and significantly lower loss rates when failure on OSPF-OSPF link). This happens *even though some of the switches and*



(a) On an SDN-OSPF link



(b) On an OSPF-OSPF link

Figure 6.11: Link failure experiment with the `atlanta` topology

links were down at the time of the failure, and had to be switched on. Indeed, SDN switches do not need to wait for the OSPF convergence before rerouting traffic through the pre-established set of tunnels. The link failure mitigation mechanism further improves the situation.

We then observe a counterintuitive result: the loss rates using SENAtoR are smaller when the failure occurs on an OSPF-OSPF link rather than on an SDN-OSPF link. Two factors contribute to this result. First, we placed SDN nodes at key locations in the network where they convey more traffic. A failure at these nodes thus induces higher loss rates. Second, as soon as a downstream SDN node detects a link failure in an OSPF-OSPF link, SENAtoR limits the traffic flowing on this link by instructing upstream SDN nodes to reroute their traffic.

6.8 Conclusion

In this chapter, we presented SENAtoR, an energy aware routing solution that preserves failure tolerance and traffic overload management of the network. We enriched SENAtoR with lossless link/node turn-off, spikes, and traffic failure detection services. SENAtoR implementation and experimentation with emulated devices running full OSPF agents show that we can deal with unexpected network events correctly. More strikingly, our experiments show that even when green services are enabled, and traffic spikes occur in a non-SDN-capable node, SENAtoR provides loss rates lower than the all-

OSPF case, since the SDN controller can provide most appropriate routes. As a conclusion, SENAtoR provides energy savings while being compatible with current network infrastructures. As a future work, we can improve SENAtoR with a deeper study of the traffic network variations to provide the most adapted thresholds for the spikes and link failure detections.

Bibliography

[VVB14] Stefano Vissicchio, Laurent Vanbever, and Olivier Bonaventure. "Opportunities and research challenges of hybrid software defined networks". In: *ACM SIGCOMM Computer Communication Review* 44.2 (Apr. 2014), pp. 70–75. DOI: 10.1145/2602204.2602216 (cit. on pp. 138, 140).

[Rou+02] Matthew Roughan, Albert Greenberg, Charles Kalmanek, Michael Rumsewicz, Jennifer Yates, and Yin Zhang. "Experience in measuring backbone traffic variability: Models, metrics, measurements and meaning". In: *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM. 2002, pp. 91–92 (cit. on pp. 138, 141).

[Chu+15] Cing-Yu Chu, Kang Xi, Min Luo, and H Jonathan Chao. "Congestion-aware single link failure recovery in hybrid SDN networks". In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE. 2015, pp. 1086–1094 (cit. on pp. 138, 141).

[AKL13] Sugam Agarwal, Murali Kodialam, and TV Lakshman. "Traffic engineering in software defined networks". In: *INFOCOM, 2013 Proceedings IEEE*. IEEE. 2013, pp. 2211–2219 (cit. on pp. 140, 142).

[KW15] Dave Katz and David Ward. *Bidirectional Forwarding Detection (BFD)*. RFC 5880. Oct. 2015 (cit. on p. 140).

[Sha+16] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. "In-band control, queuing, and failure recovery functionalities for OpenFlow". In: *IEEE Network* 30.1 (2016) (cit. on p. 141).

[Ian+01] Gianluca Iannaccone, Christophe Diot, Ian Graham, and Nick McKeown. "Monitoring very high speed links". In: *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. ACM. 2001, pp. 267–271 (cit. on p. 141).

[LCD04] Anukool Lakhina, Mark Crovella, and Christiphe Diot. “Characterization of network-wide anomalies in traffic flows”. In: *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement.* ACM. 2004, pp. 201–206 (cit. on p. 141).

[LCD05] Anukool Lakhina, Mark Crovella, and Christophe Diot. “Mining anomalies using traffic feature distributions”. In: *ACM SIGCOMM Computer Communication Review.* Vol. 35. 4. ACM. 2005, pp. 217–228 (cit. on p. 141).

[Ari+03] Ismail Ari, Bo Hong, Ethan L Miller, Scott A Brandt, and Darrell DE Long. “Managing flash crowds on the Internet”. In: *IEEE/ACM MASCOTS 2003.* 2003 (cit. on p. 141).

[Wan+13] Ye Wang, Yueping Zhang, Vishal Singh, Cristian Lumezanu, and Guofei Jiang. “NetFuse: Short-circuiting traffic surges in the cloud”. In: *IEEE ICC.* 2013 (cit. on p. 141).

[Gir+03] Frédéric Giroire, Antonio Nucci, Nina Taft, and Christophe Diot. “Increasing the robustness of IP backbones in the absence of optical level protection”. In: *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies.* Vol. 1. IEEE. 2003, pp. 1–11 (cit. on p. 141).

[B C04] B. Claise. *RFC 3954 - Cisco Systems NetFlow Services Export Version 9.* 2004 (cit. on p. 142).

[Idz+16] Filip Idzikowski, Luca Chiaraviglio, Antonio Cianfrani, Jorge López Vizcaı’no, Marco Polverini, and Yabin Ye. “A Survey on Energy-Aware Design and Operation of Core Networks”. In: *IEEE Communications Surveys & Tutorials* 18.2 (2016) (cit. on pp. 142, 152).

[Gir+10] Frederic Giroire, Dorian Mazauric, Joanna Moulierac, and Brice Onfroy. “Minimizing Routing Energy Consumption: From Theoretical to Practical Results”. In: *2010 IEEE/ACM International Conference on Green Computing and Communications and International Conference on Cyber, Physical and Social Computing* (Dec. 2010) (cit. on p. 146).

[Car+16] Radu Carpa, Marcos Dias de Assuncao, Olivier Gluck, Laurent Lefevre, and Jean-Christophe Mignot. "Responsive algorithms for handling load surges and switching links on in green networks". In: *2016 IEEE International Conference on Communications (ICC)* (May 2016). DOI: 10.1109/icc.2016.7511557. URL: http://dx.doi.org/10.1109/ICC.2016.7511557 (cit. on p. 147).

[Vu+14] TH Vu, VC Luc, NT Quan, T Thanh, NH Thanh, and PN Nam. "Sleep Mode and Wakeup Method for OpenFlow Switches". In: *Journal of Low Power Electronics* 10.3 (2014), pp. 347–353 (cit. on pp. 150, 151).

[Cis14] Cisco. Aug. 2014. URL: http://www.cisco.com/c/en/us/products/collateral/routers/1900-series-integrated-services-routers-isr/data_sheet_c78_556319.html (cit. on p. 150).

[Con+14] Paul T Congdon, Prasant Mohapatra, Matthew Farrens, and Venkatesh Akella. "Simultaneously reducing latency and power consumption in openflow switches". In: *IEEE/ACM Transactions on Networking (TON)* (2014) (cit. on p. 151).

[Cho+07] Baek-Young Choi, Sue Moon, Zhi-Li Zhang, Konstantina Papagiannaki, and Christophe Diot. "Analysis of point-to-point packet delay in an operational network". In: *Computer networks* 51.13 (2007), pp. 3812–3827 (cit. on p. 157).

[Faw+04] Wissam Fawaz, Belkacem Daheb, Olivier Audouin, M Du-Pond, and Guy Pujolle. "Service level agreement and provisioning in optical networks". In: *IEEE Communications Magazine* 42.1 (2004) (cit. on p. 157).

[Min] Team Mininet. *Mininet*. http://mininet.org/ (cit. on p. 157).

[Flo] Floodlight. *OpenDaylight*. URL: https://www.opendaylight.org/ (cit. on p. 157).

[Qua] Quagga. *Quagga Routing Software Suite*. URL: http://www.nongnu.org/quagga/ (cit. on p. 157).

[Pfa+15] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. “The Design and Implementation of Open vSwitch”. In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 117–130 (cit. on p. 157).

Part III

Service Function Chaining

Network Function Virtualization

Context

In the last two parts, we considered the constraints of Software Defined Network networks for the implementation of Energy Aware Routing (EAR). We now shift our focus on the Network Function Virtualization and its impact on Energy Aware Routing.

Network Function Virtualization proposes to virtualize network functions and services that once ran on middleboxes. Each function was implement by a specific middlebox; it exists as many different specific hardware as network functions. While Software Defined Network (SDN) aims at decoupling control and data plane, the Network Function Virtualization (NFV) paradigm provides a way to decouple software from hardware. Network operators are no longer bound to the development of new hardware but only to new software which exhibits shorter development timelines. General servers can now run the network functions hosted on Virtual Machines (VMs), leaving more breathing room for operators to manage and develop new services. Indeed, functions can be moved on the network depending on the traffic, by migrating the hosting VMs.

Moreover, network services are usually composed of an ordered chain of functions to apply to the requests. They are called Service Function Chains. Removing the need for middleboxes greatly help the construction of new chains. Instead of physically deploying new middleboxes into the network, network operators can rely on an NFV orchestrator to migrate the VMs depending on the requests executed on the network. As virtual functions require various resources (CPU cores, memory, bandwidth), NFV Management & Orchestration (NFV MANO) represents an important part of the infrastructure.

The problem we consider in this part is Service Function Chain (SFC) provisioning, which comprises determining which Virtual Network Functions (VNFs) are placed on which nodes and how VNF instances are assigned to SFCs. The placement and assignment affect the traffic routing from the SFC through the servers or virtual machines' network. SFC provisioning must offer the best compromise between the conflicting goals of minimizing infrastructure and bandwidth resources, end-to-end latency (i.e., Service Level Agreements (SLAs)) and reducing the replica numbers of VNFs and SFCs. The location of the VNFs affects the end-to-end latency incurred by the packets traversing a particular SFC. Also, a poor placement will cause the flow to traverse the same path-segments back and forth inside the network, increasing the network delay and consuming more bandwidth.

Contributions and plan

We first study the problem of Service Function Chain Provisioning without energy efficiency in Chapter 7. We propose the design of a *mathematical model with a decomposition scheme* that allows a *scalable exact solution scheme*, while nearly all previous algorithms of the literature (see next section) are either heuristics or non-scalable exact algorithms. To the best of our knowledge, we are the first to propose an exact model, which scales well with the number of nodes and requests. Our model can solve within a few minutes instances with almost 10.000 different demands. The model then allows the investigation of the *best compromise between the numbers of VNF replicas, the best placement of VNFs, the resource requirements and the end-to-end delays.*

Then, in Chapter 8, we extend our model to the *Energy Efficient Service Function Chaining* problem. We study the impact of using Virtual Network Functions instead of middleboxes when deploying SDN green policies. We also propose GreenChains, a Integer Linear Program (ILP)-based heuristic to solve the Energy-Efficient Service Function Chaining (EE-SFC) problem. We show that using NFV allows to save 4 to 12% more energy than the hardware scenario (SDN+middleboxes).

We present in the next section works on SFC problem for exact and partial formulations.

Related work

Following the NFV initiative in 2012 [Gem+16], several surveys are now available on NFV, see, e.g., [LC15; HB16; Mij+16] where the various NFV challenges are discussed. Focusing on the SFC provisioning problem, various works proposed partial and exact mathematical formulations, considering several objective functions.

Partial mathematical formulations. In Martini *et al.* [Mar+15] and Riggio *et al.* [Rig+15], the authors only solve the placement and routing for each request independently. A heuristic based on an ILP is proposed in Gupta *et al.* [Gup+15]. The authors only consider the k-shortest paths for every request in the network and a simplified node capacity constraint, for which only one function per node can be deployed.

Exact mathematical formulations. Luizelli et al. [Lui+15] provide an exact model minimizing the number of instances of functions in the network. However, they consider only a couple of tens of requests. Savi et al. [STV15] propose an exact formulation in which the number of VNF nodes is minimized. Their model takes into account additional costs inherent to a multi-core environment. However, they only provide results on a small network. Bari et al. [Bar+16], the authors consider the Operational Expenditure (OpEx) for a daily traffic scenario as their objective function. Mohammadkhan et al. [Moh+15] propose an exact model along with heuristics aiming at minimizing the maximum usage of CPU and links. The scope of the experiments is limited to the case in which the number of cores per service is limited to one.

The ILPs proposed in the works mentioned above do not scale for larger networks. To the best of our knowledge, using column generation, our work is the first to optimally solve the problem of SFC placement in a network with 50 nodes and for all-to-all demand scenarios (10 000 requests). Even tough some works [STV15; Lui+15] minimize the number of function replications in the network, we are also the first to consider constraints on the number of function replications throughout the whole network while minimizing bandwidth usage.

Energy Efficient Network Function Virtualization Only a few works investigate the potential of network virtualization for energy efficiency. In [Bol+14], Bolla et al. present an extension of an open source software framework, the Distributed Router Open Platform (DROP), to enable a novel distributed paradigm for NFV. DROP includes sophisticated power management mechanisms, which are exposed by means of the Green Abstraction

Layer. In [Mij15], Mijumbi estimates the energy savings that could result from the three main NFV use cases-Virtualized Evolved Packet Core, Virtualized Customer Premises Equipment and Virtualized Radio Access Network. However, both papers do not consider the constraints of service chains. The work of Soualah et al. [Sou+17] is, to the best of our knowledge, the closest to ours. They explore a scenario with service function chains and dynamic requests arrival. However, they only reduce the energy consumption of the physical machines and not of the network.

Chapter 7
Optimal Network Service Chain Provisioning

Contents

7.1 Introduction

In this chapter, we study the *Service Function Chaining* Provisioning problem and propose the first scalable exact model. It can solve instances with 10 000 requests on network of 50 nodes.

The chapter is organized as follows. A detailed and formal statement of the Service Function Chain Provisioning problem is given in Section 7.1.1 followed by the description of the layered graph model used for all of our ILP models. We describe our optimization models in Section 7.2, first a classical ILP model, and then a decomposition ILP one. Solution process and algorithms are depicted in Section 7.3. We then present extensions of the models to handle the case in which the number of VNF replicas is limited in Section 7.4. Numerical results are described in Section 7.5.

7.1.1 Problem Statement

We consider an SDN network that is represented by a graph $G = (V, L)$ where V denotes the set of nodes and L the set of links. Each service chain c is defined as an ordered sequence of Virtual Network Functions (VNFs), with some functions possibly repeated. We denote by n_c the number of functions in c, i.e., the length of the sequence. Let C be the set of all service chains.

Demand is defined by a set of requests, where each request is characterized by a source v_s, a destination v_d, a service chain c and a bandwidth requirement, i.e., D^c_{sd} units. Let $\mathcal{SD}$ the set of node pairs with some demand.

Let F be the set of virtual network functions, indexed by f. The number of cores required by function f in any chain is equal to Δ_f per unit of bandwidth, i.e., $D^c_{sd} \times \Delta_{f^c_i}$ cores for request (v_s, v_d, c, D^c_{sd}) at the node hosting function f^c_i, the ith function of chain c.

We assume that only a subset of nodes $V^{\text{VNF}} \subseteq V$ can host VNFs. Indeed, deployment of VNFs can be made on general purpose servers or standard IT platforms like high-performance switches, service, and storage, see, e.g., [LC15] for more details. Running a VNF requires a certain amount of resources, e.g., CPU, memory, disk, while the number of required resources usually depends on the volume of traffic that passes through it. Consequently, each node $v \in V^{\text{VNF}}$ has a given core capacity $\textsc{cc}_v$. Similarly, each link ℓ of the network has a transport capacity of $\textsc{cap}_\ell$. Notations are summarized in Table 7.1.

Service Function Chain Provisioning Problem. For a given demand

$G = (V, L)$	optical (grid) network
$V^{\text{VNF}} \subseteq V$	subset of nodes which are enabled to host virtualization functions
$\mathcal{SD}$	Set of node pairs with some demand
D^c_{sd}	bandwidth demand from s to d for chain c
Δ_f	# required cores per bandwidth unit for function f
CAP_ℓ	transport capacity (bandwidth) of link ℓ
CAP_v	core capacity of node v
n^c	length (i.e., number of functions) of the chain c
f^i_c	the i^{th} function in chain c
C_{sd}	set of chains between v_s and v_d

Table 7.1: Notations.

D, provision the service chains, while minimizing the overall bandwidth requirement subject to the core and transport capacities, and possibly to a limit on the number of NFV nodes and SFC replicas.

When provisioning SFCs, each chain is assigned a path in which functions of c are encountered in the same order as in c, with some functions possibly located at the same node.

7.1.2 Layered Graph

Following a similar idea as in [DW16], we use a layered graph G^{L} that is defined as follows. The initial network graph G is transformed into a *layered graph* G^{L} by adding $\max\limits_{c \in C} n_c$ layers to the graph (counting G as the base layer, i.e., layer 0, and where each layer is an exact copy of the original graph. For every node $v \in V$, let v^i denote the corresponding node in the ith layer ($i = 1, \ldots, n_c$). Every $(i-1, i)$ layer pair is connected vertically by links from v^{i-1} to v^i, see Figure 7.1 for an illustration.

Finding a path and a chain placement for a request (u_s, u_d, c, D^c_{sd}) consists in finding a path from node v_s on the first layer to node v_d on the $n_c + 1$th layer. Indeed, each layer represents the progression of the chain, e.g., being on the second layer means that the first function of the chain is already executed. The placement of the node is given by the link used to switch

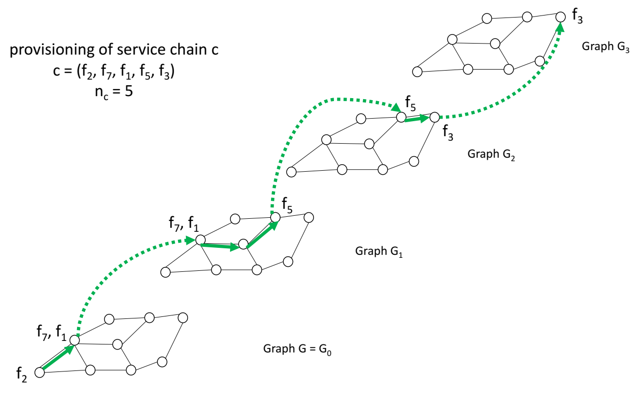


Figure 7.1: Layered Graph

between layers.

All Integer Linear Programming (ILP) models presented in the layered graph.

7.2 Optimization Models

We first present a classical Integer Linear Program (ILP) model, called NFV_ILP, in Section 7.2.1 and then a reformulation with a Column Generation (CG) decomposition model, called NFV_CG, in Section 7.2.2.

7.2.1 Model NFV_ILP

Model NFV_ILP is an Integer Linear Program based on the *layered graph* described in Section 7.1.2.

Variables. Model NFV_ILP uses two sets of variables. The first set is a set of 0-1 variables $\varphi_\ell^{sd,c,i}$ defined as follows. Variable $\varphi_\ell^{sd,c,i} = 1$ if (u_s, u_d, c, D_{sd}^c) is provisioned on link ℓ at layer i, 0 otherwise. The second set corresponds to 0-1 variables $\alpha_v^{sd,c,i}$ such that $\alpha_v^{sd,c,i} = 1$ if f_c^i is installed on node v. Note that we will use the convention that if $v \notin V^{\text{VNF}}$, $\alpha_v^{sd,c,i} = 0$.

Objective. Minimization of the bandwidth requirements

$$\min \qquad \sum_{(v_s,v_d)\in \mathcal{SD}} \sum_{c \in C_{sd}} D_{sd}^c \sum_{\ell \in L} \sum_{i=0}^{n^c} \varphi_\ell^{sd,c,i} \tag{7.1}$$

Constraints. There are three sets of constraints.
Flow constraints. They translate the requirement of a path from source to destination going through the locations of the functions of the service chain requested by each node pair demand. Only the source node on the first layer and the destination node on the last layer can have a positive outgoing and incoming flow respectively.

$$\sum_{\ell \in \omega^+(u)} \varphi_\ell^{sd,c,i} - \sum_{\ell \in \omega^-(u)} \varphi_\ell^{sd,c,i} + \alpha_v^{sd,c,i} - \alpha_v^{sd,c,i-1} = 0 \quad \forall v \in V, (v_s, v_d) \in \mathcal{SD}, c \in C_{sd}, 0 < i < n^c \tag{7.2}$$

$$\sum_{\ell \in \omega^+(v)} \varphi_\ell^{sd,c,0} - \sum_{\ell \in \omega^-(v)} \varphi_\ell^{sd,c,0} + \alpha_v^{sd,c,0} = \begin{cases} 1 \text{ if } v = v_s \\ 0 \text{ else} \end{cases} \quad \forall (v_s, v_d) \in \mathcal{SD}, v \in V, c \in C_{sd} \tag{7.3}$$

$$\sum_{\ell \in \omega^+(v)} \varphi_\ell^{sd,c,n_c} - \sum_{\ell \in \omega^-(v)} \varphi_\ell^{sd,c,n_c} - \alpha_v^{sd,c,n_c} = \begin{cases} -1 \text{ if } v = v_d \\ 0 \text{ else} \end{cases} \quad \forall (v_s, v_d) \in \mathcal{SD}, v \in V, c \in C_{sd}. \tag{7.4}$$

Link capacity. Usage of a given link ℓ is distributed over the different layers and cannot exceed the link transport capacity CAP_ℓ.

$$\sum_{(v_s,v_d) \in \mathcal{SD}} \sum_{c \in C_{sd}} D_{sd}^c \sum_{i=0}^{n^c} \varphi_\ell^{sd,c,i} \leq \text{CAP}_\ell \qquad \forall \ell \in L. \tag{7.5}$$

Node capacity. The placement of a function in node v is described by the usage of cross-layer link (v^{i-1}, v^i). The usage of a node is determined by the set of cross-layer links that are used to switch from one layer to the next, and should not exceed its core capacity.

$$\sum_{(v_s,v_d) \in \mathcal{SD}} \sum_{c \in C_{sd}} \sum_{i=0}^{n^c} \Delta_{f_i^c} D_{sd}^c \alpha_v^{sd,c,i} \leq \text{CC}_v \quad \forall v \in V^{\text{VNF}}. \tag{7.6}$$

7.2.2 Model NFV_CG

As we will see in Section 7.5.3, the NFV_ILP model does not scale well for medium to large networks. We thus propose a Column Generation model, called NFV_CG model. It relies on the concept of configurations, i.e., a potential path provisioning or service path for a given request.

More formally, a configuration, i.e., a *Service Path* for a request (u_s, u_d, c, D^c_{sd}) is composed of: *(i)* a network path, i.e., an ordered set of nodes from the source to the destination node of the request, and *(ii)* a set of locations for the VNFs in the SFC request. Each *Service Path* is thus specific to a given request and its SFC. However, a given SFC may be shared with several requests.

We use the following additional parameters.

Parameters

- $p \in P^c_{sd}$: a *Service Path* from v_s to v_d, where P^c_{sd} denotes the set of paths from v_s to v_d for chain c. A service path is composed of a path in the network and a set of pairs (v, f). Each pair (v, f) expresses that function f is executed on node v.
- $\text{LENGTH}(p) \in \mathbb{N}$: it denotes the number of links of path p.
- $a^p_{iv} \in \{0, 1\}$, where $a^p_{iv} = 1$ if f^c_i is installed on node v for *Service Path* $p \in P^c_{sd}$.
- $\delta^p_\ell \in \mathbb{N}$ denotes the number of occurrences of link ℓ in path p.

Variables

- $y^{sd,c}_p \geq 0$, where $y^{sd,c}_p = 1$ if request (u_s, u_d, c, D^c_{sd}) is forwarded through service path p, 0 otherwise.

Objective

$$\min \sum_{(v_s,v_d)\in\mathcal{SD}} \sum_{c\in C_{sd}} \sum_{p\in P^c_{sd}} D^c_{sd}\, \text{LENGTH}(p)\, y^{sd,c}_p \tag{7.7}$$

As for Model NFV_ILP, the objective is to minimize the amount of bandwidth used in the SDN network. For a path, this amount is its length, i.e., the number of hops, multiplied by the bandwidth requirement of the request. The set of constraints can then be expressed as follows.

Constraints
Exactly one path per demand and per chain:

$$\sum_{p \in P^c_{sd}} y_p^{sd,c} = 1 \qquad \forall c \in C_{sd}, (v_s, v_d) \in \mathcal{SD} \tag{7.8}$$

Link capacity: for all $\ell \in L$,

$$\sum_{(v_s,v_d) \in \mathcal{SD}} \sum_{c \in C_{sd}} \sum_{p \in P^c_{sd}} (D^c_{sd}\, \delta^p_\ell) \times y_p^{sd,c} \leq \text{CAP}_\ell \tag{7.9}$$

Node capacity: for all $v \in V^{\text{VNF}}$,

$$\sum_{(v_s,v_d) \in \mathcal{SD}} \sum_{c \in C_{sd}} \sum_{i=0}^{n_c} \sum_{p \in P^c_{sd}} (\Delta_{f^c_i} D^c_{sd} \;\; a^p_{iv}) \quad \times \quad y_p^{sd,c} \quad \leq \quad \text{CC}_v \tag{7.10}$$

7.3 Solution Scheme

Model NFV_ILP can be easily solved by an ILP solver such as CPLEX. Model NFV_CG requires more attention as, at first look, it has an exponential number of variables. Indeed, its linear relaxation can be solved *exactly* using column generation ([Chv83]), using a limited number of configurations, i.e., variables. An integer solution is next derived, together with its accuracy. Details are given below.

7.3.1 Pricing Problem

The role of the Pricing Problem is to generate a valid *Service Path* for a given request (u_s, u_d, c, D^c_{sd}). This problem corresponds to an NP-complete problem, the Shortest Weight-constrained path problem [GJ79]. Once again, the formulation relies on the layer graph (G^{L}) introduced in Section 7.1.2. The so-called reduced cost defines its objective.

- $u^{(j)}$ represents the vector of dual variables of constraints (j) in the RMP. Note that these values are given as input to the pricing problem in the column generation solution process.

Variables:

- $\alpha_v^i \in \{0,1\}$, where $a_v^i = 1$ if f_i^c is installed on node v.
- $\varphi_\ell^i \in \{0,1\}$, where $\varphi_\ell^i = 1$ if the flow forwarded on link ℓ on layer i, i.e., links in each layer in graph G^{L}.

The service path generator (pricing problem) is written for each request (u_s, u_d, c, D_{sd}^c).

$$\begin{aligned}\min \quad & \sum_{\ell \in L} \sum_{i=0}^{n_c} \varphi_\ell^i \times (D_{sd}^c - D_{sd}^c u_\ell^{(7.9)}) \\ & - u_{sd}^{(7.8)} \\ & + D_{sd}^c \sum_{v \in V} u_v^{(7.10)} \sum_{i=0}^{n_c} \Delta_{f_i^c} \alpha_v^i \end{aligned} \tag{7.11}$$

Flow conservation: they correspond to flow constraints (i.e., route) from the ith function to the $(i+1)$th function of the service chain associated with the $v_s \rightsquigarrow v_d$ request for which the pricing problem is solved (constraints (7.12)), and then flow constraints from the source node to the location of the first function of the service chain (constraints (7.13)), and similarly from the location of the last function of the service chain to the destination node (constraints (7.14)). Note that $a_v^i = 0$ for all nodes that are not VNF capable. Observe that the next set of constraints take care of the possibility that several VNFs can be located on the same node, including on the source or destination nodes.

$$\sum_{\ell \in \omega^+(v)} \varphi_\ell^i - \sum_{\ell \in \omega^-(v)} \varphi_\ell^i + \alpha_v^i - \alpha_v^{i-1} = 0 \qquad v \in V, 0 < i < n_c \tag{7.12}$$

$$\sum_{\ell \in \omega^+(v)} \varphi_\ell^0 - \sum_{\ell \in \omega^-(v)} \varphi_\ell^0 + \alpha_v^0 = \begin{cases} 1 \text{ if } v = v_s \\ 0 \text{ else} \end{cases} \qquad v \in V \tag{7.13}$$

$$\sum_{\ell \in \omega^+(v)} \varphi_\ell^{n_c} - \sum_{\ell \in \omega^-(v)} \varphi_\ell^{n_c} - \alpha_v^{n_c} = \begin{cases} -1 \text{ if } v = v_d \\ 0 \text{ else} \end{cases} \qquad v \in V \tag{7.14}$$

Lastly, we need to make sure that the produced paths are respecting the link and node capacities of the original network.
Link capacity. For $\ell \in L$,

$$D^c_{sd} \sum_{i=0}^{n^c} \varphi^i_\ell \leq \text{CAP}_\ell. \tag{7.15}$$

Node capacity. For $v \in V^{\text{VNF}}$,

$$\sum_{i=0}^{n^c} (\Delta_{f^c_i} D^c_{sd}) \times \alpha^i_v \leq \text{CC}_v. \tag{7.16}$$

Note that, since the resolutions of the pricing problems for all requests are independent, we launch in parallel the corresponding ILPs to improve the execution time.

Speeding the solution of the pricing problem with the Bellman-Ford algorithm. As discussed above, the pricing problem can be solved using an ILP. However, it can be solved more quickly using the following observation. A solution to the Pricing Problem is equivalent to a constrained shortest path in the layered graph. The weight $w_{i\ell}$ of the link ℓ at layer i is equal to

$$w_{i\ell} = D^c_{sd} - u^{(7.9)}_\ell D^c_{sd}$$

and the weight of the link going from layer i to $i+1$ at node v is equal to

$$w_{iv} = -D^c_{sd} u^{(7.10)}_v \Delta_{f^c_i}$$

If we now ignore the capacity constraints, the pricing problem becomes a simple shortest path problem. Since the dual values of the constraints (7.9) and (7.10) can be negative, we use the Bellman-Ford algorithm to solve it. As we did not consider the capacity constraints, we may obtain invalid paths. However, they are easily discarded by checking if they use more capacity than available. In this case, we then launch the ILP. This method allows us to speed up the column generation process, as shown in the next section.

7.4 Limiting Function Replicas

We now consider the case in which an operator is constrained by the number of replicas of each virtualized function. Such scenario may occur when the operator has a limited budget for its license cost or when the licenses provided for a function only allow a certain number of replicas in the network.

Having a limited number of preset NFV capable nodes, as done in Section 7.5.4, is a simple addition to the models. However, limiting the number of function replicas, which may be placed on any nodes, increases greatly the complexity of the models and the execution time. Indeed, this introduces a large number of binary variables in the model.

In section 7.4.1, we provide again two models, the first one being an ILP and the second one using column generation. As the complexity of the model is high, we introduce in Section 7.4.2 a heuristic algorithm to solve the problem for large networks. The heuristic first finds a good placement of the network functions by solving a variant of a capacitated k-mean clustering problem (or facility location problem) using an ILP, and then uses this placement in the CG model to find a solution of the general problem.

7.4.1 Models

The models are extensions of NFV_ILP and NFV_CG, as described below, and are called NFV_ILP$^+$ and NFV_CG$^+$, respectively. In our two models, we introduce a new set of variables, b_{vf}, indicating if the function f is installed on node v. We can now limit the total number of functions on the network using the following constraints in both models.

$$\sum_{v \in V^{\text{VNF}}} b_{vf} \leq L_f \qquad f \in \mathcal{F} \tag{7.17}$$

where L_f represents the maximum number of replicas for the function f.

In the NFV_ILP$^+$ model, we then limit the placement of functions for every request with the following constraints

$$a_v^{sd,c,i} \leq b_{vf_i^c} \quad v \in V^{\text{VNF}}, (v_s, v_d) \in \mathcal{SD}, c \in C_{sd}, 0 < i < n_c \tag{7.18}$$

For NFV_CG$^+$, a *Service Path* can only be used if all the execution locations of its functions are active. Another way to formulate this is to say that a location v for a function f must be active, i.e, $b_{vf} = 1$, if any path using this location is active. This can be expressed using the following constraints

$$\sum_{c \in C_{sd}} \sum_{(v_s,v_d) \in \mathcal{SD}} \sum_{p \in P_{sd}^c} \beta_{vf}^p y_p^{sd,c} \leq M b_{vf} \quad v \in V^{\text{VNF}}, f \in \mathcal{F} \tag{7.19}$$

where $M \geq |\mathcal{SD}| \times |C|$ and $\beta^p_{vf} = 1$ if function f need to be installed on node v for path p.

We can remove the big M at the cost of a higher number of constraints. Indeed, since $\sum\limits_{p \in P^c_{sd}} \beta^p_{fv} y^{sd,c}_p \leq \sum\limits_{p \in P^c_{sd}} y^{sd,c}_p = 1$, we can replace the constraints (7.19) at the cost of a higher number of constraints by

$$\sum_{p \in P^c_{sd}} \beta^p_{vf} y^{sd,c}_p \leq b_{vf} \qquad \forall v \in V^{\text{VNF}}, f \in \mathcal{F}, (v_s, v_d) \in \mathcal{SD}, c \in C_{sd}, \quad (7.20)$$

Pricing Problem

We introduce the set of variables β_{vf} into the Pricing Problem to keep track of the function placement. Note that chains can contain multiple occurrences of the same function and that this is taken into account with variables $\alpha_{vf^c_i}$, allowing a function to arise several times (e.g., firewall application) in a service chain to be used in different locations, i.e., potentially a different one for each occurrence.

The objective function of the Pricing Problem for a request (u_s, u_d, c, D^c_{sd}) for NFV_CG becomes:

$$\min \quad \sum_{\ell \in L} \sum_{i=0}^{n_c} \varphi^i_\ell \times \left(D^c_{sd} - D^c_{sd} u^{(7.9)}_\ell \right) - u^{(7.8)}_{sd} + D^c_{sd} \sum_{v \in V^{\text{VNF}}} u^{(7.10)}_v \sum_{i=0}^{n_c} \Delta_{f^c_i} \alpha^i_v + \sum_{v \in V} \sum_{f \in \mathcal{F}} \beta_{vf} u^{(7.20)}_{sd,c,v,f}.$$

Like the RMP formulation, we add the following two sets of constraints for the number of replicas in the network. First, we limit the number of replicas in the network.

$$\sum_{v \in V^{\text{VNF}}} \beta_{vf} \leq L_f \qquad f \in \mathcal{F} \qquad (7.21)$$

Then, we limit the execution of function on node where the function is installed.

$$\beta_{vf^c_i} \geq \alpha^i_v \qquad v \in V^{\text{VNF}}, 0 \leq i \leq n^c \qquad (7.22)$$

Unlike the Pricing Problem of NFV_CG, it is not possible to find a solution using the Bellman Ford algorithm and we thus use CPLEX to solve it.

7.4.2 Heuristics (NFV_Algo$^+$)

As shown in Section 7.5, NFV_CG can find quickly near-optimal solutions to the SFC placement problem, when the number of function replica is not limited. However, NFV_CG$^+$ has an execution time a lot longer and is not able to solve a large network such as `germany50`. Thus, we propose a heuristic solution using NFV_CG, called NFV_Algo$^+$. First, we consider the placement of functions in the network in the same fashion as a facility location problem, as explained below. Then, we use NFV_CG to solve the routing of the requests once the placement has been chosen.

Location problem The core of the method is a variant of a k-mean clustering problem. For each function, we have to choose $k = L_f$ possible locations. As our goal is to minimize the network bandwidth, a good solution is to choose the locations that minimize the request path lengths, that is the distances between the sources and destinations of the requests and the location associated with the requests. Our solution is not directly a k-mean, but a generalization, as we have capacity constraints of nodes to satisfy. We thus model the placement problem as an ILP.

We search for each function f exactly L_f nodes that can host it. Since each node has a limited capacity, we also need to consider on which node the function of a specific request must be executed. We thus use the two following sets of variables.

- $x_{vi}^{sd,c} \in \{0,1\}$, where $x_{vi}^{sd,c} = 1$ if the ith function of the chain c for the demand (v_s, v_d) is installed on the node v.
- $b_{vf} \in \{0,1\}$, where $b_{vf} = 1$ if the function f is installed on v.

The problem is formulated as follows.

$$\min \sum_{v \in V^{\text{VNF}}} \sum_{(v_s,v_d) \in \mathcal{SD}} \sum_{c \in C_{sd}} \sum_{i \in \{0 \dots n_c\}: f_i^c = f} (d(v_s, v) + d(v_d, v)) \times x_{vi}^{sd,c} \quad (7.23)$$

We want to minimize the distance of the request to the location of the function (7.23). The distance is given by the sum of the distance to the source, $d(v_s, v)$, and the distance to the destination, $d(v_d, v)$.

$$\sum_{v \in V^{\text{VNF}}} x_{vi}^{sd,c} = 1 \qquad (v_s, v_d) \in \mathcal{SD}, c \in C_{sd}, \; 0 \leq i \leq n_c, f = f_i^c \tag{7.24}$$

$$b_{vf_i^c} \geq x_{vi}^{sd,c} \qquad (v_s, v_d) \in \mathcal{SD}, c \in C_{sd}, 0 \leq i \leq n_c, \; v \in V^{\text{VNF}}, f = f_i^c \tag{7.25}$$

The functions of a request need to be executed on exactly one node (7.24), and they can be executed only on nodes on which the function is installed (7.25).

$$\sum_{(v_s,v_d) \in \mathcal{SD}} \sum_{c \in C_{sd}} \sum_{i \in \{0 \ldots n_c\}: f_i^c = f} (\Delta_{f_i^c} D_{sd}^c) \times x_{vi}^{sd,c} \leq \text{CAP}_v \qquad v \in V^{\text{VNF}} \tag{7.26}$$

$$\sum_{v \in V^{\text{VNF}}} b_{vf} \leq L_f \tag{7.27}$$

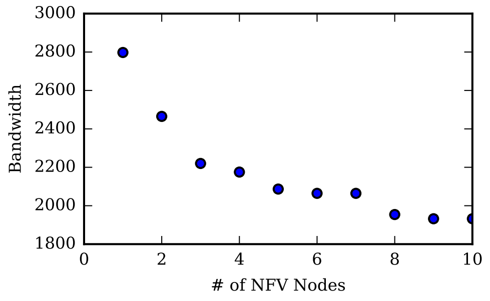
Finally, we have the node capacity (7.26) and the maximum license constraints (7.27).

7.5 Numerical Results

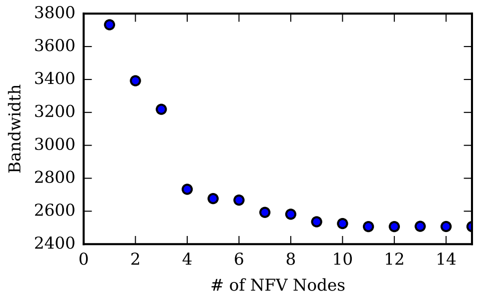
In this section, we report the numerical results. First, we describe the datasets we used (Section 7.5.1). Then, we present the performance of NFV_-CG in Section 7.5.2. Next, in Section 7.5.3, we compare the performance of the two models described in Section 7.2 and look at the compromise between the number of VNF nodes and the bandwidth requirements in Section 7.5.4. Finally, we compare the different solutions we propose for the case with a limited number of possible VNF replicas and study the impact on bandwidth requirement in Section 7.5.5.

7.5.1 Data Sets

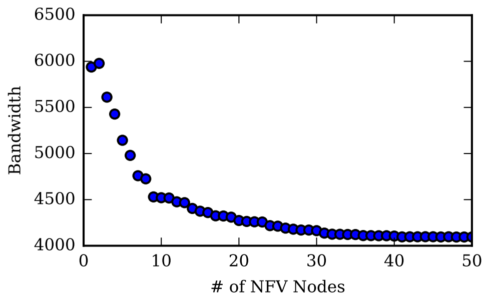
To emulate a realistic traffic, we used the data in [Cis15] in conjunction with the four chains presented in Table 7.2 as in [STV15]. Each SFC is composed



(a) `internet2`

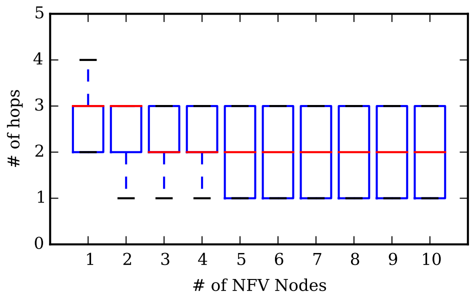


(b) `atlanta`

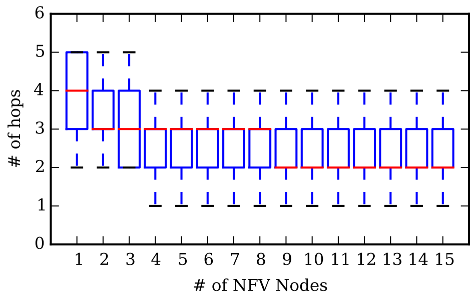


(c) `germany50`

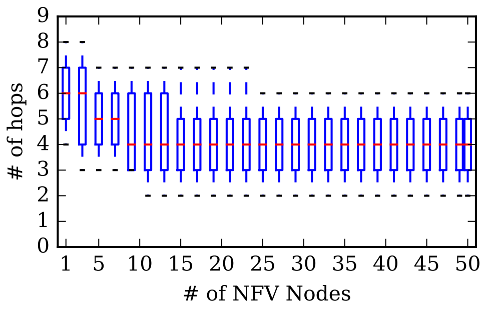
Figure 7.2: Bandwidth vs. number of VNF nodes with a 1TB offered load.



(a) `internet2`



(b) `atlanta`



(c) `germany50`

Figure 7.3: Distribution of the number of hops for each demand vs. number of VNF nodes with a 1TB offered load. Boxes are defined by the first and third quartiles. Ends of the whiskers correspond to the first and ninth deciles.

of a sequence of virtual network functions and requires a specific amount of bandwidth. We use the distribution of traffic from [Cis15] to know the number of requests of each service type. For example, a 1TB network load is composed of 699GB of Video Streaming. This amount of traffic correspond to an equivalent of $\frac{699}{4 \times 10^{-3}}$ requests. We then choose at random the source and destination for each request and then aggregate the resulting set of requests with respect to their source and destination nodes. Overall, we have a total of $4 \times n(n-1)$ demands (each type of chains for every node pair).

Service Chain	Chained VNFs	rate	% traffic
Web Service	NAT-FW-TM-WOC-IDPS	100 kbps	18.2%
VoIP	NAT-FW-TM-FW-NAT	64 kbps	11.8%
Video Streaming	NAT-FW-TM-VOC-IDPS	4 Mbps	69.9%
Online Gaming	NAT-FW-VOC-WOC-IDPS	50 kbps	0.1%

Table 7.2: Service chain requirements [STV15]

When choosing the set of nodes which can host VNFs, we select the nodes based on their betweenness centrality, which is the number of paths going through the node, when considering the shortest paths between all pairs of nodes. Betweenness centrality is a good indicator of the importance of a node in the network. Programs were tested on three different networks, whose characteristics are described in Table 7.3.

7.5.2 Performance of Model NFV_CG

Table 7.4 summarizes the performance of Model NFV_CG. We present results for the three different topologies for a selected number of VNF nodes, around the half of the size of the networks. For each instance, we simulate an overall traffic of 1 Tbps.

In the last three columns, we give the optimal value of the linear relaxation ($z^{\text{LP}}_{\text{LP}}$), the value of the ILP solution ($\tilde{z}_{\text{ILP}}$) and the accuracy of the ILP solution ε. In most instances, $\varepsilon = 0$, meaning that we obtain the optimal ILP solution For the cases where $\varepsilon > 0$, its value remains very small, meaning that $\tilde{z}_{\text{ILP}}$ is very close to the optimal ILP value.

Lastly, we observe that the number of generated columns is relatively low to reach very accurate ILP solutions, taking into account that we need to select one column per request, i.e., 360, 840 and 9800 columns for data

Network	Ref.	$\|V\|$	$\|L\|$
`internet2`	[14]	10	16
`atlanta`	[Orl+07]	15	44
`germany50`		50	88

Table 7.3: Network Data

instances associated with networks Internet1, `atlanta`, and `germany50`, respectively.

Network	# traffic requests	# VNF nodes	# generated columns	$z^{\star}_{\text{LP}}$	$\tilde{z}_{\text{ILP}}$	ε
`internet2`	360	5	382	2,086.7	2,086.7	0
		6	382	2,064.8	2,064.4	0
		7	379	2,064.4	2064.4	0
`atlanta`	840	7	1,198	2,591.5	2,592.9	5.4×10^{-4}
		8	1,611	2,581.7	2,581.7	0
		9	1,266	2,534.4	2,535.8	5.6×10^{-4}
`germany50`	9,800	24	28,083	4,217.6	4,218.0	8.1×10^{-5}
		25	28,140	4,211.9	4,212.3	8.8×10^{-5}
		26	26,977	4,190.7	4,191.0	7.4×10^{-5}

Table 7.4: Numerical results

7.5.3 Comparison ILP vs CG

In Figure 7.4, we compare the two models presented in Section 7.2 on the germany50 network. We also compare the computation time for NFV_CG when the Pricing Problem is solved using CPLEX or Bellman-Ford. We assume all nodes are VNF enabled nodes and the number of requests varies between 10 and 100% of the requests in an all-to-all traffic scenario.

Model NFV_ILP is solved exactly using the CPLEX ILP solver, while Model NFV_CG is solved using the solution scheme described in Section 7.3, i.e., with an ε-optimal solution scheme. As the accuracy of the solutions of Model NFV_CG is very good, the solutions of both models are identical. However, NFV_CG takes more time as the number of requests increases.

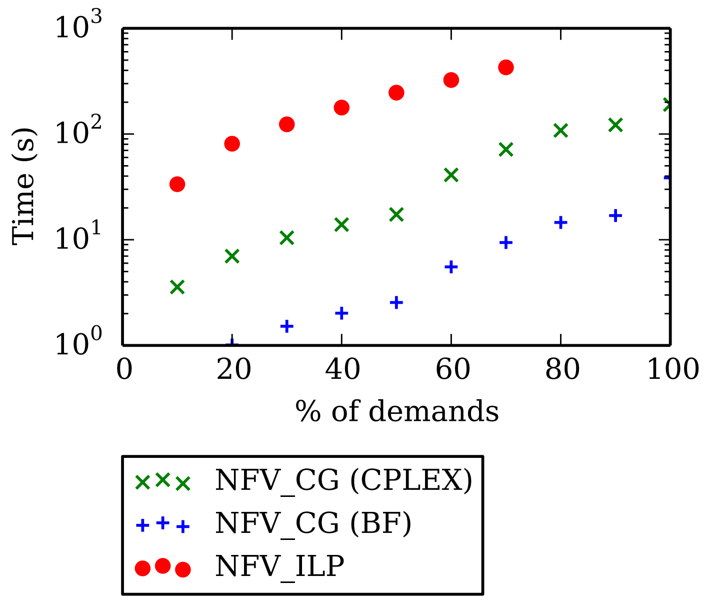
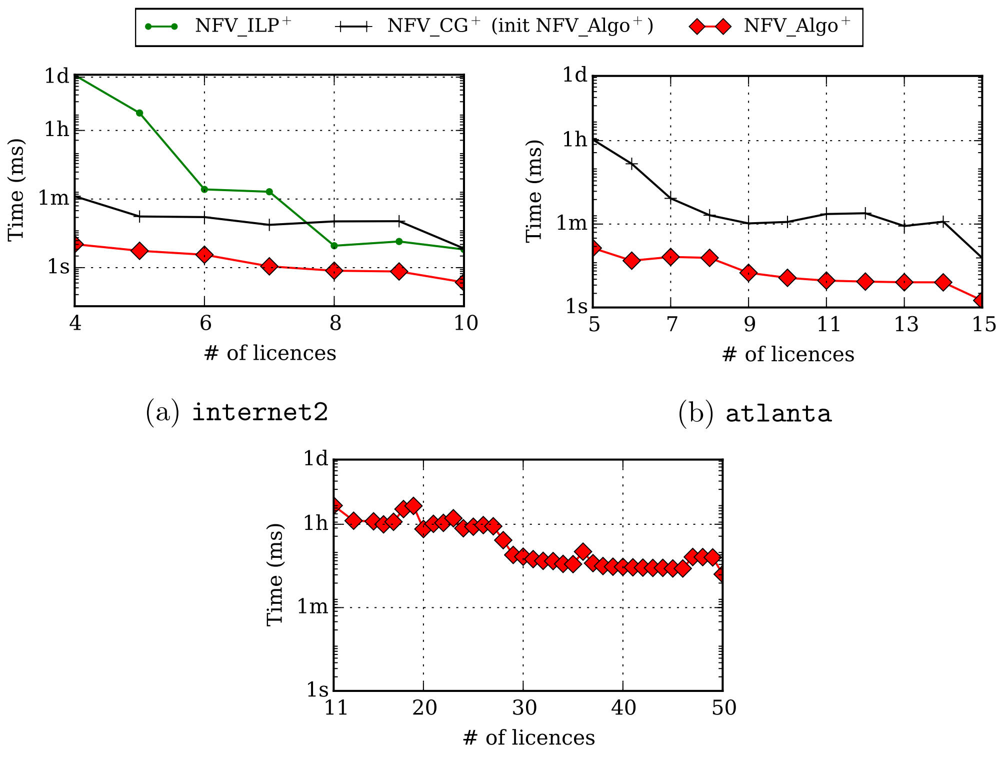


Figure 7.4: Computational times of NFV_ILP and NFV_CG on the germany50 network. NFV_ILP does not provides results for 80% and up in a reasonable time.

Indeed, when reaching 80% requests in the all-to-all scenario, NFV_ILP does not give any solution anymore, as the CPLEX solver runs out of memory. Comparatively, NFV_CG outputs an ε-optimal solution with all requests in two minutes and a half. Using Bellman-Ford, the solution is found in 62 seconds. See Figure 7.4 for the comparison of computing times, using the ratio of the computational times.

7.5.4 Bandwidth Requirement and Delay vs. Number of VNF Capable Nodes

In this set of experiments, we want to study the impact of the number of VNF nodes on the bandwidth requirement and the delay. Generating numerous VNF nodes could be quite costly (e.g., license price, CPU utilization, energy consumption...), and should be compensated by a significant decrease in the bandwidth requirement or justified by unacceptable delays otherwise. Our results show that this is not the case. We next discuss them in detail.



(a) `internet2` (b) `atlanta`

(c) `germany50`

Figure 7.5: Execution times of the three methods, the ILP, the column generation model, and the algorithm NFV_Algo$^+$, for different limits on the number of function replicas.

Figure 7.2 shows the bandwidth used for an overall 1Tbps traffic when the number of VNF nodes varies. As we allow more VNF nodes, the overall required bandwidth in the network decreases. This is as expected. Since every request requires a SFC, their provisioning must go through VNF nodes in the required order, possibly requesting more hops than in one of the shortest paths in the network. However, what we learn from Figure 7.2 is that, when reaching 50% for VNF capable nodes, the bandwidth gain is getting significantly smaller.

We next investigated the increase of the number of VNFs with respect to the delay, as measured by the number of hops. Results are described in Figure 7.3 using a box-and-whisker plot. It shows that the median value for the number of hops stabilizes as soon as the number of VNF nodes reaches 3, 9, 9 for the Internet 2, `atlanta` and `germany50` networks, respectively. While the stabilization occurs later with bandwidth requirements, these results say that, indeed, only few requests are affected when increasing the number of VNFs beyond the 3, 9 and 9 values for Internet 2, `atlanta` and `germany50` networks, respectively. Consequently, for homogeneous traffic as in our experiments, there is little advantage both in terms of delays and bandwidth requirements to increase much the number of VNF nodes. It might be slightly different with heterogeneous traffic, depending on the type of traffic that is impacted.

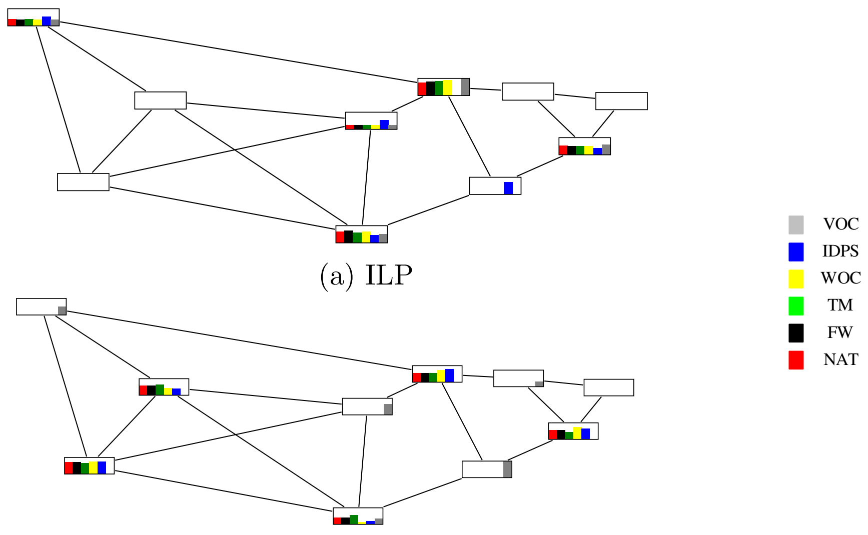
7.5.5 Limited Number of Function Replicas

We now evaluate the three methods proposed for a limited number of function replicas, NFV_ILP$^+$, NFV_CG$^+$ and the heuristic algorithm NFV_Algo$^+$. We consider the scenarios presented in Section 7.5. All nodes can potentially host VNFs. We first present the execution times and then the evolution of the bandwidth usage as a function of the number of allowed function replicas.

Execution Times We provide in Figure 7.5 the execution times of the NFV_ILP$^+$, NFV_CG$^+$, and of NFV_Algo$^+$ for `internet2`, `atlanta`, and `germany50`.

We first observe that, as expected, the more stringent the constraint on the number of function replicas, the harder it is for the methods to find a solution. For `internet2`, NFV_CG$^+$ takes only a few seconds to propose a solution when the number of replications is 10 (equal to the number of nodes), 9 or 8. However, more than one day is necessary to solve the case in which the number of allowed replications is 4. NFV_ILP$^+$ and NFV_Algo$^+$ experience a similar trend, with lower orders of execution times (respectively between 4 s and 1 min and between 400 ms and 4 s).

The second observation is that the execution time of NFV_ILP$^+$ becomes almost prohibitive (more than one day) when the number of allowed function replications is small. In fact, we were not able to run NFV_ILP$^+$ for networks larger than `internet2`. On the contrary, NFV_CG and NFV_Algo$^+$ have low execution times for any number of replications. Considering larger networks, NFV_ILP$^+$ runs on `atlanta` for any number of replicas, but not on `germany50`, for which, only NFV_Algo$^+$ provides solutions.



(a) ILP

(b) NFV_Algo$^+$

Figure 7.6: Placement of functions given by the ILP (a) and by NFV_Algo$^+$ (b) when the number of function replicas is limited to 5 for `internet2`. Bar height in a node gives the percentage of requests (between 0 and 36%) associated to the node function replica.

Replica Placement We provide the results of the function replica placement in Figure 7.6 for `internet2` and in Figure 7.7 for `germany50`. For each node and each function, we present the percentage of requests associated to the function replica. A value of 0 means that there is no function replica on the node. The height of the box represents the maximum value over all nodes,

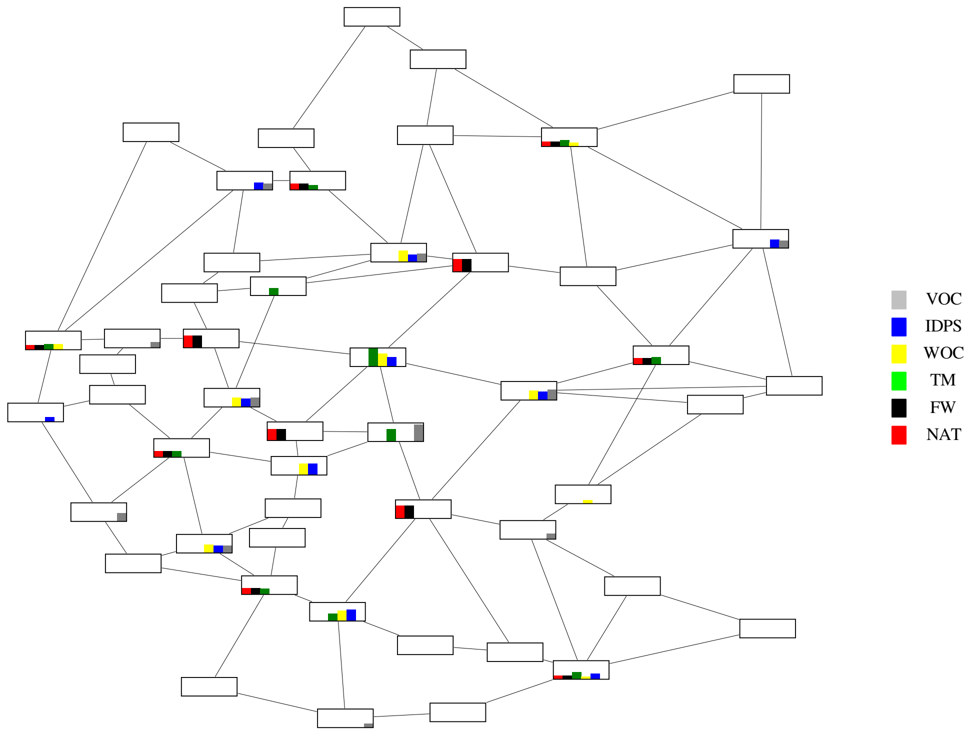
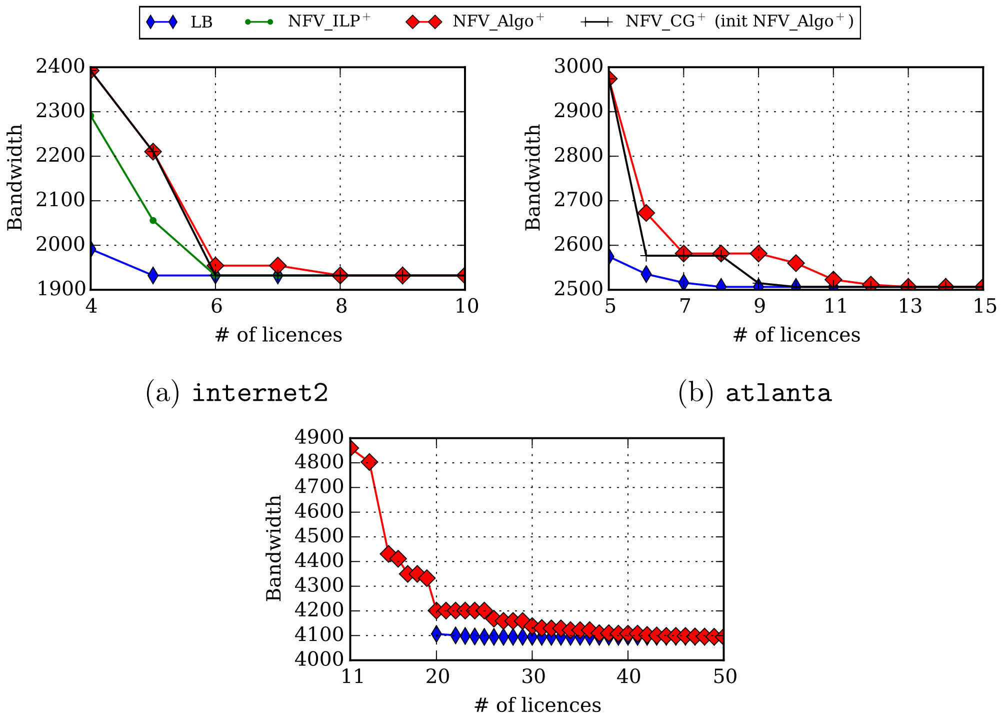


Figure 7.7: Placement of functions given by NFV_Algo$^+$ when the number of function replicas is limited to 11 for `germany50`. Bar height in a node gives the percentage of requests (between 0 and 20%) associated to the node function replica

respectively 36% for `internet2` and 20% for `germany50`. For `internet2`, the maximum number of function replicas was set to 5. We compare the result of the first phase of NFV_Algo$^+$ (placing function replicas) to the one of the ILP. We observe that the optimal solution provided by the ILP has selected five nodes (only a replica of the IPDS function is in another node), which are central in the network. This confirms the intuition that this kind of nodes are good candidates, as they have a small average distance to the other nodes, which are sources and destinations of requests. This intuition is at the core of NFV_Algo$^+$'s first phase, which selects a set of nodes minimizing the distances between the sources and destinations of the requests and the replica positions. NFV_Algo$^+$ also selected five nodes (only four replicas of the VOC function are in 4 other nodes). Three of them are common with the ILP, and two are different. The two last ones seem less central, but they have a high degree, which reduces their distances to the other nodes. For `germany50`, the number of function replicas was limited to 11. Only the result of NFV_-Algo$^+$ is presented as the ILP does not run on the network. We observe that central nodes with often a high degree are mostly selected. Indeed, again, they are nodes with a small average distance to the other nodes, and thus are suitable candidates for function replicas. We thus validate NFV_Algo$^+$, which provides good solutions to the placement problem. We now study the bandwidth usage.

Bandwidth Usage Figure 7.8a, 7.8b and 7.8c show the bandwidth usage respectively for `internet2`, `atlanta`, and `germany50`. The number of allowed replications varies between n and 1, where n is the number of nodes in the network. The results are given by NFV_Algo$^+$ (and additionally by ILP for `internet2`). To assess the quality of the solutions, we compare the results given by NFV_Algo$^+$ with the results of the relaxation of NFV_CG$^+$, which constitutes a lower bound of the optimal solution. The gap for NFV_CG$^+$ is higher than the one of NFV_CG. The reason is that limiting the number of replicas requires the introduction of new constraints, which are hard to relax efficiently[1]. However, the gap is within few percent for most of the values. For a small number of replicas, the gap is higher. However, the difference to optimal can be a lot smaller than the gap. For instance, for `internet2`, we see that the largest gap is around 20% when the difference to optimal given

[1]Note that, when the number of replicas is small for *germany*, the relaxation does not run. This shows the difficulty of the problem on large networks.



(a) `internet2`

(b) `atlanta`

(c) `germany50`

Figure 7.8: Bandwidth usage as a function of the number of allowed function replicas.

by NFV_ILP$^+$ is only 4%. To summarize, the solutions given by NFV_Algo$^+$ are within few percent from optimal for most values (and 16% for the worst case) and for the three networks. This shows that NFV_Algo$^+$ provides good solutions.

First, we need at least 4, 4, and 11 function replicas for `internet2`, `atlanta`, and `germany50`, respectively. We observe a general trend for the three networks. Starting from n, there is a first phase in which the number of function replications can be reduced without affecting the bandwidth usage significantly. This first phase corresponds to values between 10 and 6 for `internet2`, 15 and 7 for `atlanta`, 50 and 20 for `germany50`. In a second phase, on the contrary, the number of function replications can be reduced only at the cost of a strong increase of bandwidth usage: from 1932 to 2392 (+24%) for `internet2`, from 2506 to 2974 (+19%) for `atlanta`, from 4095 to 4860 (+19%) for `germany50`. Another way to state it, as a takeaway for a network operator: having a few more replicas than necessary leads to important gains, when adding further supplementary replicas has less impact.

7.6 Conclusions

In this chapter, we look at the Service Function Chain placement problem and propose two Integer Linear Program models to solve it. We show that a simple ILP does not scale well for large networks. However, with a decomposition model like Model NFV_CG, we can solve *exactly* the Service Function Chain Provisioning Problem. Taking into account the work of the literature, this is the first model that scales with an increasing number of nodes, but also, with an increase in the number of requests with service chain requirements. We are also the first to tackle the minimization of bandwidth with a maximum number of VNF replicas. To this end, we extended our models and proposed a heuristic algorithm based on NFV_CG and a capacitated k-mean clustering problem. Model NFV_CG then allowed us to look at the trade-off between the network bandwidth requirement, the number of VNF capable nodes, and the limit of the number of VNF replicas. We found that diminishing returns occur when adding VNF capable nodes, and that, when more than 50% of the network can host VNF, they are only little benefits. A similar trade-off exists for the number of replicas: from the configurations with the minimum possible number of replicas, adding a small number of them decreases the bandwidth usage significantly, but then adding more replicas shows a little supplementary gain.

Chapter 8

Energy Efficient Service Function Chaining

Contents

8.1 Introduction

In this chapter, we explore the potential energy savings of using NFV for Service Function Chains. We consider the problem of reducing network energy consumption while placing service functions using generic hardware along the

paths followed by flows. A specific difficulty is that the network functions have to be executed in a specific order and can be repeated several time in the same chain.
In summary, the contributions of this work are the following

- We show how virtualization can be used to improve the energy efficiency of networks, when demands have to go through a chain of services. To the best of our knowledge, we are the first to propose such a method.
- We propose a way of modeling this problem based on Integer Linear Programming. The ILP can solve optimally instances of small sizes. We thus propose and validate a *heuristic algorithm*, GreenChains, to handle instances of larger sizes.
- We formulate a Column Generation model to solve the EE-SFCP problem on large instances.
- We provide enhancements of the model with the use of cuts as the EE-SFCP problem is a difficult optimization problem. As a matter of fact, it contains a sharp On-Off phenomena, as a network device consumes a large portion of its energy as soon as it is used, even if very lightly used. Cuts allow the reduction of the integrality gap.
- This allows us to carry out extensive simulations on networks of different sizes. We study three different scenarios: a *legacy scenario* which serves as baseline for comparison, a *hardware scenario* in which the routing can be changed dynamically by a centralized SDN controller, but in which network functions are executed by specific hardware, and finally, an *NFV scenario* in which the network functions are virtualized and can be placed dynamically. We show that between 22% to 62% of energy can be saved during the night while respecting the constraints of the service chains.
- Finally, we propose a latency analysis with respect to switching off some network elements for energy savings.

The chapter is organized as follows. The problem is presented in Section 8.1.1 along the power model and the layered graph model used in our mathematical formulations. We present in Sections 8.2, 8.3 and 8.4 the ILP formulation, GreenChains and column generation scheme, respectively. We then compare the models and assess their quality in Section 8.5.

8.1.1 SFC and VNF Placement

Notations

We assume the network to be represented by a directed graph $G = (V, A)$, where V is the set of nodes (indexed by u), and L is the set of links (indexed by (u, v)). Each node $u \in V$ has a set of computing, storage and network resources denoted by C_u to host network functions. Within this study, we assume that the resources are described by a given number of CPU cores.

Traffic is described by a set of requests D, in which each request d is defined by a 4-tuple (u_s, u_d, c, D^c_{sd}), where u_s is the source of the request, u_d its destination, D^c_{sd} its bandwidth requirement, and c the requested service chain. Indeed, each request d is associated with a given application, which is required to pass through a given SFC. Let F be the overall set of virtual functions arising in the service chains, indexed by f, and C be the set of service chains, indexed by c. Each service chain c corresponds to a sequence of n_c functions $f^c_1, \ldots, f^c_i, \ldots, f^c_{n_c}$, where f^c_i denotes the ith function of chain c. Note that some functions may appear more than once in a given chain. Each virtual function f has its one resource requirement, and we denote by Δ_f the number (fraction) of cores required by the function f per bandwidth unit.

The *Energy Efficient Service Function Chain Provisioning (*EE-SFCP*)* problem consists in jointly provisioning a set D of requests coupled with service function chains C and placing virtual functions arising in the chains, in order to minimize the network energy consumption, subject to link and node capacities.

Power Model

Campaigns of measures of power consumption (see, e.g., [Cha+08]) show that a network device consumes a large amount of its power as soon as it is switched on and that the energy consumption does not depend much on the load. Following this observation, on/off power models have been proposed and studied. Later, researchers and hardware constructors have proposed more energy proportional hardware models [Nic+12]. To encompass those different models, we use a hybrid power model in which the power of an active link (u, v) is expressed as

$$\mathrm{P}_{uv} = P_{uv}^{\mathrm{IDLE}} + \frac{\mathcal{F}_{uv}}{C_{uv}^{\mathrm{LINK}}} P_{uv}^{\mathrm{LOAD}},$$

where P_{uv}^{IDLE} represents the energy used when the link (u, v) is switched on, $\mathcal{F}_{uv}$ the bandwidth that is carried on (u, v), and P_{uv}^{LOAD} the additional energy consumed by (u, v) when it is fully capacitated, i.e., when the amount of carried bandwidth equals the transport capacity (C_{uv}^{LINK}) of link (u, v).
We assume that links can be put into sleep mode, by putting to sleep both endpoint interfaces. Two links in opposite direction between a pair of nodes are assumed to be in the same state (active or in sleep mode), as the send and receive elements of a unidirectional fiber are usually controlled by the same interface. Routers cannot be put into sleep mode, as there are the sources/destinations of network traffic. However, cores may be put into sleep mode and the power used by node u is equal to

$$\mathrm{P}_u = \mathrm{P}_u^{\mathrm{UNIT}} \times \#\mathrm{cores}$$

with $\mathrm{P}_u^{\mathrm{UNIT}}$ being the energy consumption of a single core.

Layered Graph

Like in Chapter 7, we use a layered graph G^{L} that is defined as follows. We add $\max\limits_{c \in C} n_c$ layers to the original graph G and each layer contains a copy of G. For every node $u \in V$, let u^i be the corresponding node in the ith layer ($i = 0, 1, \ldots, n_c$). Every $(i-1, i)$ layer pair is connected by (u^{i-1}, u^i) links. Provisioning of a chain and node placement of its functions amounts to find a path from node u_s on the first layer, i.e. u_s^0, to node u_d on the n_cth layer, i.e., $u_d^{n_c}$. Placement of a function on a node is given by the endpoints of the link used to switch between layers.

8.2 Compact formulation

We now present the ILP formulation for the EE-SFCP problem. Let us first introduce the set of variables.

- $x_{uv} \in \{0, 1\}$ where $x_{uv} = 1$ if link (u, v) is active, 0 otherwise

- $f_{iuv}^{sd,c} \in \{0,1\}$ where $f_{iuv}^{sd,c} = 1$ if the flow for the request (u_s, u_d, c, D_{sd}^c) uses the link (u,v) in layer i. We consider here un-splittable routing.
- $a_{iu}^{sd,c} \in \{0,1\}$ where $a_{iu}^{sd,c} = 1$ if the ith function of the chain c is executed on node u for the request (u_s, u_d, c, D_{sd}^c).
- $\kappa_u \in \mathbb{N}$, number of CPU cores used in node u.
- $f_{uv} \in \mathbb{R}$, flow passing through link (u,v). This variable is linked and is added to the ILP for clarity of the presentation.

The formulation is given as follows.

Objective

$$\min \sum_{(u,v)\in A} \left(P_{uv}^{\text{IDLE}} \times x_{uv} + P_{uv}^{\text{LOAD}} \times \frac{f_{uv}}{C_{uv}^{\text{LINK}}} \right) + \sum_{u\in V} \mathrm{P}_u \kappa_u \tag{8.1}$$

Flow Conservation

$$\sum_{u\in N^+(u)} f_{iuv}^{sd,c} - \sum_{u\in N^-(u)} f_{iuv}^{sd,c} + a_{iu}^{sd,c} - a_{i-1u}^{sd,c} = 0$$
$$\forall (u_s, u_d) \in \mathcal{SD}, c \in C_{sd}, u \in V, 0 < i < n_c \tag{8.2}$$

$$\sum_{u\in N^+(u)} f_{0uv}^{sd,c} - \sum_{u\in N^-(u)} f_{0uv}^{sd,c} + a_{0u}^{sd,c} = \begin{cases} 1 \text{ if } u = u_s, \\ 0 \text{ else} \end{cases}$$
$$\forall (u_s, u_d) \in \mathcal{SD}, c \in C_{sd}, u \in V \tag{8.3}$$

$$\sum_{u\in N^+(u)} f_{n_c uv}^{sd,c} - \sum_{u\in N^-(u)} f_{n_c uv}^{sd,c} - a_{n_c-1,u}^{sd,c} = \begin{cases} -1 \text{ if } u = u_d, \\ 0 \quad \text{else} \end{cases}$$
$$\forall (u_s, u_d) \in \mathcal{SD}, c \in C_{sd}, u \in V \tag{8.4}$$

Link Capacity

$$f_{uv} = \sum_{(u_s,u_d)\in\mathcal{SD}} \sum_{c\in C_{sd}} \sum_{i=0}^{n_c} D_{sd}^c \times f_{iuv}^{sd,c} \le C_{uv}^{\text{LINK}} \times x_{uv} \qquad \forall (u,v) \in A \tag{8.5}$$

Number of CPU cores used

$$\sum_{(s,t)\in\mathcal{D}} \sum_{i=0}^{n_c-1} \left(\Delta_{f_i^c} D_{sd}^c\right) \times a_{iu}^{sd,c} \leq \kappa_u \qquad u \in V \tag{8.6}$$

Node Capacity

$$\kappa_u \leq C_u^{\text{NODE}} \qquad u \in V \tag{8.7}$$

8.3 Solving large Instances with GreenChains

As the ILP proposed in the previous section cannot provide solutions for large networks, we propose here an ILP-based heuristic algorithm called Green-Chains to solve the EE-SFCP problem. The problem can be decomposed into three sub-problems.

- First, the *energy saving problem* tries to put into sleep mode as many links and cores as possible to decrease the energy consumption of the network.
- Second, the *routing problem* computes a path for each request, respecting the link capacity constraints.
- Last, the goal of the *service chain placement problem* is to find a placement of the NVF respecting the capacities of the nodes and the order defined by the service chains, according to the path computed for each request.

8.3.1 Energy Saving Module.

The goal of this module is to put links into sleep mode.

It first launches the routing module and then places the network functions on the requests' paths. If both modules succeed, it creates a list U of all links according to their usage (volume of traffic). It then chooses the least loaded link $\ell_{\min}$ as a candidate to be put in sleep mode. It now considers the graph $G' = (V, A \setminus \{\ell_{\min}\})$. It launches the routing and placement modules again. If they succeed, $\ell_{\min}$ is put in sleep mode. The list U is actualized with the new routing, as well as the least loaded link. If at least one of the two modules fails, GreenChains considers that $\ell_{\min}$ cannot be into sleep mode and the link is kept active for the final solution. The second element of U is then considered. The algorithm goes on till all links have been tried and set either as definitely in sleep mode or active. The goal of this module is to reduce the energy used by the links by putting in sleep mode as many links as possible.

8.3.2 Routing Module

We consider the requests one by one and compute a weighted shortest path on a residual graph for each one of them. To favor links with a lower load, the weight of the link in the residual graph is equal to the inverse of its residual capacity. When we assign a path to a request, we decrease the capacity of the residual graph by the amount of charge requested. Furthermore, when considering a new demand to be routed, we remove links with a residual capacity smaller than the demand.

8.3.3 Service Chain Placement Module.

This module is in charge of choosing the execution location of the chains functions. We propose the following ILP that aims at minimizing the total number of cores used.

Given a path $P_{sd,c}$ for every request (u_s, u_d, c, D^c_{sd}), we need to find the execution location of each function of the chain c. Each node of the path is indexed by i, i.e., $P^i_{sd,c}$ is the ith node of $P_{sd,c}$.

We introduce the following two sets of variables.

- $a^{sd,c}_{iu} \in \{0,1\}$ where $a^{sd,c}_{iu} = 1$ if f^c_i for request (u_s, u_d, c, D^c_{sd}) is executed on node u
- $\kappa_u \in \mathbb{N}$, #cores used in node u.

The formulation is given as follows.

Objective function

$$\min \sum_{u \in V} \kappa_u \tag{8.8}$$

Execution constraints

$$\sum_{u \in P_{sd,c}} a_{iu}^{sd,c} = 1 \qquad (u_s, u_d) \in \mathcal{SD}, c \in C_{sd}, 1 \leq i \leq n_c \tag{8.9}$$

Order constraints

$$a_{i,P_{sd,c}^k}^{sd,c} \leq \sum_{j=1}^{k} a_{i-1,P_{sd,c}^j}^{sd,c}$$

$$\forall (u_s, u_d) \in \mathcal{SD}, c \in C_{sd}, 1 \leq k \leq len(P_{sd,c}), 1 \leq i \leq n_c \tag{8.10}$$

Number of cores used

$$\sum_{(u_s,u_d) \in \mathcal{SD}} \sum_{c \in C} \sum_{i=1}^{n_c} \left(\Delta_{f_i^c} D_{sd}^c\right) \times a_{iv}^{sd,c} \leq k_v \qquad \forall u \in V \tag{8.11}$$

Node Capacity constraints

$$\kappa_u \leq C_u \quad \forall u \in V \tag{8.12}$$

8.4 Decomposition Models

As the ILP does not scale, we propose a column generation scheme to help validate our heuristic for larger networks. We first present here a model using Column Generation, *CG-simple*. We then introduce two variants of the models, *CG-cut*, and *CG-cut+*. Indeed, problems dealing with energy-efficiency frequently lead to large integrality gap and bad precision. This is due to the On-Off phenomena of power models, which translates into large steps of the objective function. We thus try to improve the precision of the model by introducing different sets of constraints. We discuss the precision of the models in Section 8.5.3.

8.4.1 Column Generation Formulation

We propose a column generation formulation that relies on the concept of *Service Path*: each *Service Path* p is associated with a 4-uplet (u_s, u_d, c, D_{sd}^c) and defines: *(i)* a potential route for the request (u_s, u_d, c, D_{sd}^c) between u_s and u_d, *(ii)* node placement of the functions of chain c along the potential route. A route is described by parameters δ_{uv}^p, equal to the number of occurrences of the link (u, v) in the path p. Node placement is given by a_{ui}^p, equal to 1 if the ith function of the chain c is located at node u, 0 otherwise. We denote by P^c_{sd} the overall set of *Service Path* for each request (u_s, u_d, c, D^c_{sd}).

We now define the set of variables. First set of decision variables: $x_{uv} = 1$ if link $(u, v)]$ is on (active), 0 otherwise. Note that links are powered off by pair, i.e., $x_{uv} = x_{vu}$. Second set of decision variables: $y^p_d = 1$ if demand d is routed using configuration p, 0 otherwise. Integer variables: $\kappa_v = \#$ required cores in node u.

The objective, i.e., the minimization of the energy, can be written

$$\min \underbrace{\sum_{(u,v)\in A} P^{\text{IDLE}}_{uv} x_{uv}}_{\substack{\text{link switch}\\ \text{on energy}}} + \underbrace{\sum_{(u,v)\in A}\sum_{p\in P^c_{sd}} \delta^p_{uv} \left(\sum_{d=(u_s,u_d,c)\in D} \frac{D^c_{sd}}{C^{\text{LINK}}_{\ell}} \mathrm{P}^{\max}_{uv} \right) y^p_d}_{\text{link bandwidth energy}} + \underbrace{\sum_{u\in V} \mathrm{P}_u\, \kappa_u}_{\text{node resource energy}} \tag{8.13}$$

The constraint set decomposes into three sets of constraints.

$$\text{One path per demand:} \quad \sum_{p\in P^c_{sd}} y^p_d = 1 \qquad (u_s, u_d) \in \mathcal{SD}, c \in C_{sd} \tag{8.14}$$

$$\text{Link capacity:} \quad \sum_{d=(u_s,u_d,c)\in D} \sum_{p\in P^c_{sd}} D^c_{sd}\, \delta^p_{uv}\, y^p_d \le x_{uv}\, C^{\text{LINK}}_{uv} \qquad (u,v) \in A \tag{8.15}$$

$$\text{Node capacity:} \sum_{d\in D}\sum_{p\in P^c_{sd}} D^c_{sd} \left(\sum_{i=1}^{n_c} \Delta_{f_i} a^p_{uf_i} \right) y^p_d \le \kappa_u \le C^{\text{NODE}}_u \quad u \in V \tag{8.16}$$

As we faced issues with large integrality gaps, we enhanced model (8.13)-(8.16) with different sets of cuts, through the next two models.

***CG-cut* model.** The first set of cuts in (8.17) states that, for each node, at least one incident link should always be on. Moreover, the second inequality given by Equation (8.18) enforces that at least $n-1$ links should be active to have a connected network (or different if not all-to-all).

$$\sum_{u \in N^+(u)} x_{uv} \geq 1 \qquad u \in V \tag{8.17}$$

$$\sum_{(u,v) \in A} x_{uv} \geq n-1 \tag{8.18}$$

***CG-cut+* model.** We further enhance the *CG-cut* model with:

$$x_{uv} \geq \sum_{p \in P^c_{sd}} \gamma^p_{uv}\, y^p_d \qquad (u,v) \in A, (u_s,u_d) \in \mathcal{SD}, c \in C_{sd} \tag{8.19}$$

where $\gamma^p_{uv} = 1$ if the link (u,v) belong the path p. Using (8.14), it follows that $\sum\limits_{p \in P^c_{sd}} \gamma^p_{uv}\, y^p_d \leq 1$. It avoids the use of a big M formulation at the expense of a large number of constraints.

8.4.2 Solution Scheme

To solve the model of Section 8.4.1 efficiently, we need to recourse to column generation for solving the linear relaxation, and then to derive an ILP value, using the last restricted master problem. We refer the reader to [Chv83] for more precision on linear programming and column generation schemes.

There is a configuration generator, i.e., pricing problem, for each request (u_s, u_d, c, D^c_{sd}). Two sets of decision variables are required. First set is made of variables φ^i_{uv} such that $\varphi^i_{uv} = 1$ if the provisioning of demand d uses link (u,v) in layer i of the layered graph G^{L}, 0 otherwise. Second set contains variables a^i_u such that $a^i_u = 1$ if the ith function (f^c_i) of chain c for request (u_s, u_d, c, D^c_{sd}) is placed on NFV node u, 0 otherwise. The formulation of the *Service Path* generator is given as follows.

$$\begin{aligned} \min - u^{(8.14)}_{sd} \\ &+ \sum_{(u,v) \in A} \sum_{i=0}^{n_c} \varphi^i_{uv} \times \left(\mathrm{P}^{\max}_u v \frac{D^c_{sd}}{C^{\text{LINK}}}_{uv} + u^{(8.15)}_{uv} D^c_{sd} \right) \\ &+ \sum_{u \in V} \sum_{i=0}^{n_c - 1} a^i_u \times \left(u^{(8.16)}_v \Delta_{f_i} D^c_{sd} \right) \end{aligned} \tag{8.20}$$

Path computation (flow conservation constraints):

$$\sum_{v \in N^+(u)} \varphi^i_{uv} - \sum_{u \in N^-(u)} \varphi^i_{uv} + a^i_u - a^{i-1}_u = 0 \qquad u \in V, 0 < i < n^c \quad (8.21)$$

$$\sum_{v \in N^+(u)} \varphi^0_{uv} - \sum_{v \in N^-(u)} \varphi^0_{uv} + a^0_u = \begin{cases} 1 \text{ if } v = v_s \\ 0 \text{ else} \end{cases} \qquad u \in V \quad (8.22)$$

$$\sum_{v \in N^+(u)} \varphi^{n_c}_{uv} - \sum_{v \in N^-(u)} \varphi^{n_c}_{uv} - a^{n_c}_v = \begin{cases} -1 \text{ if } v = v_d \\ \ \ 0 \text{ else} \end{cases} \qquad u \in V. \quad (8.23)$$

$$\text{Link capacity:} \quad D^c_{sd} \sum_{i=0}^{n_c} \varphi^i_{uv} \leq C^{\text{LINK}}_u v \qquad (u, v) \in A. \quad (8.24)$$

$$\text{Node capacity:} \quad D^c_{sd} \sum_{i=0}^{n_c} \Delta_{f_i} a^i_u \leq C^{\text{NODE}}_u \qquad u \in V. \quad (8.25)$$

Speeding up the Pricing Problem

The Pricing Problem corresponds to a constrained shortest path with negative weights on the layered graph, and we can use CPLEX to solve it. However, if we discard the capacity constraints, the problem becomes the simpler *shortest path with negative weights* problem. It can be solved much faster than the original problem using the Bellman-Ford shortest path algorithm. Since we remove the capacity constraints, the set of solutions considered is a superset of the initial set of solutions. It is possible to find a path that might use more resources than available. In this case, we fall back the ILP solver to obtain a valid path. The weight of the inter-layer arcs is thus given by

$$w_{iu} = u^{(8.16)}_u \Delta_{f_i} D^c_{sd} \qquad 0 \leq i \leq n_c, (u, v) \in A$$

and the weight of intra-layer arcs by

$$w_{iuv} = \mathrm{P}^{\max}_{uv} \frac{D^c_{sd}}{C^{\text{LINK}}{}_{uv}} + u^{(8.15)}_{uv} D^c_{sd} \qquad 0 \leq i < n_c, u \in V$$

Particularities of *CG-cut+*

By introducing the constraints (8.19) into the model, we also need to introduce a new set of variable γ_l into the Pricing Problem that indicates if the link (u, v) is used in the path. The objective function becomes

$$\begin{aligned}
\min - & u_{sd}^{(8.14)} \\
& + \sum_{(u,v)\in A} \sum_{i=0}^{n_c} \varphi_{uv}^{i} \times \left(\mathrm{P}_{u}^{\max} v \frac{D_{sd}^{c}}{C^{\mathrm{LINK}}{}_{uv}} + u_{uv}^{(8.15)} D_{sd}^{c} \right) \\
& + \sum_{u\in V} \sum_{i=0}^{n_c} a_{u}^{i} \times \left(u_{u}^{(8.16)} \Delta_{f_i} D_{sd}^{c} \right) \\
& + \sum_{(u,v)\in A} \gamma_l u_{sd,c,l}^{(8.19)}
\end{aligned} \tag{8.26}$$

and the link capacities constraints becomes

$$D_{sd}^{c} \sum_{i=0}^{n_c} \varphi_{uv}^{i} \leq C_{\ell}^{\mathrm{LINK}} \times \gamma_u v \qquad (u,v) \in A. \tag{8.27}$$

Moreover, adding enhanced cuts creates negative cycles in the layered graph used for the Pricing Problem. We choose not to get rid of the cycles by enumerating them all. Instead, we check if the solution provided by the solver contains any negative cycles. If that is the case, we add the corresponding constraints in the formulation and call the solver again. We repeat this process until the obtained solution no longer contains any negative cycles or the reduced cost is no longer negative. Removing the cycle does not impact too much the performance of the column generation scheme as it is executed only a few times at the start of the algorithm.

8.5 Numerical Experiments

In this section, we investigate the energy savings obtained by the Column Generation model. We compare the results with the ones of NFV_Algo^{+} heuristic algorithm. We first present the data sets we use for the experiments. We then take a look at the precision of the solutions obtained by the Column Generation model and GreenChains. We investigate different improvements of the model presented in Section 8.1.1. We then present the energy savings achieved for network topologies of different sizes. Last, we discuss the impact of the solutions on link usage and path lengths.

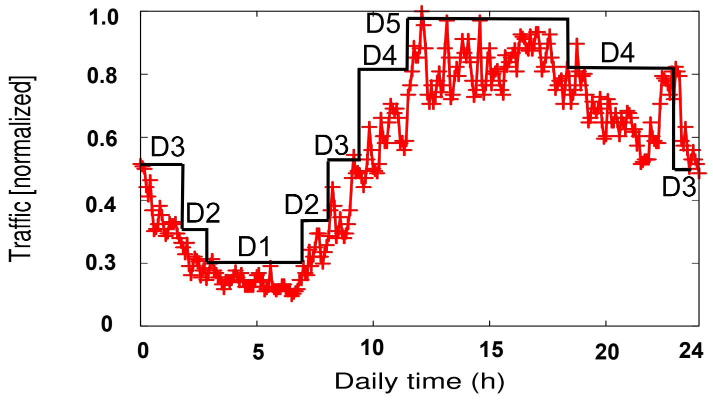


Figure 8.1: Normalized daily variation of traffic of a France Telecom network link and multi-period approximation

Service Chain	Chained VNFs	rate	% traffic
Web Service	NAT-FW-TM-WOC-IDPS	100 kbps	18.2%
VoIP	NAT-FW-TM-FW-NAT	64 kbps	11.8%
Video Streaming	NAT-FW-TM-VOC-IDPS	4 Mbps	69.9%
Online Gaming	NAT-FW-VOC-WOC-IDPS	50 kbps	0.1%

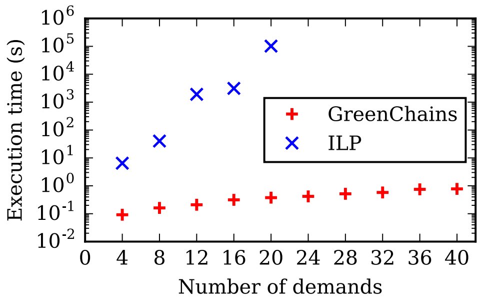
Table 8.1: Service Chain Requirements [STV15]

8.5.1 Data sets

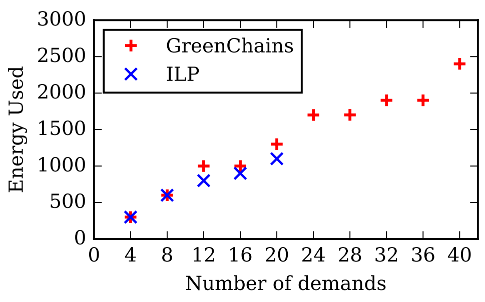
In networks, each type of flows has to go through a different chain of network services. In our experiments, we consider four of the most frequent types of flows, as presented in Table 8.1: Video Streaming, Web Service, Voice-over-IP (VoIP), and Online Gaming. The traffic percentages are from [Cis15]. For each one, we give the ordered set of functions required and the bandwidth used. In total, we have six different functions, and each function requires a different amount of cores to be executed.

We tested the CG models and GreenChains on three topologies of different sizes from SNDlib [Orl+10]: *pdh* (11 nodes and 64 directed links), *atlanta* (15 nodes and 44 directed links), and *germany50* (50 nodes and 176 directed links).

For each network, we generate a set of demands from the traffic matrices provided in SNDlib: we divide each aggregate flow from a source to a destination into four demands corresponding to the four different types of traffic. The original load of the flow is conserved, and each sub-flow load is given by



(a) Execution time



(b) Energy used

Figure 8.2: Comparison between the compact formulation and GreenChains

the distribution of the last column of Table 7.2. For example, a flow with a charge of 1 is split into a Web Service, a VoIP, a Video Streaming and an Online Gaming sub-flows with a load of 0.182, 0.118, 0.699 and 0.001, respectively.

We tested the solution on a daily traffic to see how much energy can be saved during the day or at night. The variations of traffic come from a trace of a typical France Telecom link shown in Figure 8.1. Previous work [Ara+16] indicates that using a small number of configurations during the day is enough to obtain most of the energy savings. In our case, we considered five different levels of traffic called D1 to D5. D1 represents the period with the lowest amount of traffic and D5 the one with the highest.

8.5.2 Compact formulation evaluation

We compare the results obtained by the heuristic algorithm, GreenChains, with the optimal results given by the integer linear program on a small network, *pdh*, with 11 nodes and 64 links. We consider instances with an increasing complexity: the number of demands varies from 4 to 40. Note that we consider multiples of 4 demands, as the traffic between a pair of nodes is divided into four different demands corresponds to different categories of traffic.

We compare the execution times of the ILP model and the algorithm in Figure 8.2a. The experiments are made on a Intel(R) Xeon(R) CPU E5620 @2.40GHz 16 cores with 24GB of RAM. We see that the ILP model can be used to solve the problem with a reasonable time for a maximum number of 16 demands. In this case, it takes around 45 minutes to return the optimal solution. The increase is exponential: for 20 demands, the execution time is almost 3 hours. GreenChains, on the contrary, is a lot faster as it can find a solution in less than 1 second for 20 demands (0.38 s). It solves an instance with 40 demands in 0.78 s and the all-to-all instance (with 440 demands), considered in the following, in less than 7 s. We see that the ILP cannot be used in practice to solve instances with a large number of demands, and thus we use the GreenChains for the experiments on larger networks in the following.

The results regarding energy savings are given in Figure 8.2. GreenChains finds results within a precision between 0% to 16% for the different number of demands. We consider this as good results given the difficulty of the EE-SFCP problem. Moreover, it means that the potential energy savings of using dynamic traffic and virtualization are in fact even greater than the one presented in the following.

8.5.3 Quality of the Column Generation models

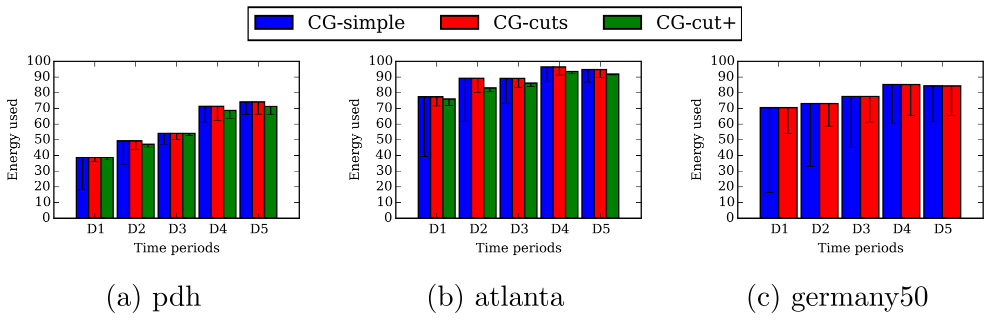
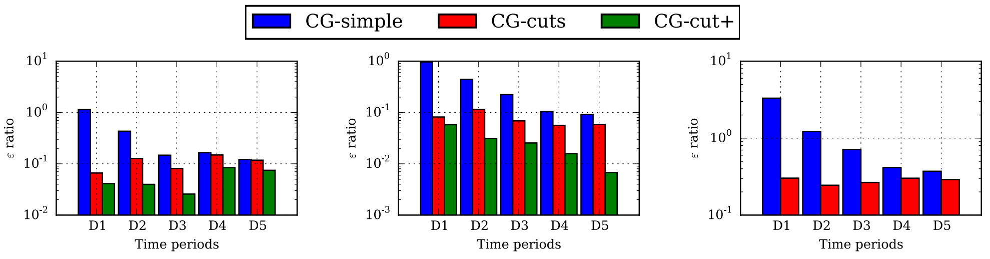


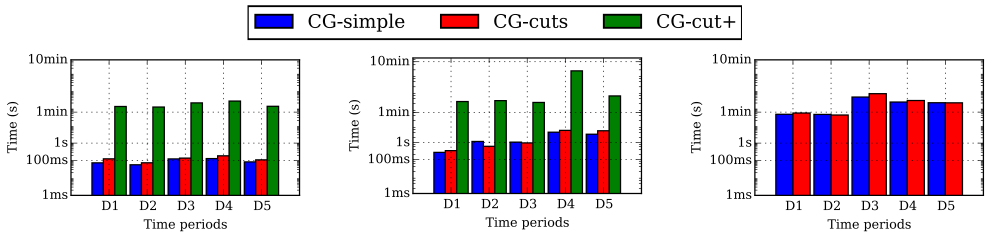
Figure 8.3: Performance of all three CG models on the (a) *pdh*, (b) *atlanta* and, (c) *germany50* network topologies.

We now compare the performance of the three different CG models (*CG-simple*, *CG-cut*, and *CG-cut+*) with respect to their accuracy as given by $\varepsilon = (\tilde{z}_{\text{ILP}} - z^{\star}_{\text{LP}})/z^{\star}_{\text{LP}}$, where $z^{\star}_{\text{LP}}$ represents the optimal value of the relaxation of the Restricted Master Problem, and $\tilde{z}_{\text{ILP}}$ the integer solution obtained at the end of the column generation algorithm. In Figure 8.3, 8.4, and 8.5, we compare the solutions found by the three CG models for all three networks and for the



(a) pdh (b) atlanta (c) germany50

Figure 8.4: Accuracy, ε, of all three CG models on the (a) `pdh`, (b) `atlanta` and, (c) `germany50` network topologies



(a) pdh (b) atlanta (c) germany50

Figure 8.5: Execution times of all three CG models on the (a) `pdh`, (b) `atlanta` and, (c) `germany50` network topologies

5 different levels of traffic. We first observe in Figure 8.3, in which error bars represent the gap between the relaxed and integer solutions, that *CG-simple* and *CG-cut* provide similar solutions. However, ε dramatically varies, as shown in Figure 8.4. Cuts significantly improves ε: for *CG-simple*, it varies between 12% and 113% for pdh, 10% and 97% for atlanta, and 37% and 330% for germany50. For *CG-cut*, ε is between 7 to 15% for pdh, 6 and 12% for atlanta, and 24 and 30% for germany50. The ratio is further improved with *CG-cut+*: between 4 and 8% for pdh, 1 and 6% for atlanta. However, no solutions were found in a reasonable amount of time for the germany50 topology. As the energy savings are similar for the three models, it shows that the *three CG models provide rather accurate solutions, as confirmed by the solutions and accuracy of the* CG-cut *and* CG-cut+ *models.*

Finally, in Figure 8.5, we compare the execution times of the models. We observe that *CG-cut+* execution time (between 17 s and 5 h) is orders of magnitude higher that the one of *CG-simple* (between 50 ms and 440 s) and *CG-cut* (between 70 ms and 670 s). This is greatly due to the fact that we speed up the resolution of the two previous model using the Bellman-Ford shortest path algorithm for the Pricing Problem. The second factor is that *CG-cut+*'s cuts slow the convergence time of the column generation drastically.

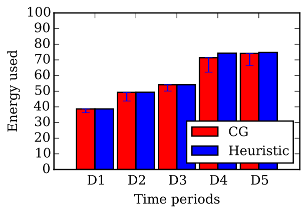
We now *focus on the* CG-cut *model*, as it offers the best compromise in terms of accuracy (w.r.t. *CG-simple* model) and computation time requirements (w.r.t. *CG-cut+* model) to solve large networks.

8.5.4 Energy Savings

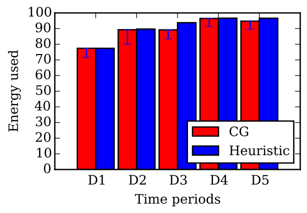
We now compare the energy savings obtained by GreenChains and *CG-cut*. We consider three scenarios in the experiments:

- *Legacy scenario.* This scenario corresponds to the one of a legacy network, whose operator does not try to reduce the energy consumption of its network. Its goal is to minimize the total bandwidth used while respecting the link capacity and the chain constraints. This scenario is used as a *baseline for comparison* for the energy-aware algorithms.

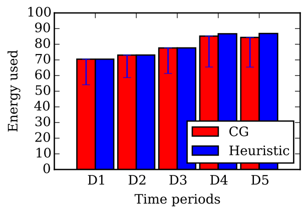
- *Hardware scenario.* The hardware scenario corresponds to one of an SDN (non-virtualized) network in which an operator tries to reduce its energy consumption by adapting the routing to the demands. In



(a) pdh



(b) atlanta



(c) germany50

Figure 8.6: Energy used for GreenChains and the CG model on the three topologies.

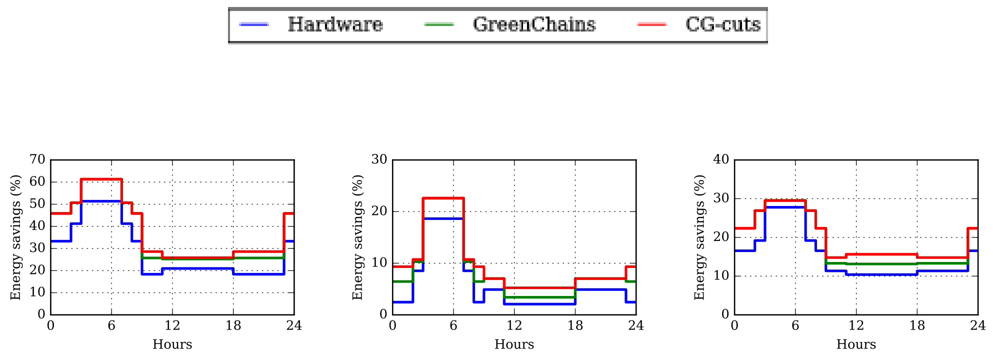
this scenario, the network functions are carried out by some specific hardware placed at given positions in the network.

- *NFV scenario.* The NFV scenario is the one of a virtualized SDN network in which generic hardware nodes can execute any virtual network functions. This is the scenario solved by the solutions provided in Sections 8.2, 8.3 and 8.4.

We provide in Figure 8.6 the energy used for the five levels of demands for `pdh`, `atlanta`, and `germany50`. The values are normalized: 100 corresponds to the legacy scenario. We also present in Figure 8.7 the corresponding energy savings during the day. We see that we obtained important savings using virtualization: between 25 and 61% for *pdh*, 5 and 22% for *atlanta*, and 15 and 30% for *germany50*.

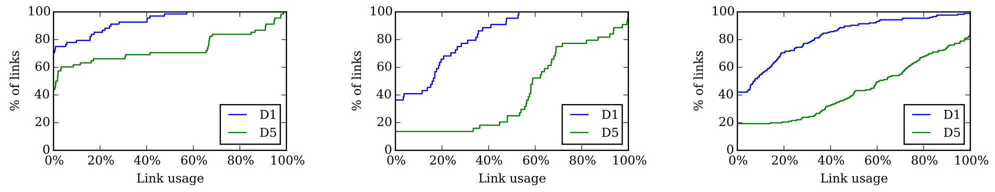
Validating GreenChains with *CG-cut*

We now compare the solutions provided by both GreenChains and *CG-cut* in Figure 8.6. Error bars on the *CG-cut* solutions represent the lower bounds given by $z^\star_{\text{LP}}$. For the lowest traffic periods (D1, D2 and in D3), both methods provide similar solutions for `pdh` and `germany50`. The CG model provides slightly better solutions when the traffic is higher, with a difference of 3 and 1% for *pdh*, of 5,and 2 for *atlanta*, and of 2 and 3% for *germany50* respectively in the D5 period. Observe that, even if CG only provides slight improvement of the heuristic's solutions, it shows (c.f. ε accuracy value) that the heuristic gives good results, regardless of the traffic period.



(a) pdh (b) atlanta (c) germany50

Figure 8.7: Saved energy for GreenChains for the three network topologies.

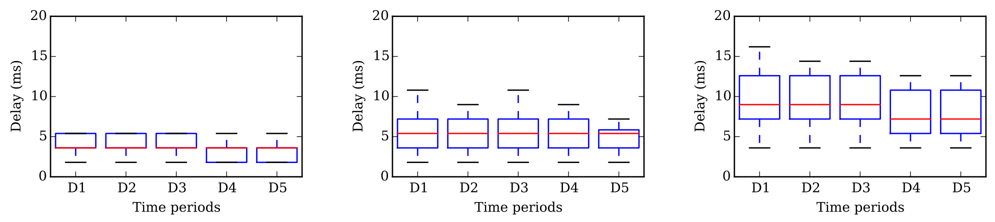


(a) pdh (b) atlanta (c) germany50

Figure 8.8: Link load for GreenChains for the three network topologies.

Link load

To reduce the amount of energy used by the network, we reroute some of the flows to be able to put links into sleep mode. This means that the remaining active links are more loaded. In Figure 8.8, we look at the link load given by GreenChains for the highest and lowest traffic periods. First, we see that, unsurprisingly, the percentage of links with no traffic is higher when the traffic is low, around 40% of the links for *atlanta* and *germany50*. When the network is at its highest utilization, it drops to around 15% for both networks. The *pdh* network, due to its higher link density, can have more links put into sleep mode. Indeed, between 44% and 71% of the links have no traffic. Moreover, at the lowest traffic period, no links are used at 100%



(a) pdh (b) atlanta (c) germany50

Figure 8.9: Delay in milliseconds for GreenChains for the three network topologies.

for *pdh*, *atlanta* and are at most used up to 57%, 52% of their capacity, respectively. At rush hour, *pdh* and *atlanta* have at most links at 98 and 99% capacity while `germany50` has only one link at full capacity.

Impact on Delay

When some links are put into sleep mode, some of the paths are becoming longer. However, we show in Figure 7.3 that the maximum delay of every path stays below the usual 50 ms latency value in Service Level Agreements: experienced delay is less than 5.4, 10.8 and 16.2 ms on *pdh*, *atlanta* and *germany50* respectively. Moreover, the median of the delay stays constant for *pdh*, *atlanta* at 3.6 and 5.4 ms, respectively. For *germany50*, it only increases from 7.2 for D5 (no link into sleep mode) to 9 ms for D1.

8.6 Conclusion

In this work, we investigate the potential of network virtualization to reduce the energy consumption of networks. We introduce a Column Generation model to solve the problem of minimizing network energy consumption while satisfying the SFC requirements. We also propose GreenChains, an ILP-based heuristic that we validate using our Column Generation model. We then compare three different scenarios corresponding to a continuous deployment of the SDN and NFV paradigm for energy efficiency. We show that compared to a legacy scenario SDN can provide between 18 and 51% energy savings during the night. We also demonstrated that the deployment of VNF in an SDN network leads to additional energy savings between 4 and 12%.

Bibliography

[Gem+16] Aaron Gember, Prathmesh Prabhu, Zainab Ghadiyali, and Aditya Akella. “Toward software-defined middlebox networking”. In: *ACM Workshop on Hot Topics in Networks (HotNets).* 2016, pp. 7–12 (cit. on p. 168).

[LC15] Y. Li and M. Cheng. “Software-Defined Network Function Virtualization: A Survey”. In: *IEEE Access* 3 (2015), pp. 2542–2553 (cit. on pp. 168, 172).

[HB16] J.G. Herrera and J.F. Botero. “Resource Allocation in NFV: A Comprehensive Survey”. In: *IEEE Transactions on Network and Service Management* 13.3 (Mar. 2016), pp. 32–40 (cit. on p. 168).

[Mij+16] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. “Network Function Virtualization: State-of-the-Art and Research Challenges”. In: *IEEE Communications Surveys & Tutorials* 18.1 (2016), pp. 236–262 (cit. on p. 168).

[Mar+15] B. Martini, F. Paganelli, P. Cappanera, S. Turchi, and P. Castoldi. “Latency-aware composition of Virtual Functions in 5G”. In: *IEEE Conference on Network Softwarization (NetSoft).* 2015, pp. 1–6 (cit. on p. 169).

[Rig+15] R. Riggio, A. Bradai, T. Rasheed, J. Schulz-Zander, S. Kuklinski, and T. Ahmed. “Virtual network functions orchestration in wireless networks”. In: *International Conference on Network and Service Management (CNSM).* 2015, pp. 108–116 (cit. on p. 169).

[Gup+15] A. Gupta, M.F. Habib, P. Chowdhury, M. Tornatore, and B. Mukherjee. *Joint virtual network function placement and routing of traffic in operator networks.* Tech. rep. UC Davis, Davis, CA, USA, 2015 (cit. on p. 169).

[Lui+15] M.C. Luizelli, L.R. Bays, L.S. Buriol, M.P. Barcellos, and L.P. Gaspary. "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions". In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2015, pp. 98–106 (cit. on p. 169).

[STV15] Marco Savi, Massimo Tornatore, and Giacomo Verticale. "Impact of processing costs on service chain placement in network functions virtualization". In: *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. IEEE, Nov. 2015 (cit. on pp. 169, 183, 186, 209).

[Bar+16] Md.F. Bari, S.R. Chowdhury, R. Ahmed, R. Boutaba, and O.C.M.B. Duarte. "Orchestrating Virtualized Network Functions". In: *IEEE Transactions on Network and Service Management* PP (2016), pp. 1–1 (cit. on p. 169).

[Moh+15] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K.K. Ramakrishnan, and T. Wood. "Virtual function placement and traffic steering in flexible and dynamic software defined networks". In: *IEEE International Workshop on Local and Metropolitan Area Networks*. 2015, pp. 1–6 (cit. on p. 169).

[Bol+14] Raffaele Bolla, Chiara Lombardo, Roberto Bruschi, and Sergio Mangialardi. "DROPv2: energy efficiency through network function virtualization". In: *IEEE Network* 28.2 (2014), pp. 26–32 (cit. on p. 169).

[Mij15] Rashid Mijumbi. "On the Energy Efficiency Prospects of Network Function Virtualization". In: *CoRR* abs/1512.00215 (2015) (cit. on p. 170).

[Sou+17] Oussama Soualah, Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache. "Energy Efficient Algorithm for VNF Placement and Chaining". In: *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. CCGrid '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 579–588 (cit. on p. 170).

[DW16] A. Dwaraki and T. Wolf. "Adaptive Service-Chain Routing for Virtual Network Functions in Software-Defined Networks". In: *Workshop on Hot topics in Middleboxes and Network Function Virtualization (HotMIddlebox)*. 2016, pp. 32–37 (cit. on p. 173).

[Chv83] V. Chvatal. *Linear Programming.* Freeman, 1983 (cit. on pp. 177, 206).

[GJ79] Michael R Garey and David S Johnson. "A Guide to the Theory of NP-Completeness". In: *WH Freemann, New York* 70 (1979) (cit. on p. 177).

[Cis15] Cisco. *Cisco Visual Networking Index: Forecast and Methodology, 2014–2019.* CISCO. May 2015 (cit. on pp. 183, 186, 209).

[14] *Internet2 network infrastructure topology.* http://www.internet2.edu/media_files/422. 2014 (cit. on p. 187).

[Orl+07] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. "SNDlib 1.0–Survivable Network Design Library". In: *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium.* Apr. 2007, pp. 276–286 (cit. on p. 187).

[Cha+08] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. "Power Awareness in Network Design and Routing". In: *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications.* Apr. 2008 (cit. on p. 199).

[Nic+12] Luca Niccolini, Gianluca Iannaccone, Sylvia Ratnasamy, Jaideep Chandrashekar, and Luigi Rizzo. "Building a power-proportional software router". In: *Proceedings of the 2012 USENIX Conference on Annual Technical Conference.* USENIX ATC'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 8–8 (cit. on p. 199).

[Orl+10] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. "SNDlib 1.0–Survivable Network Design Library". In: *Networks* 55.3 (2010), pp. 276–286 (cit. on p. 209).

[Ara+16] Julio Araujo, Frédéric Giroire, Joanna Moulierac, Yaning Liu, and Modrzejewski Remigiusz. "Energy Efficient Content Distribution". In: *Computer Journal* (2016), pp. 1–18 (cit. on p. 210).

Chapter 9
Conclusion and future research

Being energy aware has become a crucial issue in the recent years. For economic reasons or ecological concerns, reducing its energy footprint has become a growing concern. It is particularly the case for Information and Communication Technologies (ICT) and more precisely for computer networks. On the other hand, the emerging SDN paradigm draw a lot of attention in academia and the industry in last few years. It brings a new way of managing the network by decoupling its brain and muscle(s). The NFV paradigm also brought a new approach to deal with the deployment of services inside a network.

In this thesis, we studied how to leverage SDN and NFV to enable energy efficiency in telecommunication networks. We focus on the challenges brought by these new technologies and how they might impact the deployment of green policies. We first looked at, in Part I, at the constraints imposed by the Ternary Content-Addressable Memory (TCAM) installed in SDN-capable forwarding devices. Combined with the increased complexity of the OpenFlow rules, the table size of SDN switches is quite limited. We propose to use wildcard rules as well as the default rule offered by OpenFlow to reduce the size of forwarding tables and thus formulated the Compression Problem in Chapter 3. We present a variety of solutions (ILPs, greedy heuristic and approximation algorithm), and show that we can, in average, compress routing tables by up to 85% using these aggregation rules. We then studied, in Chapter 4, the Compression Problem in the context of energy efficiency, i.e., the Energy Aware Routing with Compression problem. EAR increases the total number of required forwarding rules because it increases the path's length of the requests. However, we observed better compression ratio due to the decreased number of ports on each switch. We also showed that compression enabled us to obtain energy savings close to the EAR scenario with no limit on the table size. Finally, in Chapter 5, we focus our study on data center networks and provide a testbed implementation of our previous solutions, without energy efficiency. In collaboration with members of the SigNet (I3S) team, we propose Minnie, a controller application for joint compression and routing inside SDN networks. We show that relatively small amount of clients can overload the TCAM tables and that using Minnie, we can deploy up to 3000 clients and manage up to a million flows with only 1000 rules per switches. We also compare the performances of software and hardware implementation of rules and demonstrated that TCAM is still required to obtain good matching performances.

In Part II, we considered the scenario of progressive deployment of SDN-capable devices alongside legacy routers and protocols. This scenario represents the most practical implementation of the paradigm in current networks as network operators are reluctant to upgrade their whole network at once. However, it poses some compatibility issues for green policies as a shutdown SDN-capable device is detected as faulted by legacy protocols such as OSPF. We thus proposed Smooth ENergy Aware Routing (SENAtoR) to solve the problem of Energy Aware Routing in hybrid networks. It is a framework that comprises a routing heuristic combined with the use of backup tunnels, smooth link extinction, and mechanisms for unexpected traffic changes due to failure or flash crowd. We show that SENAtoR can bring the deployment of energy policies closer to reality and that it can quickly react to failure or flash crowd leveraging the SDN paradigm.

In Part III, we shifted our focus on Network Function Virtualization and the Service Function Chaining Provisioning Problem. We proposed, in Chapter 7, a scalable exact model for the problem which can solve all-to-all scenarios with 10000 requests on a 50 node network in a minute, using the column generation algorithm. We also extended the model to the case where the number of replicas of the functions is limited. We showed that using more than 50% of the network as NFV-capable nodes shows little improvement on the bandwidth usage. The same goes for function replicas. We then looked at the Energy Aware Routing with Service Function Chaining, in Chapter 8. We extended the model proposed in Chapter 7 as well as proposed the GreenChains heuristic. We compare the energy savings between a legacy scenario and a hardware scenario (SDN without NFV). We show we can save between 4 and 12% more energy using VNF than compared to using middleboxes.

Perspectives First, we provided in Section 5.3.3 some directions for the *Compression Problem* such as deletion of rules and different workload scenarios. For the SFC problem, we only considered a fraction of the specifications provided by the RFC 7665 [HP15] and we believe that we could integrate constraints such as anti-affinity rules and partial order for chains.

Moreover, we only considered some constraints imposed by the SDN and NFV paradigms. For example, we never consider the controller placement in any of our solutions. There is interest in studying the impact of the controller placement on green policies as it constitutes the core of the paradigm. An aggressive policy that would shutdown too many links might negatively impact the performances of the remaining network devices.

Finally, we could also add two important improvements in the model that we studied. Firstly, we could consider the Quality of Service (QoS) or Quality of Experience (QoE) as a requirement of our solution. While we partly evaluated our solutions by looking at metrics such as link load and end-to-end delay, our model could be improved by adding QoS as a requirement of our solutions. Secondly, we need to consider resiliency and its trade-off with energy efficiency. By nature, both concept are contradictory as EAR relies on reducing the set of active equipment and resiliency usually requires redundancy. These two improvements could apply to SDN and NFV.

Bibliography

[HP15] J. Halpern and C. Pignataro. *Service Function Chaining (SFC) Architecture*. RFC 7665. RFC Editor, Oct. 2015. DOI: 10.17487/RFC7665 (cit. on p. 222).

Appendix A
Study of Repair Protocols for Live Video Streaming Distributed Systems

Contents

A.1 Introduction

We consider in this study *live streaming systems*. In these systems, a source streams a video to a set of clients who want to watch the video in real-time. Streaming video can be done over a classic client/server architecture or a distributed (e.g., peer-to-peer (P2P)) one. Distributed solutions are very efficient for *live streaming* scenarios in which clients watch the video at the same time. The advantage is that the bandwidth of every user can be used to forward the video to other users, lightening the source load.

P2P networks are of two types, with structured or unstructured overlays. In the first type, the nodes are organized according to a (or several) logical tree(s), called *diffusion tree(s).* The source of the video is the root and the video is distributed from the source to the leaves, fathers forwarding the video to their children. In an unstructured overlay, the tree is not explicitly defined: a node having chunks of the video forwards opportunistically these chunks to nodes who miss them. This second type of systems are the most frequently used as they handle very easily *churn*, i.e., the departure and arrival of users, which are very frequent in live video systems. *Frequent churn is the main problem of live distributed streaming system* and the main difference from classical multicast systems. *Structured overlays* have the disadvantage that churn breaks their diffusion trees. However, we have hints that such systems can in fact be very efficient. *If their structure could be maintained by using very simple distributed repair protocols even under frequent churn,* this would allow to keep the advantages of structured overlays, optimal diffusion rate and continuity of the diffusion, while being resistant to churn.

Goal of the study. Our goal is to propose simple distributed repair mechanisms for *structured* live streaming systems. To understand such systems, we then want to develop formal models, which can be efficiently simulated. Last, we aim at proving that they can be very efficient in practice (and potentially more efficient than unstructured systems).

Contributions. In this work, we study a structured network for live video streaming experiencing frequent node departure and arrivals.

- We propose different repair protocols for the diffusion tree using different amount of information in Section A.2.2.

- We show that a system using these protocols can be, first, formally analyzed and, second, efficiently simulated. We provide estimation of different system metrics, e.g., bandwidth usage, delay, or number of interruptions of the streaming, via simulations, as well as analysis.

- We provide analytical formulas of the system's metrics in Section A.3.

- We developed a discrete-event simulator, presented in Section A.4.1. We used it to compare the different repair protocols. The results are presented in Section A.4.

- We present first evidences that, by using simple distributed repair protocols, structured live streaming systems can be very resistant to churn.

A.1.1 Related Work.

Structured versus Unstructured Systems. There are two major classes of P2P live video streaming architectures, the first one being named either *unstructured*, *mesh-based*, *gossiping* or *torrent-like*; the second named either *structured* or *tree-based*, see for example [Zha+05a] or [Li+13a] for a classification. Note that this distinction is not strict and that mixed systems were also proposed, e.g. [WXL10].

Early systems, like [CRZ00], influenced by IP multicast, attempted at constructing a multicast tree to stream the media. To avoid the shortcomings such as resistance to churn and low bandwidth usage, this simple idea has evolved into elaborate algorithms like Splitstream, proposed in [Cas+03a] or ZIGZAG, in [THD03]. The signature of this group of systems is an active maintenance of an overlay structure that clearly defines the data flow, thus the name *structured overlays*. SpreadIt, proposed in [DBG01], is the closest work to ours. It considers multiple protocols for handling arrivals and departures of peers in the network but presents few empirical results and no analysis of the system.

On the other hand, we have systems inspired by BitTorrent, one of the best-known peer-to-peer protocols, described in [Coh03]. The core idea of this class of overlays is organizing the peers into a random, highly-connected graph and disseminating the data using a simple, probabilistic algorithm. The first instance of an unstructured system was introduced in [Ban+03] as a way of enhancing a single-tree overlay. It was then the base for the first real peer-to-peer network that streamed video to a big number of simultaneous clients [Zha+05a]. The characteristic of this group of networks is that they do not have an explicit overlay structure for the data flow, thus the name *unstructured overlays*.

Unstructured systems are widely regarded the better choice. That is often explained by the complexity of making a structured system reliable. However, we show in this study that reliability can be ensured, for a simple system, efficiently by a simple algorithm.

Analysis of Structured Systems. The existing analysis of these systems focus on the feasibility, construction time and properties of the established overlay network, see for example [Cas+03a; VYF06] and [DFC07] for a the-

Variable	Signification	Default value
n	Number of nodes of the tree, root not included	1022
d	Node bandwidth (or ideal node degree)	2
h	Height of the tree (root is at level 1)	10
μ	Repair rate (avg. operation time: 100 ms)	1
λ	Individual churn rate (avg. time in the system: 10 min)	$\frac{1}{6000}$
Λ	System churn rate ($\Lambda = \lambda n$)	$\frac{1022}{6000} \approx 0.17$

Terminology	Values
unit of time	100 ms
systems with low churn	$\Lambda \in [0, 0.4]$
systems with high churn	$\Lambda \in [0.4, 1]$

Table A.1: Summary of the main variables and terminologies used in this work.

oretical analysis. But these works usually do not consider over the issue of tree maintenance. Generally, in these works, when some elements of the networks fail, the nodes disconnected from the root execute the same procedure as for initial connection. This is not the case in our study. To the best of our knowledge, there are no theoretical analysis, except [Gir+13a], on the efficiency of tree maintenance in streaming systems, reliability is estimated by simulations or experiments as in [Cas+03a].

In [Gir+13a], the authors propose an efficient maintenance scheme for trees. The distributed algorithm ensures that the tree fastly recovers to a "good shape" after one or multiple failures occur. The authors give analytic upper bounds of the convergence time. This paper is a starting point of our study. We introduce new repair algorithms and then provide an average case analysis of these protocols.

A.2 Distributed Protocols and Modeling

A.2.1 Modeling

We consider a system, a source streaming a live video, and n nodes which want to watch the video. A summary of the variables used in this work is given in Table A.1. This source is the single reliable node of the network, all

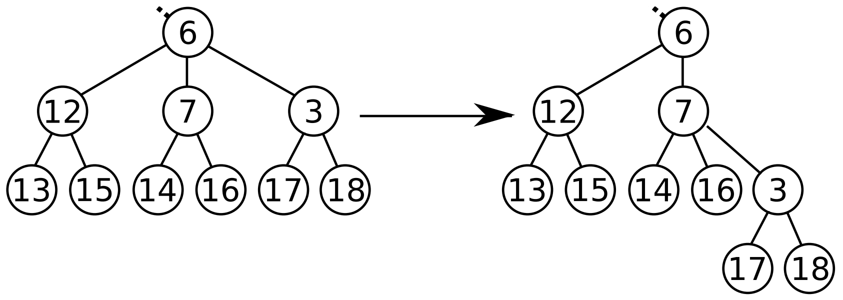


Figure A.1: Example of a *push* operation carried out by the overloaded Node 6.

other peers may be subject to failure. Nodes are organized according to a tree of size $n+1$. The source is the root of the tree. Nodes forward the video to their children in the tree. Each node has a given bandwidth allowing him to serve a given number of other nodes d. In this study, we consider that all nodes have the same bandwidth 2. A node is said to be *overloaded*, when it has more than d children. In this case, he cannot serve all its children and some of them do not receive the video. Note that the delay between broadcasting a piece of media by the source and receiving by a peer is given by its distance from the root in the logical tree. Hence our goal is to minimize the tree depth, while following degree constraints.

Each node applies the following algorithm without the knowledge of the whole network.

- When a node is *overloaded*, it carries out a *push* operation. It selects two of its children, and the first one is reattached to the second one, becoming a grandchild. Figure A.1 presents an example of such operation.

- When a *node leaves* the system, one of its child is selected to replace him. It reattaches to its grandfather. The other children reattach to it. An example is given in Figure A.2. In this work, we only consider single failure but multiple failures could be handled by considering the great grandfather of a node or by reattaching to the root.

- When a *new node arrives*, it is attached to the root.

Churn. We model the system churn rate with a Poisson model of rate Λ. A node departure (also called *churn event*) occurs after an exponential time

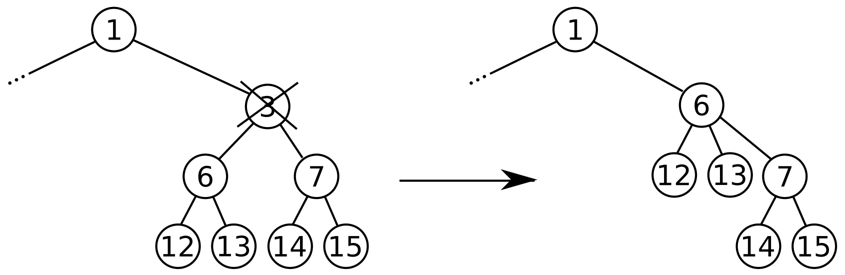


Figure A.2: Example of the operation made after the departure of Node 3.

of parameter Λ, that is in average after a time $1/\Lambda$. We note the individual failure rate $\lambda = \Lambda/n$. In this work, we study scenarios with constant size population. Thus, when a node leaves the system, we consider that a new node appears. Authors in [H+06; Vu+07] carried out a measurement campaign of a large-scale overlay for multimedia streaming, PPLive [H+06]. Among other statistics, the authors report that the median time a user stays in such a system is around 10 minutes. In this study, we use this value as the default value (after normalization see the following on default values).

Repair rate. When a node has a *push* operation to carry out, it has first to change its children list, and then to contact its two children implicated in the operation so that they also change their neighborhood (father, grandfather or children). This communication takes some amount of time, that can vary depending on the node to contact and congestion inside the network. To take into account this variation, we model the repair time as a random variable with exponential distribution of parameter μ. [Li+08] reports that the communication time in a streaming system is in average 79 ms. Thus, we believe that assuming an average repair time of 100 ms is appropriate.

Default values. In the following, for the ease of reading, we normalize the repair rate to 1. We call *unit of time* the average repair time, 100 ms. The normalized default churn rate, λ is thus $1/6000$ and the system churn rate is $\Lambda = n\lambda \approx 0.17$. These default values are indicated as typical examples, but in our experiments we present results for a range of values of Λ between 0 and 1. We talk of *low churn systems* for values of Λ below 0.4, and of *high churn* systems for values above 0.4.

A.2.2 Description of the Protocols

We now define *four different protocols* according to four different ways to select the children during the operations and four different *levels of knowledge* of the streaming system. We will study the trade-off between knowledge and performance in the following.

Smallest Subtree Protocol (SSP)

. In this protocol, each node knows the subtree size of each of its sons. When a churn occurs, the son with the largest subtree of the failing node takes over the role of its father by adopting every of its sibling. It is itself adopted by its grandfather. When a node is overloaded, its son with the third largest subtree is pushed into the son with the second largest one.

Random Protocol (RP)

. In this protocol, nodes do not keep information about their subtree sizes. They store their children in a queue and each new node attached to them is put at the end of it. A node receiving the video only gives it to the two children at the start of its queue. When a churn occurs, the eldest son takes over the role of its father by adopting every of its sibling. It is adopted by its grandfather. There is no interruption in the video transmission between the son and the father. The order in the grandfather children is conserved (i.e. If the father was the first child of its parent, the son takes the first place in the grandparent sons). Every of its sibling is reattached to the new father and thus is at the end of the queue, in the same order as it was in the failing node queue. An overloaded node chooses at random the son that will be pushed and also the node that will receive the pushed node.

No Interruption Protocol (NIP)

This protocol shares some similarity with the random protocol. The same operation is done when a node leaves the system: the eldest child takes over the role of the failing node. When an overloaded carries out a push operation, it chooses uniformly at random a node *not receiving* the video and pushes it under a node *receiving* the video. This ensures no interruption of the video distribution.

Partial Information Protocol (PIP)

The idea of the protocol starts with the observation that for most operations, it is relatively easy to determine the largest subtrees *without storing the exact size of all subtrees.* When a new node arrives, we know that it has a subtree size of one and that it has to be pushed to the bottom of the tree. So we will label it as *new*. Then, during this arrival process, we will have to decide in which subtree to push it. If we do the hypothesis that the tree stays relatively well balanced during the protocol lifetime, we may choose at random without adding too much imbalance in the tree. When there is a churn, we know that one of the children replaces its father. It will have three children, one large (its former brother), that we label *big*, and two smaller ones, that we label *normal*. Hence, it has to push one of the smaller into the second one. Again, if we suppose that the tree is relatively balanced, this choice can be made randomly. The subtree pushed is relabeled as *big*, its new brothers are labeled as *normal*, and we remove the label of its former brothers.

A.2.3 Metrics

To evaluate the performance of the different protocols, we are interested by the following metrics.

Time to attach a new node. When a new node arrives into the system, it has to attach to a node which has enough bandwidth to forward the video to it. Basically, it is first attached to the root and then pushed to the bottom of the tree to become a leaf. Ideally, this takes a small amount of time, logarithmic in the number of nodes watching the video, n, see Section A.3.

Repair time after a node departure. When a node leaves the system, a repair processed starts as described in Section A.2.1. Basically, one of its son replaces it and some nodes are pushed in its subtree. The simulator records this repair time at each churn event.

Number of people not receiving the video. Due to these two phenomenas, attachment of a new node and repairs, some people do not receive the video during small periods of time during the life of the protocols. We study what fraction of the nodes do not receive the video and during which amount of time.

Height of the tree or delay. The height of the diffusion tree gives the maximum delay between the source and a node. Ideally, it is equal to $\lfloor \log_d n \rfloor + 1$, where d is the maximum node degree.

Number of interruptions and interruption duration. We monitor the number of interruptions of the video diffusion to a node during the protocol lifetime, as well as the distribution of interruption durations.

A.3 Analysis

In this section, we analyze the Smallest Subtree Protocol. We give analytical formulas to estimate different metrics: the repair time, the average number of people not receiving the video, and the number of interruptions.

The analysis is done under an *independence hypothesis* of the failures, meaning that the repair of one failure is done before another happens. Remark that it implies that the diffusion tree is always a balanced binary tree when a failure happens. Indeed, we are considering a system with constant population in which a new node arrives when a node leaves. Using the information about subtree sizes, SSP will push this new node in the optimal position. We will consider here complete binary trees of size $n = 2^h - 2$, but the analysis extends to other values. The independence hypothesis is evaluated by simulation for different values of churn in Section A.4.2.

Analysis of the repair time. We study the repair time of the diffusion tree when a node selected at random leaves the system. We consider the position of the node in the tree.

If the failing node is a leaf, the tree is still balanced and nothing happens.

If the failure happens at depth $i < h$, the tree is repaired (that is, all nodes have degrees less or equal to 2) after $h - i - 1$ push operations. Note that when $i = h - 1$, no pushes are needed.

We give now the average time to repair the tree after a node departure. In average, a node carries out an operation in a time $\frac{1}{\mu}$. Hence, the average time to carry out $h-i-1$ operations is $\frac{h-i-1}{\mu}$. The number of nodes at depth i, $2 \leq i \leq h$, is 2^{i-1}. Thus, the probability that the node leaving the system is at depth i is $\frac{2^{i-1}}{n}$. Recall than $n = 2^h - 2$. If we note T the time to repair the diffusion tree, we have

$$\begin{aligned}\mathbb{E}[T] &= \tfrac{1}{2^h-2}\textstyle\sum_{i=2}^{h-2} 2^{i-1}\tfrac{h-i-1}{\mu} = \tfrac{2^{h-1}-2(h-1)}{\mu(2^h-2)} \\ &\sim \tfrac{1}{2\mu} \text{ when } h \text{ is large.}\end{aligned}$$

The Smallest Subtree Protocol is very efficient. Indeed, only one half operation is needed in average to repair the tree after a node departure.

Analysis of the number of nodes not receiving the video. During the repair process, some nodes do not receive the video. Indeed, recall that only the two biggest subtrees of an overloaded node receive the video. When a node at level i leaves the system, its child with the largest subtree replaces it. It becomes overloaded with three children of subtree sizes $2^{h-i}-1$, $2^{h-i-1}-1$, and $2^{h-i-1}-1$ (A node at depth i is of height $h-i+1$. A subtree of height i contains 2^i-1 nodes). Thus, $2^{h-i-1}-1$ nodes do not receive the video. At each repair, the height of the tree is reduced by one and this number is divided by around two. That is after k repairs, it is equal to $2^{h-i-1-k}-1$, see Figure A.3. A push operation is done in average in a time $\frac{1}{\mu}$. Hence, if we note n_i the number of people without video when the failing node is at level i, we have

$$\mathbb{E}[n_i] = \frac{\Lambda}{\mu}\sum_{k=0}^{h-i-1}(2^k-1) = \frac{\Lambda}{\mu}\left(2^{h-i}-h+i-1\right).$$

Recall now that the probability that the failing node is at level i is $\frac{2^{i-1}}{2^h-2}$. If we note n_{bad}, the number of people without video, we have

$$\begin{aligned}\mathbb{E}[n_{bad}] &= \textstyle\sum_{i=2}^{h-2}\tfrac{2^{i-1}}{2^h-2}n_i = \tfrac{2^{h-1}(h-5)+2(h+1)}{2^h-2}\tfrac{\Lambda}{\mu} \\ &\sim \tfrac{h-5}{2}\tfrac{\Lambda}{\mu} \text{ when } h \text{ is large.}\end{aligned}$$

Analysis of the number of interruptions. Recall that when a node leaves the system at level i, its first child is overloaded with three children of subtree sizes $2^{h-i}-1$, $2^{h-i-1}-1$, and $2^{h-i-1}-1$. Only the two biggest sons of a node receive the video. Thus, $2^{h-i-1}-1$ nodes are interrupted. With a repair, the interruption for these nodes stops and the height of the tree interrupted is reduced by one, that is after k repairs, it is equal to $2^{h-i-1-k}-1$. Every interrupted node is only interrupted once. Hence, if we note $int_i(t)$ the number of interruption when the failing node is at level i as

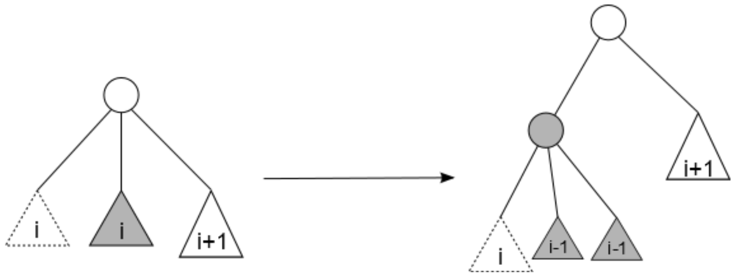


Figure A.3: Effect of a push operation during a repair.

a function of the time, we have

$$E[int_i(t)] = t\Lambda \sum_{k=0}^{h-i-1} 2^k - 1 = t\Lambda(2^{h-i} - h + i - 1)$$

Recall now that the probability that the failing node is at level i is $\frac{2^{i-1}}{2^h-2}$. If we note $n_{int}(t)$, the average number of interruptions, we have

$$\begin{aligned} \mathbb{E}[n_{int}(t)] &= \sum_{i=2}^{h-2} \frac{2^{i-1}}{2^h-2} int_i = \frac{2^{h-1}(h-5)+2(h+1)}{2^h-2} t\Lambda \\ &\sim \frac{h-5}{2} t\Lambda \text{ when } h \text{ is large.} \end{aligned}$$

Analysis of the arrival time. When a new node arrives in the system, it is first attached to the root. It does not receive the video until it is then pushed to the bottom of the tree by $h-2$ successive push operations. Since a node carries out an operation in an average time of $\frac{1}{\mu}$, if we node T the time for the new node to receive the video, we have:

$$\mathbb{E}[T] = \frac{h-2}{\mu}.$$

A.4 Simulations

We developed a discrete-event simulator of the streaming system described in Section A.4.1. We use it to analyze and compare the different protocols. A summary of results is provided in this section for different metrics.

A.4.1 The simulator

Our desire to focus on high level simulation led us to develop a custom C++ discrete-event simulator to evaluate metrics of the system described

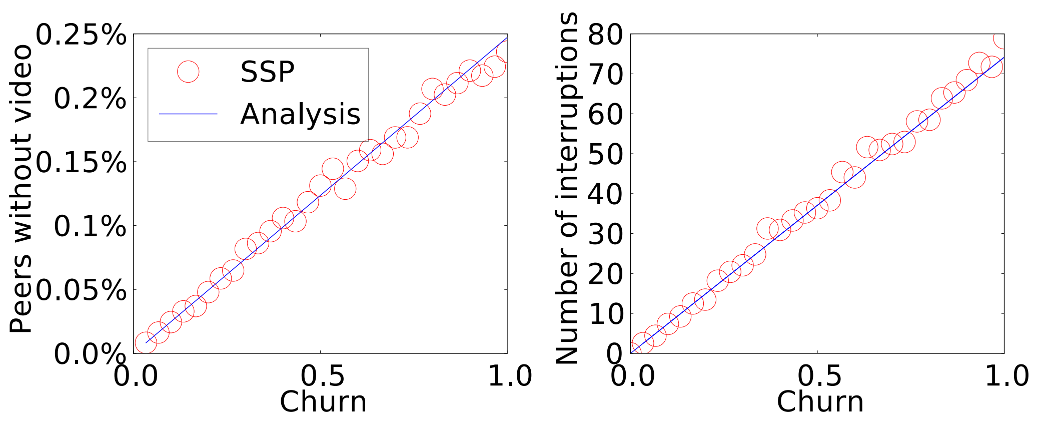


Figure A.4: Comparison between the formal model of Section A.3 and the simulations for: (Left) Average fraction of peers not receiving the video as a function of the churn rate due to the repair (1022 nodes). (Right) Average number of interruptions per node (1022 nodes) for a time of 30,000 units of time.

in Section A.2.3. We did not use low level network simulators like NS-2 or OMNET because they would require more computation time and give metrics non pertinent to our analysis. Our goal is to focus on the tree structure after churns and reparations to validate our protocols.

A.4.2 Validation of the analytical model

To validate our analysis, we first compared the results given by the simulator to the analytical formulas obtained in Section A.3.

People without the video. In Figure A.4 (Left) is given the average fraction of people not receiving the video. New nodes joining the systems are not taken into account until they start to receive the video. Only the non distribution of video due to the repairs is counted. As an example, for the default churn rate value, 0.17, only 0.03% of the nodes do not receive the video in average, and for a high value of churn 1, only 0.25%.

We see that the closed formula models very closely the system. Recall that the formulas were given for low churn, that is when a churn event is repaired before another churn event. This corresponds to values of the churn rate Λ between 0 and 0.5. However, we see that, even for larger churn values (> 0.5), the system behavior is well predicted by the formula.

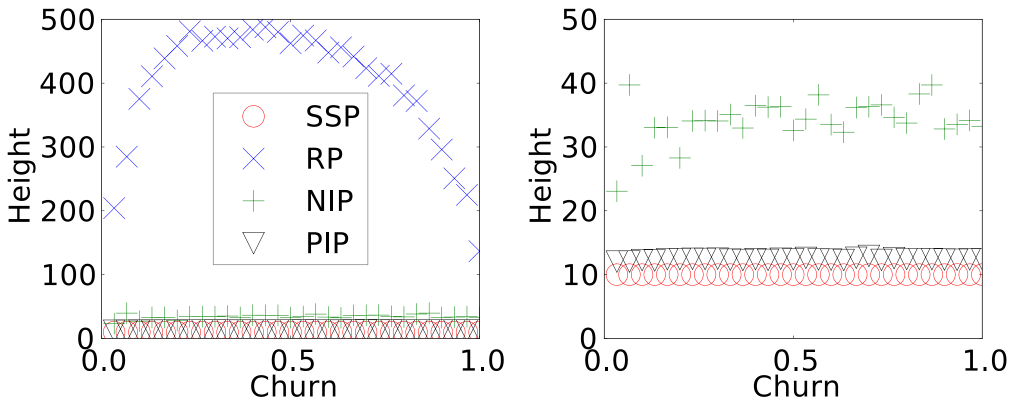


Figure A.5: Average height (over time) of the tree (1022 nodes) as a function of the churn rate for the four repair protocols. Left: y-axis range from 0 to 500. Right: y-axis range from 0 to 50.

Interruptions. The average number of video interruptions is given in Figure A.4 (Right) as a function of the churn rate. Again, the interruptions are due to the repair of the diffusion tree when there is a node departure.

This number is the average over all nodes of the number of interruptions per node for a period of 30,000 units of time. For example, for a value of churn of 0.2, the diffusion for a node is interrupted 20 times in average, that is, a node experiences an interruption every 1500 units of time, that is every 150 seconds. As we will see it later, the durations of the interruptions are very short. Thus, they are without consequence for the end-user experience.

We see that the analytical formula gives a very good estimation of the behavior of the system. This is true even for high churn.

A.4.3 Comparison of the protocols

In this section, we compare by simulation the performances of the different protocols that we propose. Recall that these different protocols use different levels of information. Several providers of live video may have different criteria on the different metrics and on the implementation, leading to different choices of the adequate protocol.

Levels of needed information. We recall that, for every protocol, each node has at least to store the ids (and addresses) of children, father and grandfather. In RP and NIP, each node stores the order in which its children attached to it. In SSP, each node needs to know the size of its subtree. Last, in PIP, only two additional bits are necessary: a new node arriving in the system is tagged as *new* till it finds a place in the diffusion tree and a node pushed by a repair process is tagged as *big*. For the detailed discussion, the reader may refer itself to Section A.2.2.

We now compare the protocols for the different metrics. As we will see, the protocols do not behave at all similarly for some metrics. *To improve the readability of the figures, for each metric, we propose two plots with the same data, but with different scales of the y-axis.*

Height of the diffusion tree and delay. Figure A.5 shows the average height (over time) of the tree as a function of the churn rate.

The first observation is that RP exhibits a very different behavior than the other protocols, see Figure A.5 (Left). It behaves very badly: the height of the diffusion reaches a value of 500. It is not of order logarithmic in n, the number of nodes. On the contrary, it gets close to a linear height! The diffusion tree looks like a path in this case. The protocol is very inefficient. Thus, pushing the nodes randomly is not possible in practice.

We are now interested by the three other protocols (Figure A.5 (Right)). They behave a lot better. First note that the average height does not depend on the churn rate. Thus, the difference between the three protocols only is the value of the height:

- SSP has a constant height of 10 which is exactly $\lfloor \log_d n \rfloor + 1$ for $n = 1022$. The protocol is optimal for the maximum height, and thus delay, of a node.

- Partial Information Protocol has an average height around 12.3. The height is not very far from the optimal 10.

- No Interruption Protocol behaves worstly. The average weight is around 33. It is nevertheless a lot better than the completely random protocol RP, and could still be used in practice.

The explanation of the different protocols' efficiency is due to their different behaviors, (1) when there is a churn event, and, (2) when there is an arrival of a new node.

First, when a failing node is close to the root, a large subtree does not receive the video anymore and has to be pushed in the tree by the repair process. The SSP protocol succeeds to reconstruct a perfectly balanced tree

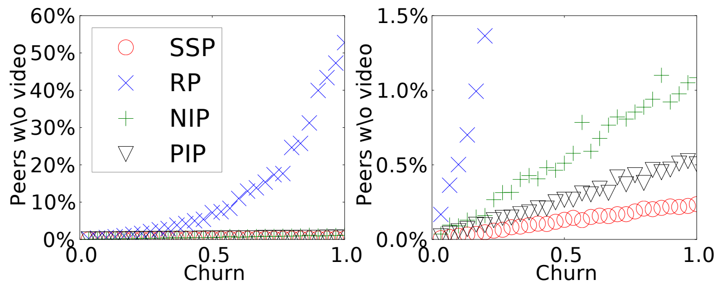


Figure A.6: Average fraction of peers not receiving the video as a function of the churn rate (1022 nodes). Left: y-axis range from 0 to 60%. Right: y-axis range from 0 to 1.5%.

by using the information about the subtree sizes. It knows exactly what is the right node to push. The two protocols RP and NIP do not have this information: they blindly push the subtree. It can thus happen that a large subtree ends up at the bottom of the tree at the end of the repair process. This can increase its height significantly: in the worst case, the height can be doubled with only one churn event. We see that PIP has performance not too far from optimal. We see that, even without the information of the subtree sizes, a simple guess of the node to be pushed is efficient in terms of delay. The protocols recognizes large subtrees (of size 2^i) from small subtrees (of size 2^{i-1}), leading to a near optimal repair process.

Second, when there is an arrival of a new node in the tree. SSP can push this new node exactly at the right position of the diffusion tree. PIP and NIP cannot distinguish two subtrees with a small difference of size and thus push the new node randomly to the bottom of the diffusion tree. RP has in this case a very bad behavior. As it does not have any information, it does not know that a node is new. Hence, it pushes randomly a node that can have a very large subtree, instead of the new node of subtree size one! On the contrary, NIP does not push subtrees receiving the video to ensure a continuity of the video diffusion. Hence, the arrival of a new node in the system cannot trigger that a large subtree is pushed at the bottom of the tree.

To sum up, the repair protocol SSP *is optimal in terms of tree height and, thus, of delay. This protocol uses information about node subtree sizes. If this information is considered too costly to maintain by an operator, it can obtain close to optimal performances with* PIP, *which uses a very small*

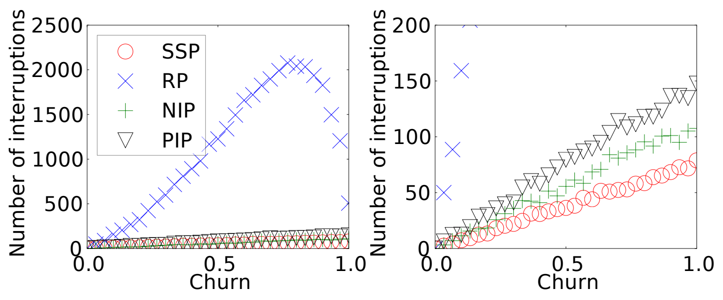


Figure A.7: Average number of interruptions of the streaming per node as a function of the churn rate after 10,000 units of time. Left: y-axis range from 0 to 2500. Right: y-axis range from 0 to 200.

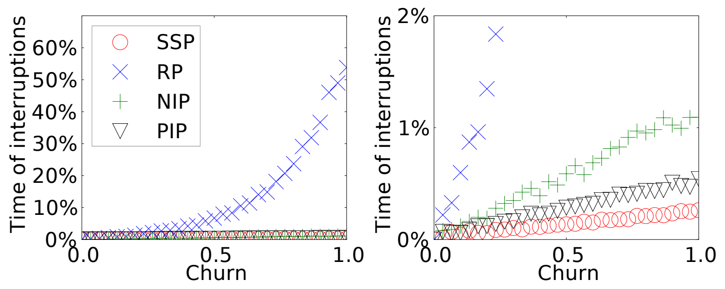


Figure A.8: Average fraction of time for which the streaming was interrupted as a function of the churn rate (after 30,000 units of time for 1022 nodes).

amount of information (two node labels).

Percentage of people without the video during time. Figure A.6 shows the average percentage of nodes that do not receive video as a function of the churn rate.

We see again, in Figure A.6 (Left), that RP behaves a lot worse than the other three protocols. As much as 52% of the nodes do not get the video for a churn rate of 1. It already reaches 6% for a churn rate of 0.5. This protocol is thus very inefficient.

The three other protocols, SSP, NIP and PIP, behave similarly, and are very efficient. The average percentage of people without the video is very small for churn values expected in a viable live streaming system (churn rate between 0 and 0.4). For a churn rate of 0.2, the average percentage is respectively 0.04%, 0.1% and 0.15 % of the nodes for the three protocols, see Figure A.6 (Right).

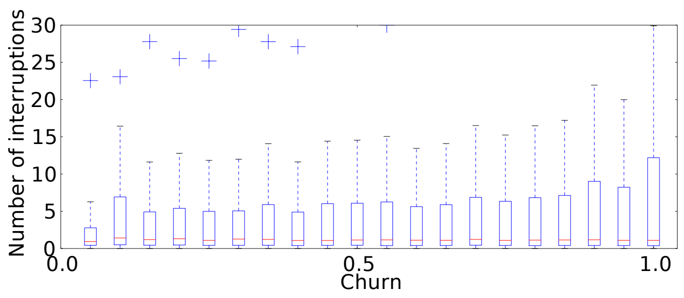


Figure A.9: Distribution of the durations for different churn values (1022 nodes over 30,000 units of time). Boxplots give: median value (red line), first and third quartile (box), maximum value (blue cross).

Number of interruptions during the diffusion. We studied the number and durations of interruptions of the video diffusion process for a node in the system. Indeed, the diffusion to a node can be interrupted after the departure of one of its ancestors. We report in Figure A.7 the average number of interruptions for a node present in the system during 30,000 units of time (3,000 seconds for the default values), as a function of the churn rate. Again, we see in Figure A.7 (Left) that RP behaves very badly with a peak of 2,000 interruptions for a churn of 0.8. This represents an interruption every 15 units of time, that is every 1.5 second. This protocol is not viable. On the contrary, we see in Figure A.7 (Right) that SSP, NIP and PIP behaves very well and similarly. Even for a high value of churn like 1, the number of interruptions per node is at most 150, representing an interruption every 200 units of times. For a low churn, e.g., $\Lambda = 0.1$, the number of interruptions is close to 10, that is an interruption every 3,000 units of time, that is 5 minutes. We note that NIP behaves better than PIP for this metric, when it is behaving worse for the other metric. The explanation is that the NIP was specially designed to avoid the interruption of the video diffusion. During the repair process, the two nodes receiving the video are never pushed, even if they have a smaller subtree than a node not receiving the video. This is not the case for SSP, PIP and RP.

We plot in Figure A.8, the average fraction of time of node was interrupted during the simulation, that during 30,000 units of time. We see that for a value of churn of 0.1, a node is interrupted in average for 0.15% of the time only.

We plot in Figure A.9 the distribution of the duration of an interruption for Smallest Subtree Protocol. We see that the median time is 1. More than half of the interruptions lasts 1 unit of time. The value of the third quartile is less than 5 units of time for almost all values of churn rates. The maximal interruption lasts less than 30 units of time for a system with low churn (churn rate between 0 and 0.4).

To summarize, in a system in which nodes stay on average 15 minutes (churn rate $\Lambda = 0.1$*), a node watching a video for one hour will experience 12 interruptions of median duration 100 ms, few interruptions of duration 500 ms, and if it is not lucky, one interruption of 2.5 seconds. A buffer of few seconds (e.g., 10s) of video will make these interruptions imperceptible to the end-users. For a video rate of 480 kbps, it corresponds to a buffer size of only 40MB.*

A.5 Conclusion, Current and Future Work

In this work, we study a live video streaming system via formal analysis and simulation. We show that, using a simple repair protocol, a *structured* peer-to-peer system can be very efficient. The diffusion tree can be maintained thanks to independent distributed operations of the nodes. This leads to well-balanced diffusion trees, with almost optimal (logarithmic) distance to the source. We additionally show that the diffusion of the video is interrupted only for very short durations of times, imperceptible by an end user.

We are currently investigating analytical models of the streaming system. In particular, the closed formulas are given for Smallest Subtree Protocol. We wish to obtain models for the other protocols. We are also working on models for higher churn rates. For these rates, the source becomes a bottleneck, as new peers attach to it. We can estimate repair times by modeling the number attached to the source as a Markovian queuing system.

Appendix B

The Structured Way of Dealing with Heterogeneous Live Streaming Systems

Contents

B.1 Introduction

In a live streaming scenario, a source streams a video to a set of clients, who want to watch it in real-time. Live streaming can be done either over a classic client-server architecture or over a distributed one. The high bandwidth required by a large number of clients watching the stream at the same time may saturate the source of a classic centralized architecture. In a distributed scenario (e.g., P2P), the bandwidth required can be spread among the users and the bottleneck at the source can be reduced. So, peer-to-peer systems are cheap to operate and scale well with respect to the centralized ones. Because of this, the P2P technology is an appealing paradigm for providing live streaming over the Internet.

In P2P context, two families of systems exist: structured or unstructured overlay networks. In structured overlay networks, peers are organized in a static structure with the source at the root of a diffusion tree. A node receives data from a parent node that can be a peer or the source of the stream. In unstructured overlay networks, peers self organize themselves without a defined topology. Unlike the structured case, a peer can receive each piece of the video from a different peer. The main challenge for P2P systems, compared to a classical multicast system, is to *handle churn*, that is the frequent arrival and departure of peers. In unstructured systems, the diffusion is done opportunistically and, as the authors show e.g. in [Bon+08], this ensures efficiency and robustness to the dynamicity of peers. Nevertheless, in this kind of systems, the control overhead may have a negative impact on the performance: network links and peer states need to be constantly monitored. In structured overlay, the content distribution is easier to manage. But, the departure of users may break the diffusion tree. Our goal is to too show, by proposing *simple distributed repair mechanisms of the diffusion tree*, that *structured systems may be robust to churn*, while keeping their advantages, an optimal diffusion rate and the continuity of diffusion.

Contributions. In this work, we study a structured network for live video streaming experiencing frequent node departures and arrivals in systems where nodes have heterogeneous upload bandwidth.

- We propose, in Section B.4, simple distributed repair protocols to rebuild the diffusion tree, when peers are leaving. Different protocols use different levels of information.

- We compare the protocols using different metrics, i.e., delay, percentage of clients without video, number and duration of interruptions, see Section B.5. We validate the protocols using different peer bandwidth distributions from literature.

- We study the efficiency of the protocols versus the level of information they use. We show that a repair protocol can be very efficient, even when using only a small amount of local information on its neighbors, see Sections B.5.2.

- We use real Twitch traces to compare our different heterogeneous protocols in a real-life scenario of a streaming session in Section B.5.3. We show that our simple distributed repair protocols work very well in practice.

- Finally, in Section B.5.4 we compare the protocols in a setting where exact information is not always available.

B.2 Related Works

Structured vs unstructured systems. A general overview of P2P based live streaming services can be found in [Li+13b]. As stated, there are two main categories of distributed systems for video live streaming: unstructured and structured, even if hybrid solutions are also possible [WXL07]. In unstructured overlay networks, peers self organize themselves in an overlay network, that does not have a defined topology. CoolStreaming/DONet [Zha+05b] is an implementation of this approach. In structured overlay networks, peers are organized in a static structure with the source at the root of the tree. There are many techniques used in P2P live streaming with single-source and structured topology. These techniques may fall into two categories: single-tree and multiple-tree. In the single-tree approach each node is connected to a small number of nodes to which it is responsible for providing the data stream. ZIGZAG [THD03] is an example of this approach. In the multiple-tree approach the main idea is to stripe the content across a forest of multicast trees where each node is a leaf in every tree except one, as done by SplitStream [Cas+03b].

Reliability In terms of reliability, unstructured systems are considered the best choice. As shown in [Bon+08] this kind of systems is able to handle churn (peers leaving the system) is a very efficient way. That is often explained by the complexity of making a structured system reliable. However, we show in this study that reliability can be ensured, for a simple system, efficiently by a simple algorithm.

In existing studies of structured systems, see e.g. [Cas+03b; VYF06] and [DFC07], authors focus on the feasibility, construction time and characteristics of the established overlay network. But these works usually do not

Variable	Signification	Default value
n	Number of nodes of the tree, root not included	1022
d	Node bandwidth (or ideal node degree)	-
h	Height of the tree (root is at level 1)	-
μ	Repair rate (avg. operation time: 100 ms)	1
λ	Individual churn rate (avg. time in the system: 10 min)	$\frac{1}{6000}$
Λ	System churn rate ($\Lambda = \lambda n$)	$\frac{1022}{6000} \approx 0.17$

Terminology	Values
unit of time	100 ms
systems with low churn	$\Lambda \in [0, 0.4]$
systems with high churn	$\Lambda \in [0.4, 1]$

Table B.1: Summary of the main variables and terminologies used in this work.

consider over the issue of tree maintenance. Only few works focus on it. In [Gir+13b; Gir+17] the authors propose a simple distributed repair algorithm that allows to fast recover and obtain a balanced tree after one or multiple failures. In [GH15] the authors develop a simple repair and distributed protocol based on a structured overlay network. By providing, through analysis and simulations, estimations of different system metrics like bandwidth usage, delay and number of interruptions of the streaming, they show that a structured live streaming systems can be very efficient and resistant to churn. However, their study is based on the fact that all nodes have the same bandwidth, where the bandwidth determines the maximum number of other nodes that can be served simultaneously. We extend their work to the case of peers with heterogeneous bandwidth.

B.3 Distributed Systems and Modeling

B.3.1 Modeling

We model a live streaming system as a tree of size $n + 1$ where the root represents a source streaming a live video. A summary of the variables used in this work is given in Table B.1. The n other nodes are clients wanting to watch the video. The source is the only reliable node of the network, all

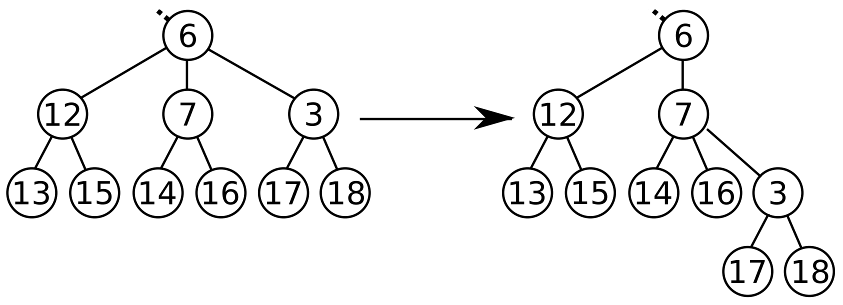


Figure B.1: Example of a *push* operation carried out by the overloaded Node 6.

other peers may be subject to failure. A node v has a limited bandwidth $d(v)$ used to serve its children. A node v is said to be *overloaded*, when it has more than $d(v)$ children. In this case, it cannot serve all its children and some of them do not receive the video. Note that the delay between the broadcast of a piece of media by the source and the reception by a peer is given by its distance from the root in the logical tree. Hence our goal is to minimize the tree depth, while respecting degree constraints.

Each node applies the following algorithm with a limited knowledge of the whole network.

- When a node is *overloaded*, it carries out a *push* operation. It selects two of its children, and the first one is reattached to the second one, becoming a grandchild. Figure B.1 presents an example of such operation.

- When a *node leaves* the system, one of its child is selected to replace him. The other children reattach to it. An example is given in Figure B.2. In this work, we only consider single failure. But multiple failures could be handled by considering the great grandfather of a node or by reattaching to the root.

- When a *new node arrives*, it is attached to the root.

Churn. We model the system churn rate with a Poisson model of rate Λ. A node departure (also called *churn event*) occurs after an exponential time of parameter Λ, that is in average after a time $1/\Lambda$. We note the individual failure rate $\lambda = \Lambda/n$. Authors in [H+06; Vu+07] carried out a measurement campaign of a large-scale overlay for multimedia streaming, PPLive [H+06].

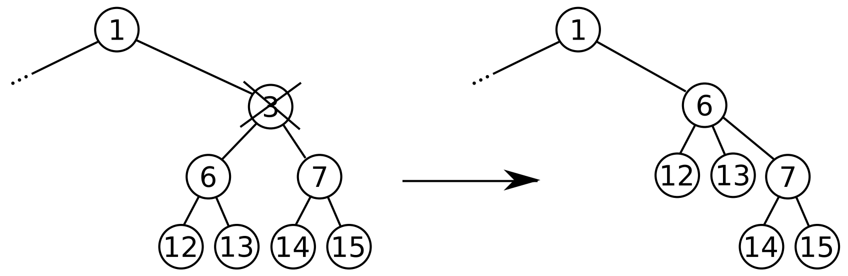


Figure B.2: Example of the operation made after the departure of Node 3.

Among other statistics, the authors report that the median time a user stays in such a system is around 10 minutes. In this study, we use this value as the default value.

Repair rate. When a node has a *push* operation to carry out, it has first to change its children list, and then to contact its two children implicated in the operation so that they also change their neighborhoods (father, grandfather or children). This communication takes some amount of time, that can vary depending on the node to contact and congestion inside the network. To take into account this variation, we model the repair time as a random variable with exponential distribution of parameter μ. [Li+08] reports that the communication time in a streaming system is in average 79 ms. Thus, we believe that assuming an average repair time of 100 ms is appropriate.

Default values. In the following, for the ease of reading, we normalize the repair rate to 1. We call *unit of time* the average repair time, 100 ms. The normalized default churn rate, for an average stay of 10 minutes, λ is thus 1/6000 and the system churn rate is $\Lambda = n\lambda \approx 0.17$. These default values are indicated as typical examples and are reported in Table B.1, but, in our experiments, we present results for a range of values of Λ between 0 and 1. We talk of *low churn systems* for values of Λ below 0.4, and of *high churn* systems for values above 0.4.

B.3.2 Metrics

To evaluate the performance of the different protocols, we are interested by the following metrics.

Number of people not receiving the video. Due to repairs and churn events, some people do not receive the video during small periods of time. We study what fraction of the nodes do not receive the video and during

which amount of time.
Height of the tree or delay. The height of the diffusion tree gives the maximum delay between the source and a node.
Number of interruptions and interruption duration. We monitor the number of interruptions of the video diffusion to a node during the broadcast lifetime, as well as the distribution of interruptions duration.

B.4 Protocols for Heterogeneous Systems

We present three new repair protocols using different levels of knowledge. We then compare them, in particular to understand the trade-off between knowledge and performance.

B.4.1 Description of the Protocols

To obtain a tree with minimum height while respecting the degree constraints, the following *two optimality conditions* must hold:

1. *If there exists a node at level L, all previous levels must be complete*;
2. *For each pair of levels* $(L, L+1)$ *the minimum bandwidth between all the nodes at level L must be greater than or equal to the maximum bandwidth between all the nodes at level* $L+1$.

Thus, the distributed protocols that we propose try to maintain nodes with high bandwidth on top of the diffusion tree. To this end, when a node is overloaded, it has to select carefully which node is pushed under which node using the information it has on the diffusion tree.

Local Bandwidth Protocol (LBP) In this protocol, each node knows the bandwidth of each of its children. Moreover, a node keeps track of the number of *push operations* done on each of its children. Note that this is a very local information that the protocol can easily get.

When a node is overloaded, it pushes its son with the smallest bandwidth, as nodes with higher bandwidth should stay on top on the diffusion tree. This son has to be pushed on a node receiving the video. We consider all the nodes receiving the video and push on them proportionally to their bandwidth (e.g., a node with a bandwidth of $d = 4$ should receive twice as much pushes than a node with a bandwidth of $d = 2$). In details, the expected number of pushes to each son is proportional to its bandwidth. The

father then pushes into the son with the largest difference between the push operations done and the expected number of push operations.
Bandwidth Distribution Protocol (BDP) In this protocol, nodes have additional information. Each node knows the bandwidth of each of its children and the bandwidth distribution of the subtree rooted in each of its children. Note that the bandwidth distribution can be pulled up from the subtree. This may be consider costly by the system designer. However, it is also possible for a node to estimate this distribution by keeping the information of the bandwidth of the nodes pushed into it.

This additional information allows to estimate the optimal height of the subtrees of each of the sons. Thus, when a node is overloaded, it can push its son with the smallest bandwidth into its son (receiving the video) with the smallest *estimated height*.
Full Level Protocol (FLP) In this protocol, each node knows the bandwidth of each of its children and the *last full level* of the subtree rooted in each of its children.
This information allows to know in which subtree nodes are missing. The main idea is to push toward the first available slot in order to not increase the height of the diffusion tree. When a node is overloaded, its son with the smallest bandwidth is pushed into the son with the smallest *last full level*.
For all three protocols, a churn event is handled similarly. When a node leaves the system, the children of the falling node are adopted by their grandfather.

Discussion. In LBP, a node has no information about the underlying subtree. Push operations are carried out according to the degree of the nodes that receive the video. Since nodes may leave the system at any moment, it can thus happen that a node is pushed into the worst subtree.
In BDP, a node knows the bandwidth distribution of each subtree rooted in each one of its children. A node is pushed according to the estimated height of the subtree rooted in its children. In the estimation a node assumes that all the levels, except the last one, is complete. Hence, it can happen that a node is not pushed on the best subtree.
In FLP, a node knows the last full level of the subtree rooted in each one of its children. A node is pushed under the node with the smallest value. In this way, we are sure that the node is pushed toward the best possible position. This may not be enough. In fact, due to nodes arrival and nodes departure the *two optimality conditions* may be broken.

Thus, in none of the protocols, the two optimality conditions are always true. However, the transmission delay is not the unique factor that determines the QoS for users. In fact, other factors like time to attach a new node, number and duration of interruptions are as important as the transmission delay. So, we decided to keep the protocols as simple as possible taking into consideration all metrics.

Handling Free Riders. Some nodes in the system can have no upload bandwidth and thus cannot distribute the video to anyone. They are called *free riders*. They may pose difficulties and calls for special treatment. First, an obvious observation is that protocols should not push nodes under a free rider. All our protocols prevent this. More problematic, in some situations, deadlocks may appear in which some nodes do not receive the video and all their brothers are free riders. The situation cannot be solved by push operations. In this case, we decided to ask the concerned free riders to rejoin the system. This is a solution we wanted to avoid as the goal of a structured protocol is to maintain as much as possible the parent-children relations of the diffusion tree. But, we considered that this is a small cost that free riders can pay, as they do not contribute to the system.

B.5 Simulations

We study and compare the performance of the protocols for 4 bandwidth distributions of live video streaming systems that we found in the literature and that we present in Section B.5.1. We first consider a scenario with a constant population to understand well the differences between the protocols in Section B.5.2. We then use real traces of live streaming from Twitch in Section B.5.3. Last, we discuss how to update in practice the information needed by the protocols in Section B.5.4.

B.5.1 Bandwidth Distributions

In [Sri+04], the authors estimate the outgoing bandwidth that hosts in the live video streaming system can contribute. They use a combination of data mining, inference, and active measurements for the estimation. They set an absolute maximum bound of 20 for the out-degree. This means that each node can contributes for at most 20. In [SNG06], the authors consider uplink rates from 256 kbps to 5 Mbps according to a distribution that reflects today's

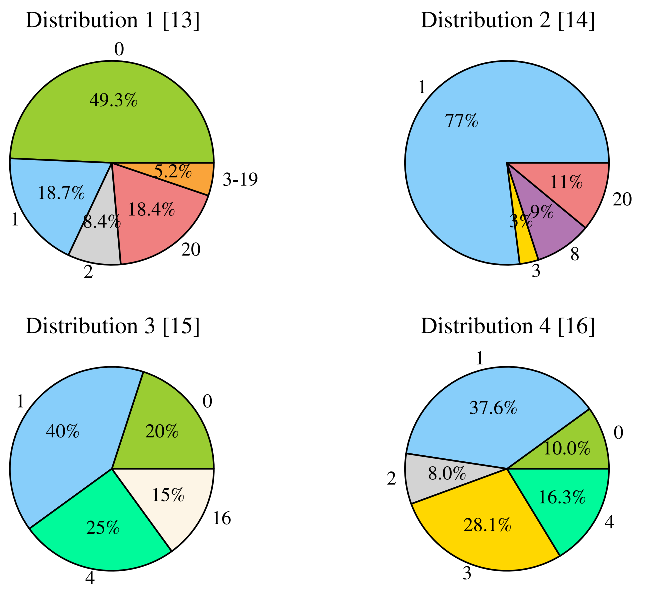


Figure B.3: Bandwidth distributions used in the simulations

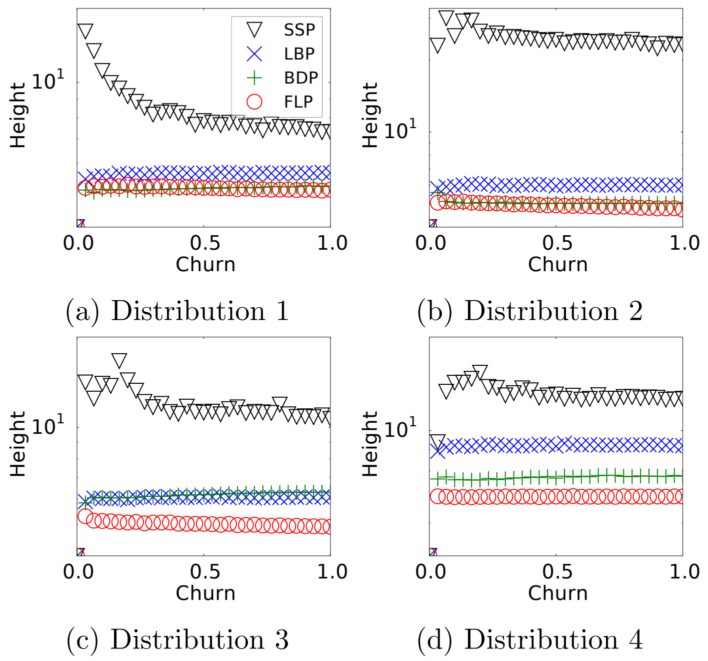
different network access technologies. They assume an encoding rate of 250 kbps. The degree is then calculated as $\lfloor uplink(Kbps)/250 \rfloor$. In [Eft+11], the authors set the uploading bandwidths equal to 4000 kbps, 1000 kbps, 384 kbps, and 128 kbps corresponding to a distribution of 15%, 25%, 40%, and 20% of peers. We use as encoding rate 250 kbps and then calculate the outgoing degree as in the previous case. In [Liu+08] the authors, to come up with an accurate bandwidth distribution of Internet users, consider the measurement studies in [Dis+07] for the overall distribution of residential and Ethernet peers and [HLR07] for the detailed bandwidth distribution of residential peers. As in the previous cases, we use 250 kbps as encoding rate to derive the degrees. The bandwidth distributions are summarized in Figure B.3. In three of them (Distributions 1, 3 and 4), there is the presence of *free riders*, peers that join the system without sharing their own resources. We see that the peer bandwidth can be very heterogeneous. For example, in Distribution 1, 49.3% of the peers are free riders, while 18.4% have bandwidth 20. Distributions 2 and 3 are similar (even if Distribution 2 has no free riders). The difference of degrees is smaller in Distribution 4. The fact that the degree distributions are pretty different allow us to test our algorithm for different settings.

B.5.2 Constant population

We compare the three protocols presented in the previous section according to four simple metrics: (i) average and maximum delay, (ii) percentage of people receiving the video, (iii) number of interruptions and (iv) their duration. We also add the SMALLEST SUBTREE PROTOCOL (SSP) to the comparison, an optimal repair protocol for homogeneous systems proposed in [GH15]. We consider a range of churn rate between 0 and 1 on a system with 1022 clients for a duration of 30 000 unit of time. The population is constant in the system as we consider that when a node leaves the system, an other node rejoins at the root. The metrics are averaged over 30 experiments.

Height of the diffusion tree and delay

Figure B.4 shows the average height (over time) of the diffusion tree as a function of the churn rate for the 4 bandwidth distributions. For all the distributions, SSP is the worst, by far, in terms of height and delay. Thus, SSP is not a good choice when peers have heterogeneous upload bandwidth capabilities. It is crucial to take the peer bandwidth into account and to try to place peers with high bandwidth on top of the diffusion tree, as we do in the three protocols proposed. For the 4 distributions, the average height of LBP is higher than the one of BDP which is higher than the one of FLP. They are respectively 5.5, 5.1, 5.1 for Distribution 1, 6, 5, 4.8 for Distribution 2, 6.2, 6, 5 for Distribution 3, 9.6, 7.6, and 7 for Distribution 4. Since the three protocols have the same behavior in the case of churn event, the explanation of the different behaviors relies on the different ways an overloaded node pushes a son. From the plots we can derive three main facts: (1) Homogeneous protocols are very inefficient for systems with heterogeneous upload bandwidth capabilities. (2) More information allows us to take better decisions. FLP is the best, followed by BDP and LBP (3) However, even with only very local information, we can have good results. In LBP, a node has access to a a very small amount of information about the underlying subtree and its performance is close to the one of FLP in which a node has more information about its subtree.



(a) Distribution 1 (b) Distribution 2

(c) Distribution 3 (d) Distribution 4

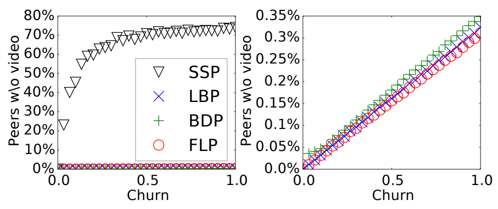
Figure B.4: Average height of the diffusion tree as a function of the churn rate (1022 nodes).

Percentage of people without the video during time.

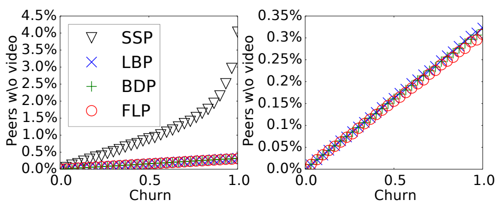
Figure B.5 shows the average percentage of nodes which do not receive video as a function of the churn rate. As for the previous metric, SSP shows very bad results when peers have heterogeneous upload bandwidth capabilities. In fact, the percentage of people without video arrives to 70% for Distribution 1. On the other side, the *three proposed protocols behave in a very similar and efficient way*. The average percentage of peers without video for the three protocols LBP, BDP and FLP never exceeds 0.2%, if we consider systems with low churn ($\Lambda \in [0, 0.4]$), and 0.6%, if we consider systems

with high churn ($\Lambda \in [0.4, 1]$). The explanation of these similar behaviors is that a churn event is handled in the same way for the three protocols: the children of the peer, which leaves the system, are adopted by their grandfather.

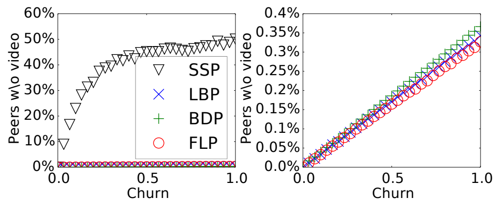
Number and duration of interruptions. Figures B.7 and B.8 show respectively the average number of interruptions for a node present in the system during 30 000 units of time and the average fraction of time a node



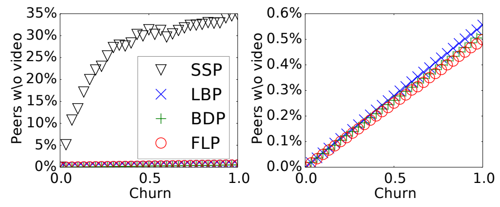
(a) Distribution 1



(b) Distribution 2

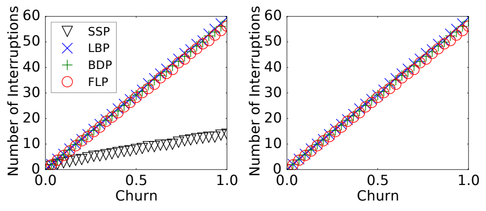


(c) Distribution 3

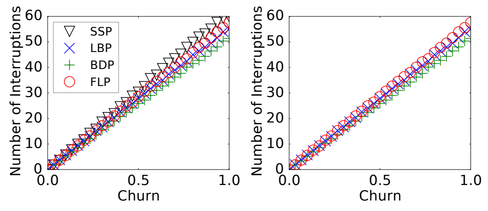


(d) Distribution 4

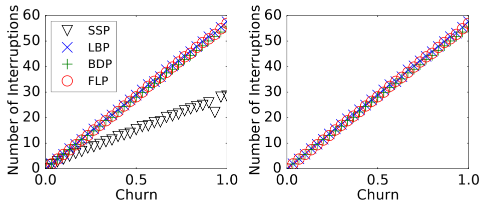
Figure B.5: Average fraction of peers not receiving the video as a function of the churn rate (1022 nodes). Right: Same plot without SSP protocol.



(a) Distribution 1



(b) Distribution 2



(c) Distribution 3

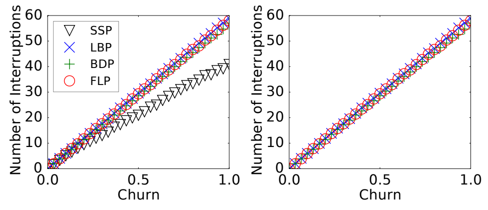
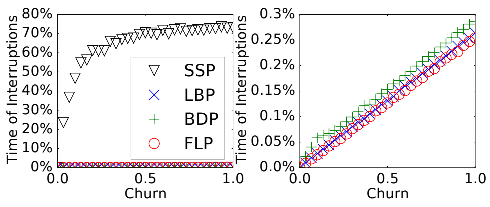
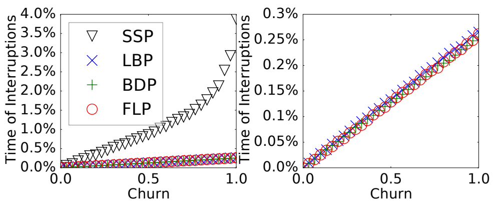


Figure B.6: Distribution 4

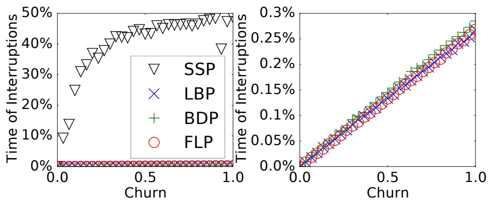
Figure B.7: Average number of interruptions of the streaming per node as a function of the churn rate. Right: Same plot without SSP protocol.



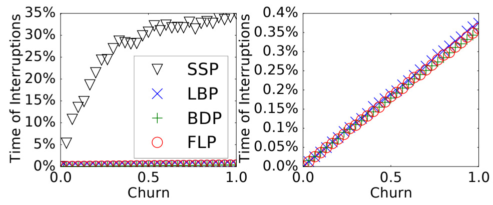
(a) Distribution 1



(b) Distribution 2



(c) Distribution 3



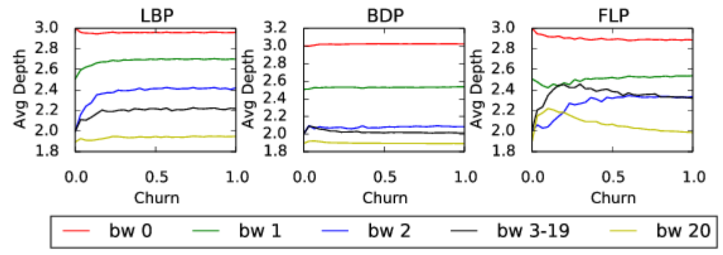
(d) Distribution 4

Figure B.8: Average fraction of time for which the streaming as a function of the churn rate. Right: Same plot without SSP protocol

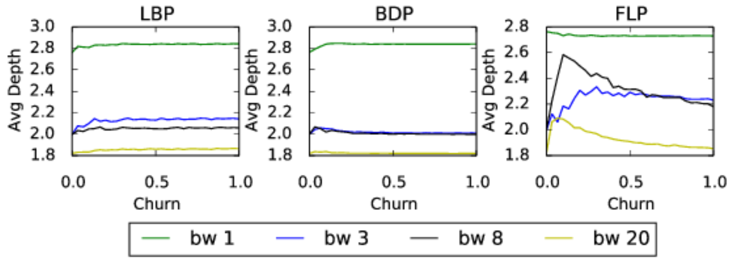
was interrupted. The tree protocols LBP, BDP and FLP have the same behavior. For the default value of churn rate, 0.17, measured on real PPLive [H+06], an average of 9 interruptions can be experienced for all the distributions.

This corresponds to an interruption every 3500 units of time (6 minutes). These interruptions have an imperceptible duration. In fact, for $\Lambda = 0.17$ in the worst case a node is interrupted in average for 0.05% of the time only. This corresponds to only 1.5 s. As discussed in the next section, with a very small buffer of few seconds of video, the users won't experience any interruption. SSP exhibits a different behavior for this metric. For 3 of the 4 bandwidth distributions, it is the best protocols in terms of number of interruptions. However, the duration of these interruptions are much bigger. In fact, a peer may be interrupted even for the 75 % of the time for Distribution 1. The explanation of this behavior relies on the fact that, when a node in SSP is interrupted, it has to wait a high amount of time until it will start to see the video again. It is also explained by the lack of handling mechanism for *free riders*. This confirms that SSP should not be used in systems with heterogeneous peers. From now on, we will exclude SSP from our analysis and focus just on LBP, BDP and FLP.

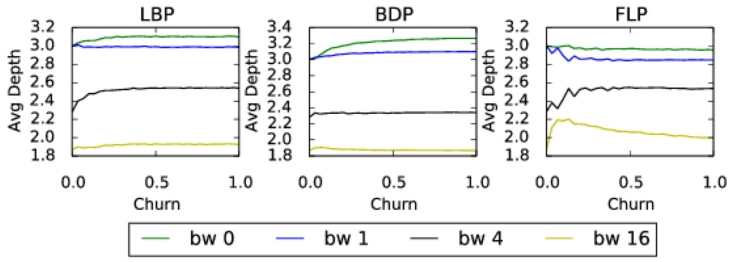
Average depth for each value of bandwidth. Figure B.9 shows the



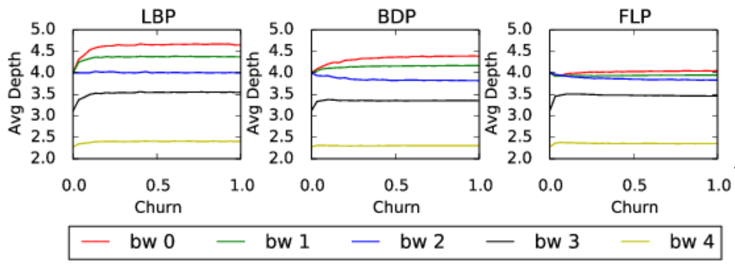
(a) Distribution 1



(b) Distribution 2



(c) Distribution 3



.l

(d) Distribution 4

Figure B.9: Average depth for each value of bandwidth of the distribution for the 3 protocols

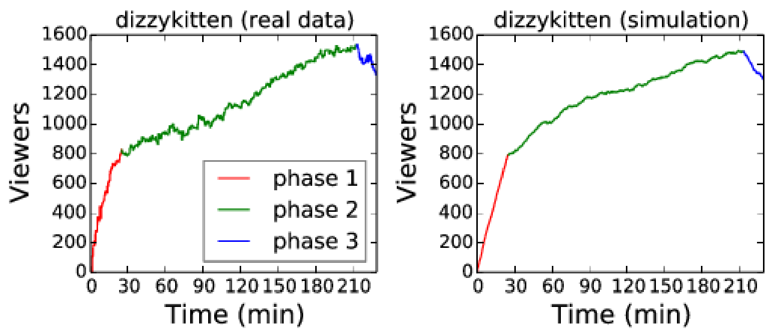


Figure B.10: Number of viewers as a function of the time for a streaming session of the user *dizzykitten*.

average depth (over time and over all nodes with the same bandwidth) for each bandwidth value as function of the churn rate. For all the distributions, nodes with the highest values of bandwidth are in the top layers of the tree and nodes with the smallest values of bandwidth are in the bottom positions of the tree. This leads to a better QoS for the peers sharing the most. These peers experience the smallest possible delay and the smallest number of interruptions among all the peers in the system. In fact, a peer is interrupted by the departure of one of its ancestors. Thus, peers in the top layers have a smaller probability of getting interrupted. On the other side, peers sharing less experience higher delay and, in average, a larger number of interruptions.

We can conclude that, for the three protocols LBP, BDP *and* FLP, *the more a peer shares, the better its QoS is. A node has thus incentives to share its own resources and the protocols are robust to free riders.*

B.5.3 Results with Twitch Traces

In order to simulate the protocols in real scenarios, we decided to use Twitch [] as a Use Case. Twitch is a live streaming video platform that mainly focuses on video gaming and e-sport events. Twitch made its first appearance in 2011 and its popularity grew very fast. Today, with 1.5 million broadcasters and 100 millions visitors per month Twitch represents the 4th largest source of peak Internet traffic in the US [MB14]. With the goal of understanding the behavior of the viewers of the stream, we monitored the 100 most famous streamers (in terms of viewers and followers [Bla]).

	Phase 1	**Phase 2**	**Phase 3**
Arrival rate ϕ	0.0623196	0.0247548	0.0247548
Individual churn rate λ	$1.572 * 10^{-5}$	$1.572 * 10^{-5}$	$3.155 * 10^{-5}$
Duration (time units)	14640	112890	9648

Table B.2: Parameters obtained from the model for the considered streaming session

We gathered the number of viewers for each moment of their streaming sessions. We noticed that most of the streaming sessions can be divided in 3 phases: (1) the start of the stream with a number of viewers increasing at an extremely high rate, (2) the central part of the stream with the number of viewers increasing at a slower rate than at the beginning of the stream and (3) the end of the stream with a decreasing number of viewers. See Figure B.10 (Left)
We defined a model to represent streaming sessions that follow the 3–phases pattern. This model allows us to abstract from the real data and to repeat the simulations several time in order to estimate the quantitative behavior of the protocols more easily. Using the fact that, in average, a user spends 106 minutes on *twitch.tv* [Twi], for each phase, we calculate the arrival rate and the individual churn rate, modeled as a Poisson model. Table B.2 shows the rates given to the simulator for the considered streaming session. In order to calculate the leaving rate of the 3rd phase, we assumed that the arrival rate of Phase 3 is the same as the one of Phase 2. Fig. B.10 compares the data obtained from monitoring the user with the data obtained from the model.

We simulated our 3 protocols using the data generated from the model for our metrics of interest and according to the four distributions of bandwidth presented in the previous section. Table B.3 summarizes the average results of the 3 protocols after 250 simulations.

Height of the diffusion tree and delay. We see that *all protocols achieve a very small height of the diffusion tree*: around 5 or 6 for the average. Recall that we have around 1500 users at the maximum of the stream. The protocols are thus very efficient. The evolution of the diffusion tree height is given as an example in Figure B.11. We see the increase of height when the users connect to the stream till a maximum height of 6 for FLP and LBP, and of 7 for BDP. FLP for all the distributions gives us the tree with the smallest height. The results of LBP and BDP are different from the

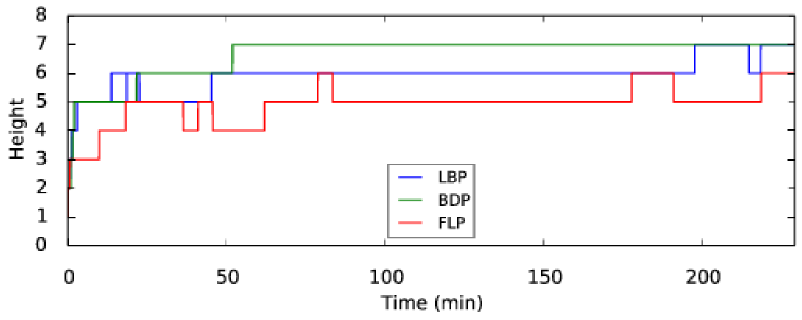


Figure B.11: Height of the diffusion tree during time for an example of Twitch session *dizzykitten*.

simulation case of previous section. In fact, since the individual churn rate is very small during the experiment ($\sim 1.5 * 10^{-5}$), *pushing according to the children's bandwidths (local information) reveals to be a good strategy*, leading to a better height than BDP for 2 of the 4 distributions. In particular LBP behaves better than BDP in the case of very distant values of bandwidth (Distributions 1 and 3) and worst when the values of bandwidth are really close between them (Distribution 4).

Percentage of people without the video during time. In this case the 3 protocols behave similarly, as in the simulation case. They are very efficient: in average, only 0.2% of peers are unable to watch the video. Recall that we count the users arriving and waiting to be connected at the right place of the tree.

Number of interruptions during the diffusion. The protocols have a similar behavior in most cases. For Distributions 1, 2 and 3 the numbers of interruptions are between 12 and 21, for Distributions 4 is between 5 and 6. This means that, in the worst case, a peer staying for all the duration of the stream experiences an interruption every 10 minutes. In all cases the duration of these interruptions is very small. Considering all protocols and all the distributions, a node is never interrupted for more than the 0.02% of the time. This means, that a peer remaining during all the stream session is interrupted for less than 3 seconds. A buffer of few seconds (e.g. 10s) of video will make these interruptions imperceptible to the end-users. For a video rate of 480 kbps, it corresponds to a buffer size of only 40MB.

	LBP	**BDP**	**FLP**
Height	5.66	6.1	5.17
People w/o video (%)	0.21	0.24	0.22
Number of Interruptions	17.54	12.12	20.87
Time of Interruptions (%)	0.01	0.01	0.017

(a) Distribution 1

	LBP	**BDP**	**FLP**
Height	6.08	6.13	4.98
People w/o video (%)	0.21	0.21	0.22
Number of Interruptions	17.05	12.35	21.47
Time of Interruptions (%)	0.008	0.008	0.0013

(b) Distribution 2

	LBP	**BDP**	**FLP**
Height	5.9	6.62	4.9
People w/o video (%)	0.21	0.21	0.22
Number of Interruptions	17.07	14.17	20.05
Time of Interruptions (%)	0.009	0.01	0.013

(c) Distribution 3

	LBP	**BDP**	**FLP**
Height	9.12	7.9	7.06
People w/o video (%)	0.2	0.2	0.2
Number of Interruptions	5.62	5.57	6.76
Time of Interruptions (%)	0.009	0.008	0.009

(d) Distribution 4

Table B.3: Average metrics after 250 simulations

B.5.4 Information update

As we just showed in the last subsection, more information about the subtree leads to better repairs and thus to a better QoS for the clients. However, in order to maintain this information accurate, a node must exchange with all the nodes in its subtree. If the rate at which these exchanges are done is too high, it can lead to message overhead and to a high delay from the moment a node joins the system until when the node starts to see the video. As a consequence, we tested the performance of FLP with the periodic update of the last full level information for a node. In the experiments we compare LBP with FLP protocol in the case a node obtains the exact last full level information after fixed time length intervals, expressed in time units.

Impact of churn rate. Figure B.16 shows the average height of the tree as a function of the churn rate for LBP, FLP and FLP with different update times. For low churn rates, the influence of the update time is barely noticeable. Introducing a delay of information only increases the average height of the tree of about half a level at most. As the churn increases, longest update time gives higher trees. However, even for high churn, FLP with update remains better than LBP for intervals between update up to 100 t.u. (i.e., 10 s), except for Distribution 2. Having larger update times, e.g. 200 t.u., gives worst results than LBP for these churn values. This shows that FLP with periodic update should work well in practice, as the churn rate of practical systems is between 0.1 and 0.2, see the discution of Section B.3.1. We confirm this in the following by consider Twitch streaming session.
With Twitch streaming sessions. Figure B.21 shows the average height of the tree during the streaming session for the protocols LBP, FLP and FLP with periodic update of the last full level. For Distribution 1, we see that FLP behaves better than LBP with refresh rates lower than 1100 units of time (corresponding to 1 min and 50 s). For Distribution 2, 3, and 4, the thresholds for the refresh rate respectively are 600 time units (1 min), 1 min and 20 s, and 2 min Such refresh rates correspond to only a few control messages around every minutes, which is negligible compared to the bandwidth used by the video streaming. This allows us to conclude that FLP can be used in real deployment, if we set a refresh rate according to the needs of the provider and to the arrival and churn rate of the peers in the stream. In fact, in this way, we can significantly reduce the overhead resulting from finding the last full level of the subtree at each iteration.

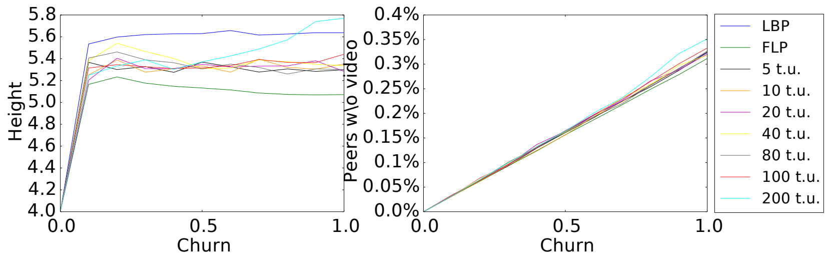


Figure B.12: Distribution 1

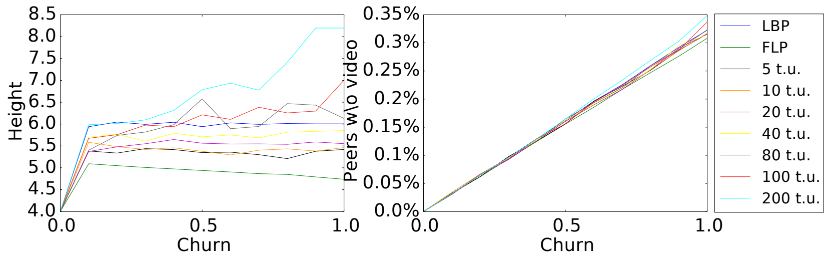


Figure B.13: Distribution 2

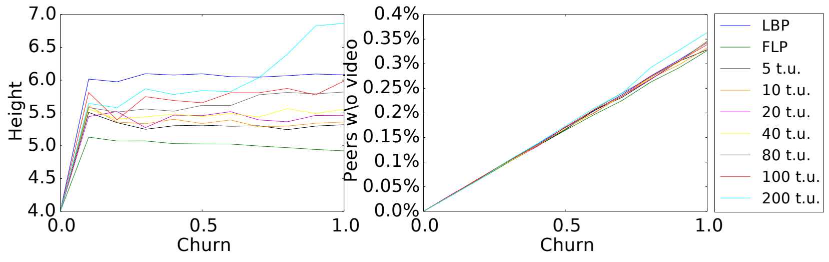


Figure B.14: Distribution 3

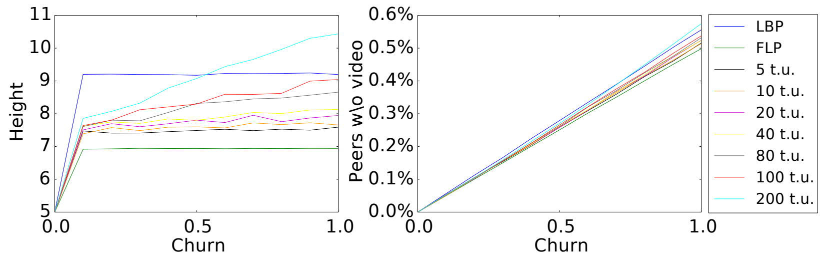


Figure B.15: Distribution 4

Figure B.16: Average height of the tree (left) and percentage of people without video (right) as a function of the churn rate for LBP, FLP and FLP with periodic update of the *last full level*

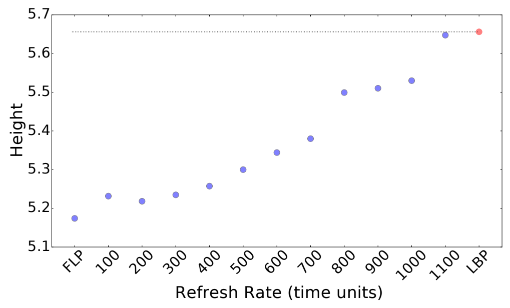


Figure B.17: Distribution 1

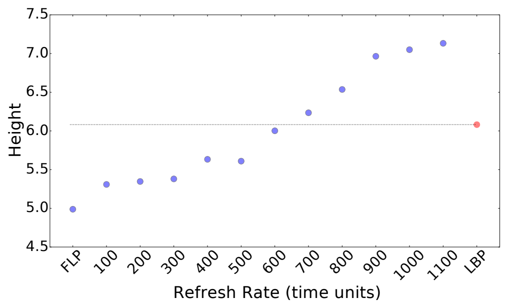


Figure B.18: Distribution 2

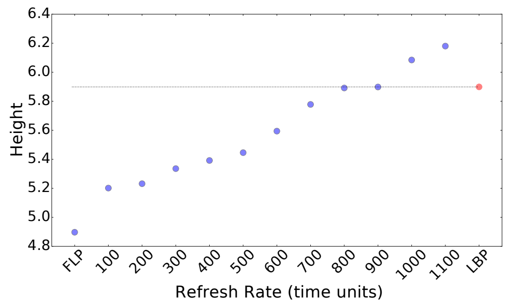


Figure B.19: Distribution 3

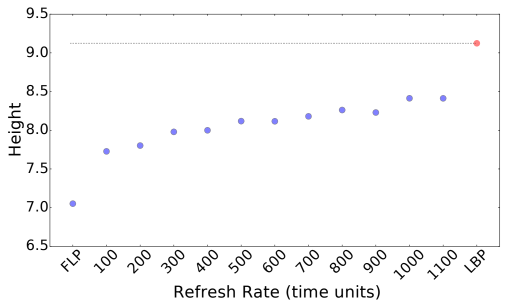


Figure B.20: Distribution 4

Figure B.21: Average height of the tree for LBP, FLP and FLP with periodic update of the *last full level* for the considered streaming session

B.6 Conclusion

In this study we examined the problem of delivering live video streaming in a P2P overlay network using a structured overlay. We have proposed 3 distributed protocols to repair the diffusion tree of the overlay, when there is churn. The protocols use different amounts of information. Using simulations and experiments with real traces from Twitch, we have shown that our protocols behave well with respect to fundamental QoS metrics, even for very heterogeneous bandwidth distributions. *Our main result is that, with very simple distributed repair protocols using only a small amount of information, structured overlay networks can be very efficient and resistant to churn.*

Bibliography

[Zha+05a] X. Zhang, J. Liu, B. Li, and T.S.P. Yum. “CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming”. In: *proceedings of IEEE Infocom*. Vol. 3. 2005, pp. 13–17 (cit. on p. 226).

[Li+13a] Baochun Li, Zhi Wang, Jiangchuan Liu, and Wenwu Zhu. “Two Decades of Internet Video Streaming: A Retrospective View”. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 9.1s (Oct. 2013), 33:1–33:20. ISSN: 1551-6857 (cit. on p. 226).

[WXL10] Feng Wang, Yongqiang Xiong, and Jiangchuan Liu. “mTreebone: A collaborative tree-mesh overlay network for multicast video streaming”. In: *Parallel and Distributed Systems, IEEE Transactions on* 21.3 (2010), pp. 379–392 (cit. on p. 226).

[CRZ00] Y. hua Chu, S.G. Rao, and H. Zhang. “A case for end system multicast”. In: *Proc. of ACM Sigmetrics*. 2000, pp. 1–12 (cit. on p. 226).

[Cas+03a] M. Castro, P. Druschel, A.M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. “SplitStream: high-bandwidth multicast in cooperative environments”. In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. 2003, p. 313 (cit. on pp. 226, 227).

[THD03] D.A. Tran, K.A. Hua, and T. Do. “ZIGZAG: an efficient peer-to-peer scheme for media streaming”. In: *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*. Vol. 2. Mar. 2003, 1283–1292 vol.2 (cit. on pp. 226, 244).

[DBG01] Hrishikesh Deshpande, Mayank Bawa, and Hector Garcia-Molina. “Streaming live media over a peer-to-peer network”. In: *Technical Report* (2001) (cit. on p. 226).

[Coh03] B. Cohen. “Incentives build robustness in BitTorrent”. In: *Workshop on Economics of Peer-to-Peer systems*. Vol. 6. Citeseer. 2003, pp. 68–72 (cit. on p. 226).

[Ban+03] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. "Resilient multicast using overlays". In: *ACM SIGMETRICS Performance Evaluation Review*. Vol. 31. 1. ACM. 2003, pp. 102–113 (cit. on p. 226).

[VYF06] Vidhyashankar Venkataraman, Kaouru Yoshida, and Paul Francis. "Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast". In: *14th IEEE International Conference on Network Protocols*. 2006, pp. 2–11 (cit. on pp. 226, 244).

[DFC07] G. Dan, V. Fodor, and I. Chatzidrossos. "On the performance of multiple-tree-based peer-to-peer live streaming". In: *26th IEEE International Conference on Computer Communications*. 2007, pp. 2556–2560 (cit. on pp. 226, 244).

[Gir+13a] F. Giroire, R. Modrzejewski, N. Nisse, and S. Pérennes. "Maintaining Balanced Trees For Structured Distributed Streaming Systems". In: *20th Colloquium on Structural Information and Communication Complexity (SIROCCO)*. Vol. LNCS 8179. Ischia, Italy, 2013, pp. 177–188 (cit. on p. 227).

[H+06] X. Hei, C. Liang, J. Liang, et al. "Insights into PPLive: A Measurement Study of a Large-Scale P2P IPTV System". In: *International Word Wide Web Conference. IPTV Workshop*. 2006 (cit. on pp. 229, 246, 255).

[Vu+07] L. Vu, I. Gupta, J. Liang, and K. Nahrstedt. "Measurement of a large-scale overlay for multimedia streaming". In: *Proceedings of the 16th international symposium on High performance distributed computing*. ACM. 2007, pp. 241–242 (cit. on pp. 229, 246).

[Li+08] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang. "Inside the new coolstreaming: Principles, measurements and performance implications". In: *27th IEEE International Conference on Computer Communications*. 2008 (cit. on pp. 229, 247).

[Bon+08] Thomas Bonald, Laurent Massoulié, Fabien Mathieu, Diego Perino, and Andrew Twigg. "Epidemic live streaming: optimal performance trade-offs". In: *ACM SIGMETRICS Performance*

Evaluation Review. Vol. 36. 1. ACM. 2008, pp. 325–336 (cit. on pp. 243, 244).

[Li+13b] Baochun Li, Zhi Wang, Jiangchuan Liu, and Wenwu Zhu. "Two Decades of Internet Video Streaming: A Retrospective View". In: *ACM Trans. Multimedia Comput. Commun. Appl.* 9.1s (Oct. 2013), 33:1–33:20. ISSN: 1551-6857. DOI: 10.1145/2505805. URL: http://doi.acm.org/10.1145/2505805 (cit. on p. 244).

[WXL07] Feng Wang, Yongqiang Xiong, and Jiangchuan Liu. "mtreebone: A hybrid tree/mesh overlay for application-layer live video multicast". In: *27th International Conference on Distributed Computing Systems (ICDCS'07)*. IEEE. 2007, pp. 49–49 (cit. on p. 244).

[Zha+05b] Xinyan Zhang, Jiangchuan Liu, Bo Li, and T.P. Yum. "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming". In: *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*. Vol. 3. Mar. 2005, 2102–2111 vol. 3. DOI: 10.1109/INFCOM.2005.1498486 (cit. on p. 244).

[Cas+03b] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. "SplitStream: high-bandwidth multicast in cooperative environments". In: *ACM SIGOPS Operating Systems Review*. Vol. 37. 5. ACM. 2003, pp. 298–313 (cit. on p. 244).

[Gir+13b] Frédéric Giroire, Remigiusz Modrzejewski, Nicolas Nisse, and Stéphane Pérennes. "Maintaining balanced trees for structured distributed streaming systems". In: *International Colloquium on Structural Information and Communication Complexity*. Springer. 2013, pp. 177–188 (cit. on p. 245).

[Gir+17] Frédéric Giroire, Remigiusz Modrzejewski, Nicolas Nisse, and Stéphane Pérennes. "Maintaining balanced trees for structured distributed streaming systems". In: *Discrete Applied Mathematics* (2017) (cit. on p. 245).

[GH15] Frédéric Giroire and Nicolas Huin. "Study of Repair Protocols for Live Video Streaming Distributed Systems". In: *GLOBECOM 2015*. San Diego, United States, Dec. 2015. URL: https://hal.inria.fr/hal-01221319 (cit. on pp. 245, 252).

[Sri+04] Kunwadee Sripanidkulchai, Aditya Ganjam, Bruce Maggs, and Hui Zhang. "The feasibility of supporting large-scale live streaming applications with dynamic application end-points". In: *ACM SIGCOMM Computer Communication Review* 34.4 (2004), pp. 107–120 (cit. on p. 250).

[SNG06] E. Setton, J. Noh, and B. Girod. "Low Latency Video Streaming Over Peer-To-Peer Networks". In: *2006 IEEE International Conference on Multimedia and Expo*. July 2006, pp. 569–572. DOI: 10.1109/ICME.2006.262472 (cit. on p. 250).

[Eft+11] Nikolaos Efthymiopoulos, Athanasios Christakidis, Spyros Denazis, and Odysseas Koufopavlou. "LiquidStream—network dependent dynamic P2P live streaming". In: *Peer-to-Peer Networking and Applications* 4.1 (2011), pp. 50–62 (cit. on p. 251).

[Liu+08] Zhengye Liu, Yanming Shen, Keith W Ross, Shivendra S Panwar, and Yao Wang. "Substream trading: towards an open P2P live streaming system". In: *Network Protocols, 2008. ICNP 2008. IEEE International Conference on*. IEEE. 2008, pp. 94–103 (cit. on p. 251).

[Dis+07] Marcel Dischinger, Andreas Haeberlen, Krishna P Gummadi, and Stefan Saroiu. "Characterizing residential broadband networks". In: *Internet Measurement Comference*. 2007, pp. 43–56 (cit. on p. 251).

[HLR07] Cheng Huang, Jin Li, and Keith W Ross. "Can internet video-on-demand be profitable?" In: *ACM SIGCOMM Computer Communication Review* 37.4 (2007), pp. 133–144 (cit. on p. 251).

[] URL: http://www.twitch.tv (cit. on p. 257).

[MB14] D. MacMillan and G. Bensinger. *Amazon to Buy Video Site Twitch for $970 Million*. Ed. by Wall Street Journal. Aug. 2014. URL: http://www.wsj.com/articles/amazon-to-buy-video-site-twitch-for-more-than-1-billion-1408988885 (cit. on p. 257).

[Bla] Social Blade. *Twitch Statistics*. URL: http://socialblade.com/twitch/ (cit. on p. 257).

[Twi] Twitch. *Twitch Blog*. URL: https://blog.twitch.tv/twitch-hits-one-million-monthly-active-broadcasters-21dd72942b32 (cit. on p. 258).