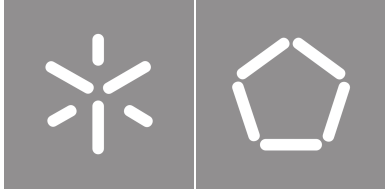
Universidade do Minho
Escola de Engenharia

Rui Pedro da Cunha Monteiro

Green communications: An environment to support energy-aware networks developments

February, 2023



Universidade do Minho
Escola de Engenharia

Rui Pedro da Cunha Monteiro

Green communications: An environment to support energy-aware networks developments

Master Thesis
Integrated Master in Informatics Engineering

Work developed under the supervision of:
João Marco Silva
Fábio André Coelho

February, 2023

Acknowledgements

I would first like to thank my supervisors, João Marco Silva and Fábio André Coelho, for the opportunity to work on this project and for all the guidance and valuable insights throughout this journey. I must also thank FCT and INESC TEC for the funding that made this work possible.

A special thanks to everyone at the University of Minho that I had the pleasure to meet and work with during these years. In particular to the amazing people at HASLab for providing a great work environment with unforgettable moments.

Finally, I'm extremely grateful to all my family and friends for their support over the years and for always being there for me. A special mention goes to my parents, that made all this possible and helped me become the person that I am today.

This work is financed by National Funds through the FCT - Fundação para a Ciência e a Tecnologia, I.P. (Portuguese Foundation for Science and Technology) within the project FLEXCOMM, with reference EXPL/CCI-INF/1543/2021.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

Braga, 06/02/2023

Assinado por: **Rui Pedro da Cunha Monteiro**
Num. de Identificação: 30061059
Data: 2023.02.06 08:51:33 +0000

(Rui Pedro da Cunha Monteiro)

Abstract

Green communications: An environment to support energy-aware networks developments

The continuous joint growth of the communication networks is causing a simultaneous increase in the energy consumption of these infrastructures. To help fight the consequent environmental impact is required that new traffic engineering techniques are developed to help mitigate this energy consumption.

This work proposes the Flexcomm Simulator [1], a simulation environment with the intent of creating a platform that helps to develop new routing algorithms, that combine Software Defined Networking (SDN) and Energy Flexibility techniques to optimize the energy consumption of large scale wired communication networks. The environment allows simulating real infrastructures conditions, so policies can be developed close to real scenarios, facilitating their deployment in real environments.

The tool's abilities have been demonstrated, by generating data that allows the evaluation of different routing strategies. Moreover, the Flexcomm Simulator has made possible the developments on early work of new algorithms that demonstrate the ability to achieve a more balanced energy consumption. These new techniques are capable of adapting to changes in energy availability and relocating network flows across different regions of a network, respecting flexibility imposed by electrical grids while maintaining a minimum Quality of Service (QoS).

Keywords: Network Simulators, Software Defined Networking, Energy Models, OpenFlow

[1] Available at `https://github.com/RuiCunhaM/Flexcomm-Simulator`

Resumo

Green communications: Ambiente para planeamento de redes eficientes

O contínuo crescimento conjunto das redes de comunicação está a causar simultaneamente um aumento no consumo energético destas infraestruturas. Para ajudar a combater o consequente impacto ambiental, é necessário que novas técnicas de engenharia de tráfego sejam desenvolvidas para ajudar a mitigar este consumo energético.

Este trabalho propõe o Flexcomm Simulator, um ambiente de simulação com o intuito de criar uma plataforma que ajude no desenvolvimento de novos algoritmos de *routing*, que combinem técnicas de SDN e Flexibilidade Energética para otimizar consumo energético em redes de comunicação de larga escala. O ambiente permite simular condições de infraestruturas reais, de forma a que políticas possam ser desenvolvidas o mais perto possível da realidade, facilitando a sua adaptação em ambientes reais.

As capacidades da ferramenta foram demonstradas, gerando informação que permite avaliar diferentes estratégias de *routing*. Para além disso, o Flexcomm Simulator tornou possível o desenvolvimento de trabalho inicial em novos algoritmos que demonstram a capacidade de atingir um consumo energético mais balanceado. Estas novas técnicas demonstram a capacidade de se adaptarem a mudanças na disponibilidade energética e de realocarem fluxos de comunicação ao longo de vários pontos de uma rede, respeitando a flexibilidade imposta pela rede elétrica enquanto mantendo níveis mínimos de qualidade de serviço.

Palavras-chave: Simuladores de Rede, Software Defined Networking, Modelos de Energia, OpenFlow

Contents

List of Figures

List of Listings

Acronyms

API Application Programming Interface
ASN.1 Abstract Syntax Notation One

CORE Common Open Research Emulator
CPU Central Processing Unity
CSMA Carrier-Sense Multiple Access

FVC Flexibility Value Chain

gRPC Google Remote Procedure Call
GUI Graphical User Interface

HPC High Performance Computing
HTTP Hypertext Transfer Protocol

ICMP Internet Control Message Protocol
ICT Information and Communication Technologies
INT Inband Network Telemetry
IoT Internet of Things
IP Internet Protocol
IPC Inter-Process Communication
IPPM IP Performance Measurement
ISP Internet Service Provider

JSON JavaScript Object Notation

LLDP Link Layer Discovery Protocol
LP Logical Process

MAC Media Access Control
MACC Minho Advanced Computing Center
MAN Metropolitan Area Network
MIB Management Information Base
MPI Message Passing Interface

NED NEtwork Description

OID Object IDentifier
ONF Open Network Foundation
OS Operating System
OSPF Open Shortest Path First
OVS Open vSwitch

P4 Programming Protocol-independent Packet Processors
PCAP Packet Capture
PPBP Poisson Pareto Burst Process
PPP Point-to-Point Protocol

QoS Quality of Service

RAM Random Access Memory
REST Representational State Transfer

SDN Software Defined Networking
SNMP Simple Network Management Protocol

TCAM Ternary Content-Addressable Memory
TCP Transmission Control Protocol
TOML Tom's Obvious Minimal Language
TOR The Onion Router

UDP User Datagram Protocol
USEF Universal Smart Energy Framework

VOIP Voice Over IP

WAN Wide Area Network

XML Extensible Markup Language

1 Introduction

The Information and Communication Technologies (ICT) sector is continuing to grow, and recent studies show that it is responsible for 5 to 9% of the global electricity consumption and more than 2% of greenhouse gas emissions [1], [2]. Furthermore, is estimated that half of these amounts are due to data transmission networks alone [3]. Considering the predicated large number of Internet of Things (IoT) devices at the global scale by 2025 [4], followed by the associated large amounts of data transmitted [5] and the current 5G [6] and upcoming 6G [7] generations of mobile networks being deployed and developed simultaneously, these numbers are only expected to continuously increase [8]. Taking this trend into account, the way how massive amounts of data are transmitted is an increasingly important topic due to their environmental impact [9], [10]. Thus, pursuing sustainable communication infrastructures to combat this tendency is now more critical than ever.

Besides the impact on the environment, the sheer scale of the infrastructures required to accommodate these data volumes, also translates into expensive operating costs for service providers. Therefore, the search for optimal solutions that help to reduce economical expenses, also promotes the research for new strategies in traffic engineering.

At the same time, the growth of these infrastructures is increasing the deployment of SDN, a technology that introduces a new paradigm in how networks are managed, since it allows to dynamically and programmatically configure networks in real-time. This is in turn also promoting the research of new techniques in efficient traffic engineering [11] combined with selectively turning off devices in low usage periods [12]. However, these last ones are usually only aimed at reducing operating costs while maintaining a minimum level of QoS, not focussing in reducing the environmental impact.

Following the innovation in all sectors, the energy market is also undergoing a digital transformation, leading to the emergence of Smart Grids. These modern energy grids aim at being more intelligent and autonomous in their operation, but are also capable of providing more data about their state. The information about their behavior can then be processed to determine the most efficient ways of optimizing the energy consumption of network infrastructures depending on the energy availability at a given time.

Due to the difficulty in developing and evaluating new green routing strategies that manipulate network infrastructures based on the information provided by Smart Grids, the main motivation behind this work

is to create a tool that helps to simplify this process. It shall recreate realistically different aspects of physical networks, ensuring that newly developed techniques can be ported to real scenarios with little to no modifications required. Moreover, the results produced by the tool should closely reflect real values, providing an accurate method to help validate the impact in QoS and energy consumption of new routing policies. Another important aspect for the Flexcomm Simulator, proposed in this work, is to be easy to use and easy to access, providing a platform that mainly focuses on developing new innovative algorithms.

1.1 Objective

The objective of this dissertation is to develop a simulation environment that allows to prototype and evaluate different strategies that combine SDN and the available data from Smart Grids to reduce the energy impact of network infrastructures. This environment consists of a tool that allows network operators and researchers to determine how different techniques affect the energy consumption and QoS of wired networks without the need to interfere with production environments. To achieve these goals the tool has the following requirements:

- Recreate real network infrastructures - Users must be able to recreate their real network topologies, so the evaluation can be as accurate as possible.
- Recreate real traffic conditions - Similarly to infrastructures, the network traffic recreated in the simulation must follow a realistic behavior to ensure trustworthy results.
- Realistic energy models - When estimating energy consumption for devices composing the network, the models must produce realistic values.
- Easy to use - To ensure that the tool can be used by anyone to develop new strategies, the tool must be simple to operate.

1.2 Dissertation outline

In the next chapters, Chapter 2 starts by presenting some background on SDN and Energy Flexibility, followed by related work on the subject of tools aimed at recreating wired networks for research. Next, Chapter 3 introduces the Flexcomm Simulator, a simulation environment that consists on the main contribution of this work. The same chapter then proceeds to discuss the considerations behind the development of the tool and its architecture. Chapter 4 showcases the simulator functionality by demonstrating its different abilities, and how it can be used to help develop new routing policies. Finally, Chapter 5 presents the conclusions on this work and the perspectives for future work.

2

Related Work

This chapter starts with the background on SDN and Energy Flexibility. It then proceeds to introduce related work in network simulation and emulation, by discussing different methods used to replicate networks. The most relevant tools with this purpose are presented alongside with some of their most important aspects and functionality.

2.1 Background

Before presenting the related work on the subjects of network simulation and emulation, this section provides an overview on SDN and its concepts, followed by some related technologies. It also discusses some insights on energy flexibility in modern electrical grids, and how it can be leveraged to improve energy consumption.

2.1.1 Software Defined Networking

SDN is a dynamic and manageable approach to network management. This architecture tries to address the static nature of the conventional networks mainly by decoupling the control and data planes [13]. To achieve this, it centralizes the control plane while maintaining an open interface between the devices of both planes and providing programmable access of the network to external applications. The main concept behind such architecture has been around since the late *90s*, but in the latest years with the emergence of new technologies like OpenFlow [14] and Programming Protocol-independent Packet Processors (P4) [15], this type of architecture has increased in popularity and deployment. In this new paradigm, the network architecture, as illustrated in Figure 1, differs from traditional networks by possessing one or more *Controllers* that have a global view of the network. These controllers can alter the behavior of a network through a *Southbound API*, while themselves can receive instructions from applications through a *Northbound API* [16].

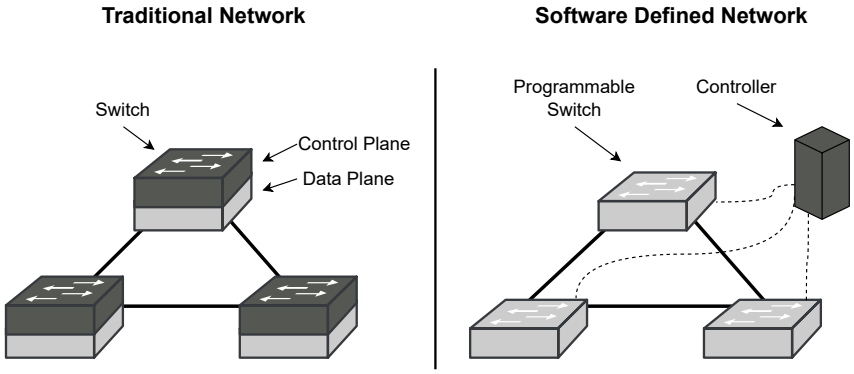


Figure 1: Traditional Network *vs* Software Defined Network

OpenFlow

OpenFlow is the most predominant communication protocol in the SDN space. It was initially developed in 2008 [14], and later released in 2011 by the Open Network Foundation (ONF). Currently, the publicly available version is 1.5.1[1]. This standard interface defines the communication between a controller and a programmable switch over Transmission Control Protocol (TCP). Through this communication, the controller can monitor the device and its port status, but most importantly, it can configure the switch's flow table entries. Upon receiving a packet, the switch identifies the flow rule that better matches that packet, and executes the associated action previously defined by the controller, like dropping it, forwarding it, or sending it to the controller through the OpenFlow communication for further inspection.

The protocol initially only accounted for twelve matching fields, but that number has now grown to a much larger value, allowing to create matching rules witch access to more than forty fields. This allows for very fine-grained control over the matching table entries, providing more flexibility and options when designing control applications. On top of that, OpenFlow has also been developed with extensibility in mind, making it easy to create customizable extensions to add new behavior. As a result, some previous works have also extended this protocol to propose techniques aimed at reducing the energy consumption of network devices [17]–[19].

P4

Typically, network devices are designed *bottom-up*, meaning that their chip is a fixed-function unit, limiting the reconfiguration of the packet process pipeline at runtime. Therefore, new features take a long time to implement, usually requiring an entire chip redesign. Even with standard communication protocols like OpenFlow mentioned in Section 2.1.1, network operators are still limited by the functionalities provided by the vendors' hardware. To address this, the P4 language [15] has been introduced by the ONF, providing a *top-down* approach to configure network devices.

[1] `https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf`

P4 is a domain-specific programming language for controlling the forward planes of network devices, specifying how they process individual packets. Similar to common programming languages, a P4 program is compiled into a target specific binary, where these targets can be either hardware or software. Besides the target data plane specific binary, a P4 compiler will also generate a runtime mapping that allows the controller and data plane to communicate using *P4Runtime*, a specification that defines the Protobuf/Google Remote Procedure Call (gRPC) Application Programming Interface (API) used between controller and targets. The controllers in P4 deployments however, can sometimes take different approaches when compared to more classic SDN, since in some cases, each target has its own controller, placed in the same device and without a wide view of the network.

What sets P4 apart is the ability to control how each individual network packet is processed and the access to shared memory in the target itself. This created the potential to bring computations to the network plane, starting the large influx of research on the subject of programmable data planes in recent years [20]. Even with such flexibility and control, P4 programs are still limited by the functionality that the target provides, especially in hardware solutions, that impose limits on the program behavior to meet the desired performance. Not only that, but P4 is also a *non-Turing complete* language, meaning that some computations are not supported. All these aspects make P4 more suitable to some network areas than others, like Inband Network Telemetry (INT) where it has shown its ability to monitor traffic flows at line rate with virtually no impact on performance.

2.1.2 Energy Flexibility

The evolution on how electrical energy is produced, transported and stored, has changed the way electrical grids operate. Energy can now be produced intermittently, at multiple and distributed sources, it can also be imported from different countries into another and even kept in batteries for later use. All these factors create a constant change in supply and demand values, thus leading to an induced flexibility on how and when energy can be consumed and generated.

The Flexibility Value Chain (FVC) [21] is a system proposed by the Universal Smart Energy Framework (USEF) that covers all aspects of energy flexibility, addressing all the needs and benefits for consumers and producers (*Prosumers*). The system settles on the idea that an active demand and supply at the *Prosumer* level, caused for example by flexible loads, creates the possibility to intentionally shift consumption and generation patterns from the usual and typically expected behavior.

This so called demand-side flexibility can prove to be relevant to many of the participants in the electrical grid. Besides helping to reduce the costs associated with purchasing energy when needed, it can also help to optimize congestion management avoiding grid reinforcements and ensuring grid stability [21]. For this chain to work, an *Aggregator* acts as an intermediate between the *Prosumers* and the grid elements that request changes in energy flexibility. This central entity coordinates and aggregates the flexibility information and exposes it through an API to the *Prosumers*.

However, the most relevant aspect for this work, is how *Prosumers* can internally conduct optimizations to deal with variable energy prices and availability, by managing their consumption and self energy production. Besides, the flexibility presented to the consumers can also have the active energy sources into account, prioritizing renewable energies, thus, sticking to those values can help reduce the environmental impact. Network infrastructures can be considered only as consumers, therefore the focus for these entities is to regulate their consumption following the available energy flexibility. Since a minimum QoS needs to be met at all times, the main goal for new traffic management policies being developed focussed on energy flexibility, is to actively move the network load across different locations, pursuing the best balanced solution between energy flexibility and provided service.

2.2 Recreating Networks

As in many areas of computer science, developing and experimenting with networks is something that cannot be done in live environments, since an error or an incorrect configuration could compromise the infrastructure and lead to a complete system failure. Thus, many methods are used to recreate networks and their behavior, allowing for safer and more flexible test environments. These solutions can either consist of physical test beds, software programs, or even a mix of both. Solutions that rely on any physical infrastructure, are however outside the scope of this work, so this section focuses on software based simulators and emulators.

The terms *simulator* and *emulator* are many times used interchangeably. However, these two are in fact different, both having their advantages and disadvantages that can impact the development of the simulation environment. Therefore, it is important to look at each one of these in order to find the adequate for the dissertation's goal. As the names partially suggest, the main difference between these two types of tools is how they function. In a general manner, a simulator works abstractly to *simulate* the real network behavior [22], [23]. Meanwhile, an emulator works by copying the real behavior of a network, that is, *emulating* it [22], [24].

2.2.1 Network Simulators

As previously stated, network simulators work by simulating in an abstract manner the real behavior of a network, and this is achieved by swapping real network components for abstractions. Simulators are often used for educational and research purposes being a widely accepted tool in the field of computer networks to help test and develop new network protocols.

Besides relying purely on abstract models, simulators typically work with discrete events. This means that any event or action in the simulated network is queued and processed in chronological order one at a time. In turn, this mechanism alters the simulation time detached from the real one, by advancing the simulation clock at discrete points in time to match the next event being processed [22], [23]. These factors combined, provide more flexibility, experimental control and scalability to network simulators. Moreover,

this internal architecture ensures more reliable and consistent results, since simulations become deterministic. However, this approach to replicating networks also has its disadvantages. On top of the extra complexity required to model real behavior into abstractions, simulators are usually limited in application realism. Using abstract models instead of running real code and Operating Systems (OSs), creates an obstacle to implement real applications (*e.g.*, SDN controllers). Nevertheless, some exceptions exist, where real applications can work outside the simulation but connected to it, as long as the simulator is also bounded to real-time. Furthermore, network simulators can also be divided into two different subtypes, either **Packet** level or **Flow** level, depending on the method they use to simulate network traffic.

Packet level simulators, as the name indicates, are based in network packets. This means that every single network packet in any communication existing in the simulation is also simulated [25], [26]. Such behavior can prove to be an advantage when studying with detail the implementation of new protocols, since it allows conducting network traffic captures, like in a real system. These captures can later be inspected with external software providing better insights on how a protocol performs. Unfortunately, this detail comes at a cost, since packet level simulators will usually present a much lower performance when compared to their counterpart [23], [25]. The duration of simulations and computing resources required tend to increase in proportion to the number of events, causing packet level simulators to perform, in many scenarios, orders of magnitude worse than flow level ones [26]. When recreating large scale Core and Metropolitan networks, considerable amounts of traffic are required to be simulated, thus such performance penalties can be an obstacle.

Flow level simulators, in another hand, were proposed to increase the speed of network simulations and base their simulations just on the concept of flows from one host to another. This implies less detail and accuracy when compared with packet level simulators, however, this simpler model is capable of achieving a higher scale and performance [23], [26], [27]. As a drawback, modeling SDN in flow level simulators can be a difficult task, since SDN allow for a rapid change in the network topology, real time changes in how the packets in a network are forwarded can be hard to model just using flows, especially due to the fact that flow level simulators do not even provide any notion of network ports, which is an important piece of SDN. Besides, measuring IP Performance Measurement (IPPM) metrics in such simulators can be nearly impossible, since for example, variations in the delay of multiple packets cannot be measured accurately.

NS-3

NS-3[2] is a discrete-event, packet level network simulator for Internet systems, and is targeted primarily at research and education. It is an open source project, and most probably the most known and used simulator across the research field. Being the successor of the previous NS-1 and NS-2, was built from scratch using C++, which means it is a mature project aiming at high performance. The project is actively maintained and constantly tries to pursue a solid simulation core based on different Simulation Models, to

[2] `https://www.nsnam.org/`

the point that the simulations can sometimes be interconnected with real-time networks. There are some efforts at the moment to allow the use of real and unmodified applications and even the entire Linux Kernel network stack, within the NS-3 simulator, however, this is something still being tested and evaluated. It also includes support for distributed simulations using Message Passing Interface (MPI), which can be used to either take advantage of multiple local cores or to run distributed simulations in supercomputers [28], [29]. Since it is largely used and maintained by a vast community, it also ensures that this is a valid and reliable simulator. NS-3 is, however, considered a difficult simulator to operate for entry level users, since it does not offer a high level kind of configuration, requiring an understanding of its internal structure to define simulation scenarios using source code.

NS-3 includes by default a module implementing an OpenFlow compatible switch, however, the specification used by this implementation is *version 0.8.9*, which corresponds to a very old and outdated version previous to the official release of the standard. Fortunately, to address this issue, Chaves *et al.* [30] introduced OFSwitch13. This module extends NS-3 with support for OpenFlow 1.3[3], a more recent version of the standard. It introduces both a switch device and controller application interface that can be extended to implement any desirable control logic. It also allows to connect a real SDN controller to the simulation, however, this is somewhat an experimental implementation that has not been completely validated, and also requires NS-3 to use a real-time scheduler to lock the simulation clock with the hardware clock, meaning that simulations need to run in real-time.

By default, some energy models are included with NS-3. These are mostly aimed at wireless devices, which are out of the scope of this work. However, this simulator provides a complete Energy Framework, that allows to implement custom energy models. In addition to these device energy models, NS-3 also provides other two components, Energy Sources and Energy Harvesters. The Energy Sources are used to represent the power supplies on each node (*e.g.*, batteries). As for Energy Harvesters, these represent elements that harvest energy from the environment and recharge energy sources (*e.g.*, solar panels). These two components can also be extended to create custom sources and harvesters that better align with this dissertation's use case.

Another valid option to model energy consumption in NS-3 is ECOFEN [31], a purposely designed energy framework for wired network components intended to help evaluate power consumption in large scale networks. It is based on configurable and real measurements, which makes it possible to model any kind of real device. It can also work at different levels of detail, allowing for more basic calculations considering just ON and OFF states, or more complex ones, that can have into account the energy that processing each Byte costs. This allows running simulations with more or less detail depending on the intended results. The framework also comes with some equipped *green functions*, that allow to turn ON and OFF equipments and/or interfaces to implement other efficient energy techniques, that help develop more advanced energy aware routing algorithms. It has not only been used in studies with relatively big size in topology scales [32], but has also been validated as a very accurate model when compared with

[3] `https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf`

real-life devices [33].

OMNeT++

OMNeT++ [34] is not a network simulator by itself, but rather a simulation library and framework, primarily designed to build network simulators. It is nevertheless another very popular simulation platform in the scientific community. It is also based on a modular approach, however, contrary to NS-3, which includes by default modules to numerous common network paradigms, most of the simulation models for OMNeT++ are independent open source projects that follow their own release cycles. Still, most of these frameworks use the INET Framework [35] as their foundation, since this one offers models for the Internet Protocol (IP) stack and it is maintained by the OMNeT++ team. These models tend to be made up of smaller modules or components also written in C++, and put together using a high-level language, nominated NEtwork Description (NED)[4], also used by OMNeT++ to describe topologies. This language is considered simple and high-level, allowing to describe topologies in a more simplistic manner when compared with for example NS-3. As for OMNeT++, an OpenFlow extension has also been developed, based on the INET framework. The initial version is no longer maintained, but there is however an active *fork*[5] that also includes increased compatibility with other OMNeT++ modules. Although being currently maintained, this module has yet only support for OpenFlow version 1.0, with more recent versions planned to be introduced.

The INET Framework for OMNeT++, also includes a power model to help model energy consumption of devices. Similar to the NS-3 built-in models, these are divided into three components. The first ones are Energy Consumption models, which describe the consumption of a device over time. Second, the Energy Generation models, that are used to simulate devices that generate energy over time. And lastly, the Temporary Energy Storage models, that simulate external storage power sources. The framework provides some built-in versions of these models, subdivided into simpler ones, that just manage energy and power quantities, and more realistic ones that can deal with different currents and voltages. Once again, these built-in models are mostly aimed at wireless devices, but they are however also extensible, and can be adapted to better integrate with this work.

SimGrid

Another framework to develop simulators is SimGrid [36]. It provides ready to use models and an API to simulate popular distributed computing platforms, including Wide Area Networks (WANs). These models have also been theoretically and experimentally evaluated, granting decent accuracy when compared with their real counterparts. SimGrid is also at the moment, in its third major version, where the simulation engine was completely rewritten to achieve better modularity and performance. Thanks to a flow level simulation approach, SimGrid is scalable, fast, and has a low memory footprint [25]–[27]. It is also very versatile when writing simulators, since it allows using C++, C, Python or even Java as programming

[4] `https://www.ewh.ieee.org/soc/es/Nov1999/18/ned.htm`
[5] `https://github.com/CoRE-RG/OpenFlow`

languages. Moreover, it allows describing and configuring topologies using a high-level markup language, more specifically Extensible Markup Language (XML).

SimGrid is also distributed with a few plugins containing energy models. One of those is used to estimate energy dissipated by Wi-Fi links, which is outside the scope of this work. Meanwhile, the other two are more pertinent, being designed to estimate energy consumption of compute nodes and wired links. Nevertheless, for network devices, and to circumvent the previously stated performance of packet based simulators in Section 2.2.1, another energy model similar to ECOFEN has been proposed for flow based simulators [27], and implemented in SimGrid. When compared with ECOFEN, it produces very similar accurate results, and being implemented in a flow based simulator, it only requires a fraction of computing power and time to do it [27].

Shadow

Shadow Network Simulator [37], [38] is a fairly recent tool built primarily using Rust and C as programming languages. The main goal for this simulator is to leverage simulated networks but still use completely real applications, thus addressing the biggest drawback with common network simulators. It was initially developed to simulate large scale The Onion Router (TOR) networks for research, and this is still the main motivation behind the simulator, nonetheless, its continuous development and increase in popularity has turned this project into a proper generic Network Simulator capable of replicating other kinds of networks.

In order to execute real applications, it works by intercepting system calls and wrapping them, providing full control by the simulator over those same applications' behavior. At the moment, not all system calls are implemented, meaning that not all applications are fully compatible, however, plans to keep expanding this support do exist.

Its simulations are configured using mostly YAML, a typical human-readable configuration language, yet, compared to other simulation alternatives, it still lacks many other features, like any kind of energy model or support for OpenFlow.

2.2.2 Network Emulators

Network Emulators are typically used to test the performance of real networks, and as a proof of concept to applications. They are also often considered the middle ground between simulation and live testing. The advantage of emulators is allowing to easily run real-life applications directly without requiring any modification [24], [25]. However, software based emulators are more limited by the host machine where the emulation is running. By executing real applications in real-time, emulators require the hardware to keep up with the emulation, otherwise, events will be missed and delayed, affecting the experiment's outcome. Although some solutions allow running distributed emulations across multiple hosts, thus splitting the computational load, this approach eventually still incurs into similar limits. Being directly dependent on how the hardware performs, also means that emulators are prone to suffer more deviations between

different test runs. This creates the need for repeating experiments, accounting for the standard deviation, further increasing the time needed to generate concrete results.

Mininet

Mininet[6] is an open source network emulator written in Python and primarily developed to experiment with SDN, offering support for OpenFlow switches out of the box. It uses process-based virtualization to run emulated hosts and switches in a single OS Kernel, where each node is essentially a bash process with its own network namespace. This implementation translates into a lightweight and scalable emulator, capable of handling hundreds of hosts in a single personal computer.

Since switches are emulated using Open vSwitch (OVS), and real and unmodified applications can easily run in the emulator, custom OpenFlow controller logics tested in Mininet, can in theory be moved directly to a real system without any major change required. This ability also allows it to extend real networks, by being easily connected to them. It offers an easy installation with a simple and usable experience, but also provides an extensible Python API to create and modify networks.

CORE

Another well known open source network emulator is the Common Open Research Emulator (CORE)[7]. This project was derived from another open source project called IMUNES[8], and was then released by Boeing to the community. Written in Python, CORE uses Linux namespaces and bridges to create node containers and virtual interfaces respectively. Compared to Mininet, mentioned above, CORE uses a higher number of Linux namespaces per node, providing more isolation between the emulated nodes, but resulting in a considerably inferior performance.

Thanks to its Graphical User Interface (GUI), this emulator is easy to use and allows to rapidly sketch and configure a network topology, being many times used for educational purposes. Nevertheless, it also provides a gRPC API that allows complete control over all aspects of the emulation. Besides being capable of connecting to real networks by tunneling traffic from and to the emulated environment, CORE also allows running distributed emulations, which means that large emulation scenarios can be deployed across multiple machines and be controlled by a single GUI. Even with lower performance and smaller support for SDN, the simple learning curve that CORE offers, makes it a very popular choice in research and education to rapidly prototype different systems.

[6] http://mininet.org/
[7] http://coreemu.github.io/core
[8] http://imunes.net/

Energy Models

Aside from Mininet-Wifi [9], a Mininet fork focussed on Wireless networks, that does include some basic and limited energy modeling capabilities, none of the previously discussed network emulators in this section has any support for energy models by default, or intends to support it in the future. This can be explained based on the fact that emulators are mostly used to evaluate network protocols, and as proof of concept for applications, not focussing on factors that involve physical network infrastructures.

2.3 Summary

This chapter presented some background on the subject of Software Defined Networking (SDN), followed by an overview of the most relevant technologies associated with this network paradigm. In addition, Energy Flexibility was also introduced, alongside with a short explanation on how this technique can be leveraged to improve energy consumption of network devices. The chapter then proceeded to enumerate the different methods and some tools used to replicate networks. Due to the different advantages and disadvantages that each individual tool presents, selecting the right(s) one(s) for this work is an important step to ensure a viable outcome. The next chapter dives into the considerations behind such choices, and presents the Flexcomm Simulator, the main contribution of this dissertation.

[9] https://mininet-wifi.github.io/

3

Flexcomm Simulator

This chapter introduces the Flexcomm Simulator, a novel simulation environment proposed by this work and built on top of NS-3. It details the different considerations made during the development of the tool and its architecture, followed by the description of the components built and modified to meet the objectives previously mentioned in Chapter 1.

3.1 Considerations

Having into account the different aspects detailed in Chapter 2, this work is based on a network simulator. The flexibility and versatility of this type of tool, makes it the most appropriate instrument for this dissertation's goal since it allows more control over all the aspects of the simulated network. Thus, it facilitates the implementation of novel and customized behavior in different network components, and to alter how some protocols work. All these aspects contribute to a better framework to develop and propose new strategies without limitations. The large size and scale of networks which this tool is intended to deal with, it also favors the use of simulators, since being driven by logical time, it allows for time dilation, meaning that simpler simulations can be executed in less time that would take in real life. Meanwhile, more complex simulations can take more time without impacting results accuracy, thus, not requiring powerful hardware to keep up like in emulation tools.

As for the particular simulator used, the Flexcomm Simulator is constructed on top of NS-3, detailed in Section 2.2.1. Choosing a packet based simulator can result in less performance compared to a flow based one, however, there are two main reasons behind this selection. First, this trade-off allows studying with greater detail the impact of routing policies in networks' QoS, and secondly, NS-3 is a very flexible simulator that can be extended with any desirable custom logic. Besides, NS-3 is still regarded as an optimized tool, capable of accurate results. It is also widely accepted in the research community with multiple implemented and validated models. The large community around it, also provides a large support and extensible documentation, both valid resources to help in the development of this work.

From a SDN standpoint, OpenFlow is leveraged by the Flexcomm Simulator, resorting to the OF-Switch13 module, as the primary method to develop this kind of routing strategies. Compared to P4,

OpenFlow is a more developed standard with more established grounds. Additionally, the concept of a centralized entity with a global view of the network, that observes the fluctuations in the energy flexibility and orchestrates the changes required to meet the targets, also better fits in the paradigm style of OpenFlow SDN, with similar approaches previously developed [39], [40].

Finally, all the energy consumption related logic is built upon the ECOFEN module. Providing the building blocks for nodes and net devices' energy models.

3.2 Architecture overview

The entire Flexcomm Simulator is built with NS-3.35 at its core, handling the simulations for all the network and communication stacks. Specific functionality of this particular work is either introduced by new modules or existing modified ones. As shown by Figure 2, the workflow behind using the simulation tool proposed by this work, first consists of creating the configuration files that define a particular simulation scenario. Those files are then interpreted by a parser component, that translates the configuration into a NS-3 simulation. During simulation runtime, multiple logging files are generated, that can later be processed generating the statistics on how the simulated network behaved, allowing to study and compare the impact of the routing strategies being tested.

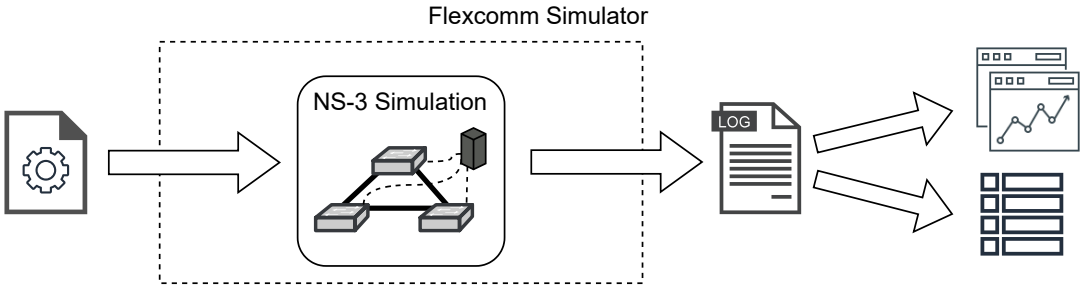


Figure 2: Flexcomm Simulator workflow

In the Flexcomm Simulator, and as illustrated in Figure 3, a NS-3's simulation scenario is composed by five main elements:

- Hosts
- Switches
- Links
- Applications
- Controller

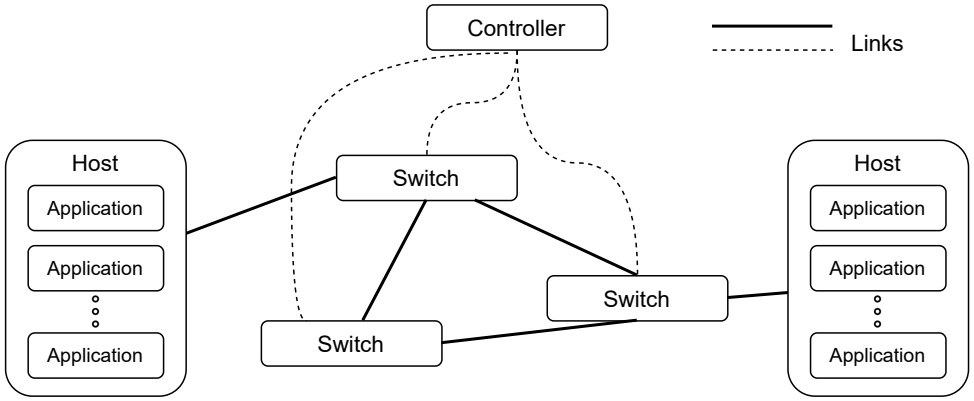


Figure 3: Flexcomm Simulator main simulation elements

Both Switches and Controller, provided by the OFSwitch13 module, map directly to their real-life representation. Meaning that these are the network devices that together, constitute a network and its administrative domain. The Hosts on another hand, are mostly an abstraction required by NS-3. These do not necessarily represent an end device as the name would indicate. In some simulation scenarios, even if just the core of a network is being replicated, a Host is still required to be in place on each network end from where traffic can be addressed or originate from. This happens because Applications, that are used to generate and receive network traffic, can only be part of a Host. At last, Links represent the connections between the multiple nodes in the simulation.

Tracing System

By having NS-3 as the base of the simulator, during runtime, all the modules can directly communicate and exchange data by calling each other's methods, like in any typical object-oriented program. Nevertheless, this creates dependencies between modules that can later affect the expansion of the simulator. Tackling this, NS-3 also offers a very powerful feature heavily utilize in the Flexcomm Simulator. The *Tracing System* consists of tracing sources and tracing sinkers, along with a mechanism to connect them to each other. Trace sources signal relevant events that happen in the simulation and provide access to data related to it. Meanwhile, trace sinkers act as consumers for those events. To hook a given sinker into a source, a callback function is used, preventing the need for inter-module dependency. Since these two systems are completely independent, multiple sources can exist in the simulation without any sinker listening to them.

Although adding trace sources represent a very small execution overhead, increasing the number of sinkers also increases the number of functions and code being executed, therefore an increase in simulation workload. Nevertheless, the ability to only listen to selected and relevant events provides a more efficient mechanism, making the *Tracing System* regarded as the best method for inter-module communication and to retrieve information from a NS-3's simulation.

3.3 Parser

This section describes the Parser component developed as part of the Flexcomm Simulator in order to facilitate the loading of new topologies and respective configurations. It also details the development of a new custom configuration syntax language to accommodate all the required features and functionality.

3.3.1 Existing configuration methods

As mentioned in Section 2.2.1, NS-3 does not offer any type of high level configuration language to configure simulations. It is instead expected that all the configurations needed, like the topology layout for example, to be described when writing the program/script for the intended simulation use case. This can present an obstacle to less experienced users, either in programming, or in the NS-3 ecosystem itself. In the past, NS-3 has offered a *Topology Generator* GUI as an individual project[1] with the intent to facilitate this process. However, since this tool would generate the correspondent C++ code required, this meant that any small change to the simulation configuration, would still require modifications into the source code, or to regenerate everything from the beginning. Not only that, but generating C++ specific code also meant that with new versions of NS-3, incompatibilities could arouse, and previously generated configurations would no longer be valid.

Although such tool has been abandoned and is no longer maintained, NS-3 still offers *Topology Input Readers*. These readers are modules that parse input files containing the topology information and create it in NS-3. This approach of parsing files at runtime also carries the advantage of not requiring compilation every time a change is made, thus enabling a faster iterative process during experiments. NS-3 offers readers based on three topology formats, namely:

- Inet [41]
- Orbis [42]
- Rocketfuel [43]

These formats are, however, not designed to manually describe existing networks or associated configurations. Inet and Orbis are used by topologies generators, programmed to create new and hypothetical network layouts based on input values and typical know topologies. As for Rocketfuel, this tool is designed to map and survey real Internet Service Provider (ISP) topologies.

Topologies readers can also be extended, introducing support for other already existing configuration languages, or for custom ones as desired. This process, however, still has some drawbacks, since these readers are only intended to load the networks' layouts and no other configuration. This implies that changes related to the simulation's configurations are still required to be made in the program's source code.

[1] `https://github.com/idaholab/Topology_Generator`

3.3.2 Custom configuration format

Based on the limitations presented in Section 3.3.1, this work introduces a new configuration format alongside the Flexcomm Simulator to address the issues described and to support additional functionality. The goal with this new syntax is to provide an easy to use and non-verbose format, that enables not only to describe the layout of the topology being simulated, but also to configure multiple aspects of the simulation without requiring knowledge of the internal structure of NS-3. This allows for better compatibility, since by being independent of the simulator, the configuration files are still valid even with internal changes in the simulator's API. On top of this, being completely agnostic to the simulator, means that the configuration files can also be used with other tools to replicate the same experiments.

This new configuration method uses files written in a modern configuration language, more specifically, Tom's Obvious Minimal Language (TOML)[2]. This language is easily human-readable and less verbose than other alternatives. Is also designed to map unambiguously to a *HashTable*, thus making it simple to parse into data structures in any programming language. With a wide range of parsing libraries, and the ability to support comments to annotate any configuration, this is the best candidate to meet the requirements detailed above.

In order to keep the configuration files simple, a simulation configuration requires four distinct files, each one responsible for different simulation components. The files *nodes.toml* and *links.toml* describe the network topology, by detailing the existing nodes and links respectively. Then the *applications.toml* holds the different traffic generators and their characteristics further detailed in Section 3.9. At last, *configurations.toml* contains generic simulation configurations, like the duration of the simulation and the different kinds of data that should be calculated and logged. Further details and examples can be found in Annex I.

Optionally, a simulation scenario configuration can also contain three other files. Two of them, are JavaScript Object Notation (JSON) files containing energy consumption estimates and energy flexibility values respectively. These files are used by the Energy API module further presented in Section 3.7. The third one contains programmable link failures to be simulated that are additionally detailed in Section 3.6.2.

3.3.3 Parser module

To handle the new configuration specification established in Section 3.3.2, the Flexcomm Simulator includes a new parser component (*Parser*) for NS-3, responsible for reading the configuration files and instantiating the correspondent simulation. This element is built with *TOML++*[3] and *JSON for Modern C++*[4], two open source C++ libraries for parsing TOML and JSON respectively.

[2] `https://toml.io/en/`
[3] `https://marzer.github.io/tomlplusplus/`
[4] `https://github.com/nlohmann/json`

By being the main program's entry point, this module is responsible for orchestrating the entire preparation of the NS-3's simulation. When starting a simulation, the parser component will look for the appropriate configuration files, and process them setting up the simulation accordingly. Essentially the *Parser* is responsible for translating the information described in the configuration files, into NS-3 specific logic. It creates and configures all the main simulation elements described in Section 3.2, as well as their connections and associations. Furthermore, it coordinates how auxiliary modules (*e.g.*, OFSwitch13, ECOFEN), mentioned in the upcoming sections, are integrated with the Flexcomm Simulator.

3.3.4 Naming System

NS-3 provides a *Naming System* for its simulations, that optionally allows assigning a name to any simulation's object. The Flexcomm Simulator leverages this system to name every single simulation main element enumerated in Section 3.2. This requires each element to have a unique name in the configuration files, however, this feature benefits the simulation tool with an easier method to index the different components being simulated. Additionally, it helps map each data source with the corresponding source object in the output files described further in the next sections. The names are implicitly assigned to each object based on the correspondent entry key in the TOML files. More details on how to name each element are available in Annex I.

3.4 OFSwitch13

The OFSwitch13 module provides the Flexcomm Simulator with the programmable switches and the controller classes that compose an OpenFlow SDN. The switches' implementation consists of a highly detailed model, with full support for OpenFlow 1.3, and where even the time required to find a matching routing rule in the Ternary Content-Addressable Memory (TCAM) can be simulated. As for the controller, some examples are provided, but any desirable custom logic can be implemented, either directly in an internal NS-3 controller, or by connecting a real controller to the simulation. As stated in Section 3.3.3, the *Parser* component ensures that the right OFSwitch13's elements are instantiated in the simulation, creating the switches, controller and their connections following the configurations. Although a complete and established module, this work introduces extra changes to OFSwitch13 to improve its usability and overall compatibility with the Flexcomm Simulator.

3.4.1 P2PEthernet

Since an OpenFlow switch expects to receive packets encapsulated in an Ethernet Frame [44], OFSwitch13 was developed using the Carrier-Sense Multiple Access (CSMA) links module provided by NS-3, because these are the only existent ones that encapsulate packets using this method. The problem with this particular CSMA implementation is that it does not properly fit modern Ethernet, by not supporting full

duplex communication. Implementations for full duplex CSMA links have been proposed into NS-3 but have never been actually implemented.

NS-3 also provides a link model implementing the Point-to-Point Protocol (PPP) [45], that has support for full duplex. This model is commonly used for generic NS-3 simulations, where a point-to-point abstraction is typically enough. However, this implementation encapsulates packets using the PPP header, a different transport layer protocol. Since the connections between switches in networks simulated in the Flexcomm Simulator can be simplified to point-to-point links, a new model called *P2PEthernet* is proposed by this work. This module borrows most of its functionality from the already existent PPP one, but instead, it encapsulates packets using an Ethernet frame, allowing for these links to be integrated with OFSwitch13.

3.4.2 External Controller

As mentioned in Section 2.2.1, the OFSwitch13 module contains experimental support for external SDN controllers (*e.g.*, ONOS [5], Ryu [6]). This implies using a real-time simulation implementation in NS-3, which means that a simulation runs synchronized with the *Wall Clock* of the host machine. By doing this, the simulator now suffers an identical drawback of a typical Network Emulator, where if the hardware is not capable of keeping up in real-time with the number of events, the simulation is no longer capable of providing accurate results. Nevertheless, as in emulation, by keeping the data rates reasonably low it is possible to test real controllers directly with the Flexcomm Simulator, providing a valuable method to validate the new strategies for real use case deployments.

To connect the simulated network with a real SDN controller, the OFSwitch13 module uses NS-3's *TapBridge* implementation. This bridge is designed to integrate real network components into simulations, by creating an illusion that a NS-3's net device is a real one. As shown in Figure 4, when using an external controller, a *ghost* controller node is instantiated into the simulation and connected to the switches. This node, in turn, communicates with a Tap Bridge, that on its own, and using Inter-Process Communication (IPC), exchanges messages with a Linux Tap Device, creating the link between the simulated switches and the real controller.

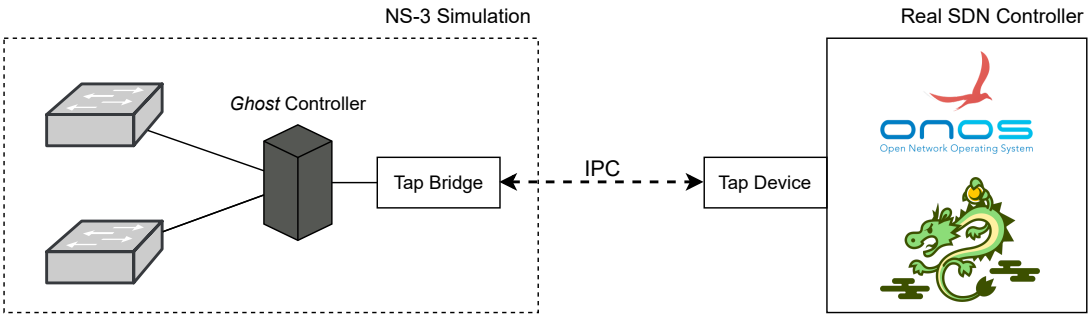


Figure 4: External controller architecture

[5] https://opennetworking.org/onos/
[6] https://ryu-sdn.org/

Any controller with support for OpenFlow 1.3 should work directly with the simulation environment, although some differences exist when compared with physical networks. Some of these changes are tied to OFSwitch13 implementations, and should not interfere with any normal controller operation, namely the lack of auxiliary connections between controller and switch, the lack of encryption in the OpenFlow channel and the out-of-band controller connection, meaning that a dedicated network is used by the controller and switches to communicate with each other. These changes tend to be invisible for controllers, and do not require any modification for programs used with or without the simulation environment. However, some modern controllers also resort to Link Layer Discovery Protocol (LLDP) to survey the network and determine its topology configuration. Yet, this protocol is not supported by the OFSwitch13 module, or the NS-3 simulator itself, thus the use of this mechanism has to be disabled in controllers that require it. Instead, the composition and layout of the network should be provided to the controller using a configuration file that can be generated with one of the scripts described in Section 3.12.

3.4.3 Modified OpenFlow version

With the NS-3's *Tracing System*, detailed in Section 3.2, the energy consumption and Central Processing Unity (CPU) usage values can be easily obtained from within the NS-3 itself. This means that when developing other NS-3 components, such as internal controllers, this data can be retrieved through method invocations. However, external applications require real protocols and real communications to exchange information with the simulation environment. Leveraging the OpenFlow ability of creating new messages to implement new functionality on top of the protocol standard, this works proposes a new OpenFlow extension. Along with the Flexcomm Simulator, a modified OpenFlow version is included, providing the means for an external controller to access energy consumption and CPU usage values from switches.

The extension then brings two new messages to OpenFlow. A request message used by the controller to inquire the switch on its energy consumption and CPU usage, and a reply message used to report back those values. Since the OFSwitch13 module uses its own OpenFlow library, compiled alongside the module, these changes are applied directly in that library, without interfering with any system related dependency. Modifications are also included on the NS-3 side of the module to receive and interpret the request and to construct and send the answer with the correct values. The CPU usage is obtained directly from the OFSwitch13's switch and the energy consumption from the ECOFEN module, this ensures that any values obtained by an external controller are identical to the ones obtained by internal controllers.

To use this custom OpenFlow version, it is required for external controllers to be slightly altered in order to support it. Further details on this process are included within the Flexcomm Simulator's manual [7].

[7] Available at `https://github.com/RuiCunhaM/Flexcomm-Simulator-User-Manual`

3.4.4 Dijkstra Controller

Included with the OFSwitch13 module are three examples of internal controller applications. The *tunnel-controller* and the *qos-controller*, are only applicable to very specific scenarios used by the module's examples. The third one, *ofswitch13-learning-controller*, is a more generic controller that can be used in any network topology. It works as an L2 learning controller, meaning that learns Media Access Control (MAC) addresses as the traffic goes by. This example, however, is intended for when the entire network being simulated belongs to the same subnet. In most cases, this will not be the case, since SDN can connect multiple subnets with each other.

To provide a more comprehensive solution, this work presents the *DijkstraController*. This controller included within the Flexcomm Simulator uses the *Dijkstra* algorithm to calculate the shortest paths between all the nodes in the simulation, and installs the appropriated L3 rules in each switch to ensure connectivity between all hosts. Being an internal controller, it has access to the entire network layout, meaning that all rules for a given switch are installed when that device first connects to the controller.

The *DijkstraController* also doubles as an example to help develop other internal controllers for the Flexcomm Simulator, since it demonstrates how different simulation data can be accessed and used at runtime, to help in routing decision-making.

3.5 ECOFEN

ECOFEN empowers the Flexcomm Simulator with the ability to model the energy consumption of switches. This component includes energy models that can be associated with each switch in the simulation and their respective network devices providing an estimated energy consumption based on their workload. Similar to other simulator elements, and as described in Section 3.3.3, the *Parser* component is the one responsible for attaching the correct energy model to each switch according to the simulation configuration. The development of ECOFEN has, however, stalled in recent years, leaving some features incomplete and preventing the addition of new ones. Addressing this, this work also proposes new additions and fixes to ECOFEN, updating its behavior and enhancing its features.

3.5.1 Output files

The default behavior used by ECOFEN to report the energy consumption of the devices, is to output that information to the *stdout* using the NS-3's *NS_LOG_UNCOND* method. Although such output can be redirected into a file, this is not ideal, and goes against the Flexcomm Simulator's style of generating logging files that can later be interpreted. To address this, some changes are introduced by this work to the ECOFEN module to create and utilize logging files to report energy consumption of devices. Additionally, considering these files are intended to be parsed and interpreted by other programs, the output format is also improved as shown in Listing 3.1, removing the unnecessary annotations, thus reducing the verbosity.

Moreover, the node's identifier is replaced by its name according to NS-3's naming system to help identify each node when processing the data. These changes combined, help to minimize approximately 50% the output file size. In this new format, for each line, the first column indicates the relative simulation time in seconds when the log was created, followed by the second column with the correspondent device's name, and the last column with the power consumption in Watts.

```
# Previous Output
Time 1.0 Node 0 Conso 500.0
Time 1.0 Node 1 Conso 540.0
Time 2.0 Node 0 Conso 510.0
Time 2.0 Node 1 Conso 550.0

# Current Output
1.0 Sw1 500.0
1.0 Sw2 540.0
2.0 Sw1 510.0
2.0 Sw2 550.0
```

Listing 3.1: ECOFEN reduced verbosity example

3.5.2 Load Based Energy Model

As referred in Section 2.2.1, the ECOFEN module includes multiple energy models, both for the chassis of a given device, or each individual network interface. These models have proven to be quite accurate [33], however, they require a high level of detailed parameters about the devices being simulated, for example, the energy consumed, in nanojoules, by each byte that is processed in a network interface. Although these values can technically be measured, this is not how most manufacturers detail the energy consumption of their devices. Typically, the energy consumption of a device is presented in discrete values dependent on the computational load, for example, the amount of Watts consumed when the device is working at 50% its capacity.

Since such detailed values are not disclosed, and conducting such measures in large scale heterogeneous networks would be a lengthy and difficult process, this work proposes a new energy model for ECOFEN. Using the OFSwitch13 module's ability to estimate the CPU usage level of a switch at any given time, based on the number of packets being processed and the switch's predefined capacity, the new energy model takes advantage of this to produce an energy consumption approximation. The model can be configured with two or more discrete percentage levels and the respective consumption in Watts. The energy consumption is then calculated with the CPU usage reported by OFSwitch13 and using linear interpolation with the closest two previously configured values. The individual CPU capacity for each device being simulated can also be configured into the OFSwitch13 module, therefore being possible to match any real device being replicated.

The ECOFEN module, and consequently the Flexcomm Simulator, allows to configure the time interval used to log energy consumption. For the original energy models included within ECOFEN, this interval has no effect on the values reported. These models work with cumulative values, either counting the time

a device was ON/OFF, or counting the number of packets processed. According to the ECOFEN's user manual [8], the logged values correspond to the average consumption in the last interval and increasing this interval, only causes a smoothing on the power consumption peaks observed, while the total energy consumed stays the same. With the *Load Based Energy Model*, introduced by this work alongside the Flexcomm Simulator, this behavior is however not applicable. Considering that the model averages the energy consumption for an interval, with the CPU usage percentage measured in a given instant, different logging intervals provide different results. Thus, smaller logging intervals benefit the use of this model with more accurate energy consumption estimations.

3.5.3 Energy Models Templates

Since different energy models have multiple configurable parameters, redefining a new one for each device in a network can become a repetitive task when they are similar. Preventing the need to redefine the same model multiple times, the Flexcomm Simulator provides the ability to create Energy Model Templates. These templates can be referenced and reused when defining an energy model associated with a given switch in the simulation. A global configuration file can be used for all the simulations containing all the templates. Some examples are available in Annex I and more details on how to create templates can be found in the Flexcomm Simulator's manual.

3.6 Links

To allow for better manipulation of the links in the simulations, the Flexcomm Simulator integrates new functionality developed in this work. This section details the newly developed components aimed at providing more experimental control over links in the simulation.

3.6.1 LinkList

During runtime, NS-3 lacks methods that allow for easy access to all the links in the simulation. Contrary for example to nodes, which can be retrieved from a global container, the connections between them can only be inquired individually for each host. To address this issue, providing a better system for internal controllers to have access to the links composing the network, this work introduces a new component called *LinkList*. This addition keeps track of all links instantiated in the simulation and supplies global methods for other NS-3 modules to access them.

3.6.2 Link Failures

An important aspect for routing algorithms, is how they are capable of detecting, reacting and adjusting to failures in the network. To help develop new SDN routing strategies that are prepared to deal with these

[8] Available at `http://people.irisa.fr/Anne-Cecile.Orgerie/ECOFEN/ECOFEN-user-manual.pdf`

events, the Flexcomm Simulator provides a mechanism for simulating link failures during simulations. Along with the typical simulation configuration, an extra file (*linkfailures*) can be provided containing programmable changes in link states. Any link in the simulation can be brought down or back up again at any time, simulating a failure or a recovery respectively.

When setting a link state down, NS-3 will start dropping all the packets that should go through that link, as expected. Meanwhile, links connected to OFSwitch13's switches will trigger an event every time their state changes, originating a correspondent OpenFlow message delivered to the controller. This way, SDN controllers are aware of failures in the network and can adapt their routing policy accordingly.

3.7 Energy API

The API provided by the *Aggregator*, mentioned in Section 2.1.2, offers Representational State Transfer (REST) endpoints where energy flexibility information can be retrieved. This data, can either be indexed and aggregated by a unique location identifier, referring to flexibility applicable to a regional zone, or it can be indexed referring to a particular *Prosumer*. For consumers, that are the main focus of this work, energy flexibility is made available as a recommended relative offset compared to the estimated consumption. Based on historical patterns, the *Aggregator* is aware of the average energy consumption, in Watts, that a zone/device is expected to consume during different intervals of the following day. Thus, in the same manner, it can provide consumers across those intervals with recommendations to either decrease or increase their energy consumption for the next twenty-four hours.

Usually, this data is available in the form of vectors with ninety-six positions, where each position corresponds to a period of fifteen minutes during a full day. To ensure controllers can have access to this information and make decisions accordingly, the Flexcomm Simulator includes a proposed model called *EnergyApi*, that mimics the real API by providing methods used by the controllers to access energy consumption estimation and flexibility values.

The module requires, as part of the simulation configuration, two JSON input files, containing the consumption estimate and the energy flexibility respectively, indexed by a unique device or zone identifier. As mentioned in Section 3.3, these files are processed by the Parser component that deals with inserting the data on the *EnergyAPI* class. Depending on the type of controller being used for the simulation, the API can be accessed in two different ways as shown in Figure 5. For internal SDN controllers, implemented directly in C++ into NS-3, the API exposes two static methods, one for retrieving the vector containing the consumption estimate for a given device/zone, and the other to obtain the vector with the energy flexibility. External controllers can have access to an identical API, available through a Hypertext Transfer Protocol (HTTP) server, more realistically resembling the real one. This server is a simple *Flask* program that reads the same JSON input files as the simulator and makes the endpoints available through *localhost*. When an external controller is being used, the external HTTP server is automatically started as a *child* process of the main simulator.

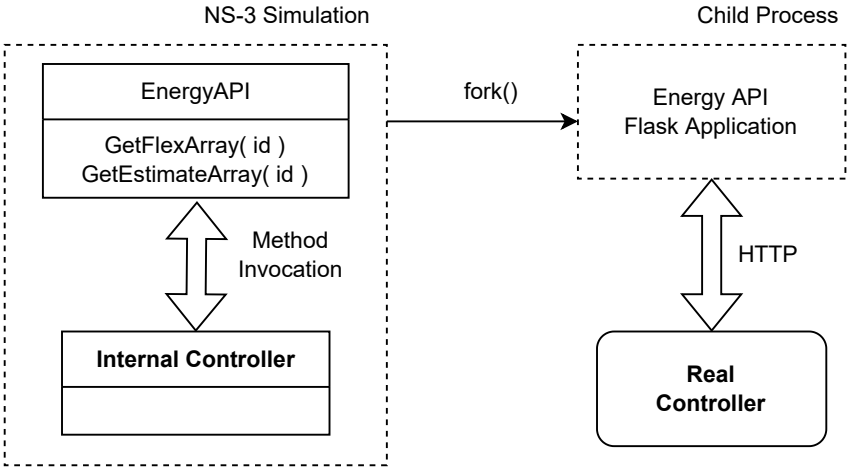


Figure 5: Energy API architecture

3.8 SNMP

Simple Network Management Protocol (SNMP) [46] is an internet standard protocol for collecting information about devices in an IP network. It can also be used to change that information and alter the device's behavior. The most common use is to obtain devices' statistics, like energy consumption and CPU usage. The usage of a custom OpenFlow version as referred in Section 3.4.3, already solves the access to this type of data, however, obtaining such information using SNMP is a more established standard.

This protocol was developed with the goal of providing management access to all devices in a network, independently of their type or manufacture. In order to properly recognize the different values and what they refer to, SNMP works on a basis of nested fields where Object IDentifiers (OIDs) identify each value, and these are by themselves grouped in Management Information Bases (MIBs). This system allows to uniquely identify each value/option, providing an easier way for devices to communicate different available information. Besides some established and standardized MIBs, each manufacturer can also create their owns, leading to a vast number of possibilities. Meanwhile, to solve the issue with different machine types and programming languages, SNMP uses Abstract Syntax Notation One (ASN.1) to define the data structures that compose one of its messages.

All these factors lead to a complex network protocol [9], that has been proposed multiple times as a nice addition to NS-3, since is still widely used in the industry, but was never actually implemented. To circumvent this issue, and to accommodate real SDN controllers that resort to this protocol, the Flexcomm Simulator introduces a cooperative approach with another simulator. The *snmpsim* [10], is a Python written SNMP agent simulator that can act as multiple devices and respond to SNMP request sent by the SDN controller. Since this tool obtains its data by reading text files, it is only required to extend NS-3 with a module that dumps the relevant OIDs and values into logging files, establishing the bridge between both

[9] `https://www.ranecommercial.com/legacy/note161.html`
[10] `https://github.com/inexio/snmpsim`

simulators as illustrated by Figure 6.

Similarly to the Energy API's external server, the SNMP simulator is started automatically as a *child* process of the main simulation program. During the simulation, a *MibLogger* periodically accesses all *mibs* being replicated across all the switches, and dumps their information to the logging files used by the *snmpsimp*. To ensure that the *snmpsim* does not read incomplete information while the Flexcomm Simulator is writing to the logging files, explicit advisory locks are utilized (*i.e.*, flock). This mechanism requires small changes in the *snmpsim* proposed by this work, but guarantees that the data is only written or read by one simulator at a time. The current implementation only allows for one way flow of data, meaning that the controller can only query information about the device, not change it. Still, the system was developed with enough modularity so that any desirable existent or custom MIB can be added into Flexcomm Simulator.

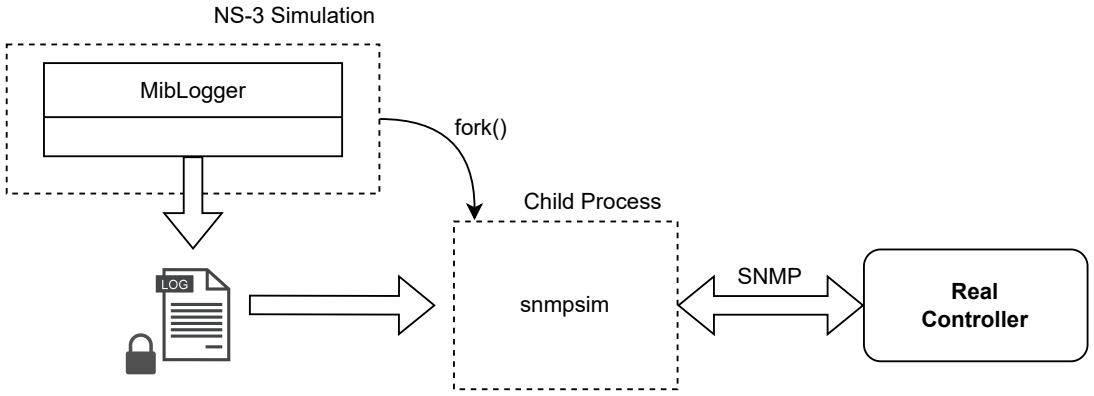


Figure 6: *snmpsim* architecture

3.9 Traffic Generators

An important part of replicating the behavior of real networks for experiments is to replicate their network traffic. Therefore, realistically modeling how traffic behaves inside a network is a requirement for this work. Whit in the NS-3 simulator, network traffic generation is achieved through so-called *Applications*, these are objects associated with nodes in the simulation and that contain sockets, through each, packets are sent and received.

Realistic network traffic can either follow many known patterns [47] or be completely unpredictable depending on the type of application, time of day, protocol and other factors. Thus, to model it accurately, there is no single solution capable of fitting all the needs. To ensure that it is possible to simulate in the best manner the intended traffic, the Flexcomm Simulator makes available different configurable traffic generators.

3.9.1 Constant Generator

The simplest traffic generator provided by the Flexcomm Simulator is called *CONSTGen* and functions as a constant traffic generator. This generator allows to set a constant data rate and generate traffic towards a single destination always sustaining that same rate. It is built based on the *OnOffApplication* part of NS-3 included applications, but disabling the Off period. By default, it generates User Datagram Protocol (UDP) traffic but can be configured to generate TCP traffic and with different packet sizes to better replicate the intended data stream.

Typically, due to congestion control in the network and the applications themselves, a data flow does not maintain a constant data rate during its lifetime. However, this application is intended to introduce constant and static loads in the network, providing a more constant test scenario that is easier to study and evaluate.

3.9.2 PPBP

Ammar *et al.* [48] proposed a realistic internet traffic generator for NS-3 based on the Poisson Pareto Burst Process (PPBP) [49]. This model was developed as a simple but accurate representation of internet traffic. It is based on multiple overlapping bursts, where their length follows a heavy-tailed distribution. Meaning it aims at reflecting aggregated data traffic. As for the generator itself, the authors have demonstrated its accuracy, and that its computation overhead is only moderate.

The Flexcomm Simulator includes a modified version of the original generator created by Ammar *et al.*, altered by this work to support NS-3's 3.35 version and to address incoherences between the generator's source code and the correct mathematical model proposed by Zukerman *et al.*

3.9.3 HTTP

Nowadays, a large part of Internet traffic is still based on the Hypertext Transfer Protocol (HTTP). Being such a major factor in networks, the ability to simulate this type of traffic is an important piece of this work. Thus, the Flexcomm Simulator includes support for a HTTP traffic generator [11] based on the distributions observed by Pries *et al.* [50]. Although this is a simple model, it provides a more accurate behavior on the characteristics of HTTP network traffic allowing for a more in depth study on how new routing policies could affect this type of service. This model includes two NS-3 applications, to distinctly represent HTTP servers and clients, allowing to configure multiple ones in a single simulation.

3.9.4 SINGen

Most traffic generator models found in literature, like the PPBP and HTTP, mentioned in Sections 3.9.2 and 3.9.3 respectively, apply to either specific applications or to a small scale of aggregated flows. Considering

[11] `https://github.com/saulodamata/ns-3-http-traffic-generator`

the large scale of network infrastructures at which this work is aimed to simulate, these models are too granular. Such networks deal with a large amount of traffic and communication flows, thus modeling this aggregated behavior is not achievable with this kind of models. Unfortunately, not only there is no such model for network traffic at this scale, but also ISPs and other entities that possess large scale networks do not typically reveal the precise numbers on the level of traffic they deal with, presenting an obstacle to develop new ones. However, across many articles in the literature, it is possible to denote that aggregated internet traffic at large scale usually follows a recurring pattern that can be approximated by a *sinusoidal* function [12], [47], [51], [52]. This type of behavior is explained by the peak and off-peak periods generated by citizens' daily routine, provoking predictable movements and consequently patterns called Tidal Effects [47].

To provide a traffic generator that better replicates this aggregated behavior, this work proposes a novel generator model for NS-3 called *SINGen*. With this model the Data Rate (DR) of the network traffic at an instant t of the simulation is given by the following equation:

$$DR = A * sin(B * t + C) + Const$$

Where the parameter A controls the amplitude of the function. Parameter B controls the function frequency. Parameter C shifts the function left and right. And at last, the parameter $Const$ represents a constant value. This generator is available within the Flexcomm Simulator and fine-tuning all these parameters allows shaping the function to better fit any intended traffic profile.

3.9.5 Debugging Applications

Moreover, the Flexcomm Simulator also provides two applications to help with diagnosing configuration issues, namely *V4Ping* and *PingAll*. The *V4Ping* application is an already included application in NS-3, and allows to specify a sender and a receiver for Internet Control Message Protocol (ICMP) ECHO requests, thus simulating the behavior of the common *ping* command. Compared to the original one, this application is slightly modified by this work to improve its output. The *PingAll* application, also introduced as part of this work, leverages the *V4Ping* to test the connection between every host in the simulation, ensuring the connectivity between them.

3.9.6 Sinkers

The *Sinkers* objects are still considered NS-3's applications, however, instead of generating network traffic, they act as the receiving end, thus complementing network generators. Asides from the HTTP generator, which provides both the client and the server, and the *Ping* applications that work with ICMP, all the other generators are required to have a receiving socket that accepts the connections, and, in the case of TCP traffic, acknowledges the data sent. Therefore, for each traffic generator application that is created, the

respective sinker is also required to be instantiated in the respective remote host. This process is needed due to the internal NS-3's architecture, however, is entirely transparent to Flexcomm Simulator's users, since it is handled by the Parser component, not requiring any specific simulation configuration.

3.10 Statistics

In order to study the impact of different routing strategies in QoS and energy consumption, it is important for the Flexcomm Simulator to provide methods that allow comparing how multiple aspects of the network change with different policies. This implies keeping track of multiple statistics related to each simulation. To meet this goal, the simulation environment presented in this work generates multiple log files containing relevant data, that can later be parsed and inspected to create a more comprehensive picture of the changes in the network behavior.

In addition to the energy consumption of each device, already reported by the ECOFEN module described in Section 3.5, metrics that provide insights on how the overall load is distributed across the network, and how the QoS is affected are also recorded.

3.10.1 SwitchStats

To understand how changes in the routing policy affect the operation of each network device, besides the energy consumption, metrics that allow measuring the amount of work executed by that device are also relevant. To achieve this, the Flexcomm Simulator proposes a new NS-3 class (*SwitchStats*) to generate logging files conveying this information. Namely, the CPU usage percentage, the number of packets and the number of bytes processed. This new module depends on OFSwitch13, since it leverages its CPU usage report, but for packets and bytes, it takes advantage of NS-3's *Tracing System*, listening to each *packet received* event. The statistics are logged at a configurable time interval and are relative to each individual interval. As shown in Listing 3.2, for each line in the output file, the first column refers to the relative simulation time in seconds when the log was created, followed by the name of the device, its CPU usage percent, the number of processed packets and also the number of processed bytes. These values apply to each logging interval.

```
1.0 Sw1 0.121 18311 7531754
1.0 Sw2 0.345 24350 8531994
2.0 Sw1 0.240 36319 14939016
2.0 Sw2 0.503 35501 12439399
```

Listing 3.2: SwitchStats output example

3.10.2 LinkStats

Similar to the device's statistics, information on the state of the links is also relevant to evaluate the impact of different routing strategies. Overloading a given link can degrade the performance of a network, and

induce losses, thereby QoS. Following the same principle as in Section 3.10.1 this work introduces a new NS-3 class (*LinkStats*) that attaches to each link in the simulation and generates the statistics about their state at runtime. This module also leverages the NS-3's *Tracing System*, by binding a callback to each packet event in the link, allowing it to keep track of the number of packets and bytes that goes through the link at any given interval. The output file format, illustrated in Listing 3.3, follows a similar scheme to the ones shown above. The first column contains the relative simulation time in seconds when the log was created, the second column the name of the link, and the third its usage percentage followed by the number of packets and bytes processed in the fourth and fifth respectively.

```
1.0 Li1 0.523 18303 7531100
1.0 Li2 0.423 15057 7431305
2.0 Li1 0.623 21802 8971080
2.0 Li2 0.323 11497 5674495
```

Listing 3.3: LinkStats output example

3.10.3 FlowMonitor

Besides the impact on physical network components, it is of utmost importance to understand how changes in the routing policy being tested affect the QoS provided by the network. To evaluate this effect, various IP Performance Measurement (IPPM) metrics can be used. NS-3's FlowMonitor module [53] provides a flexible system with this intent. The module works by installing probes in each device, and keeping track of the packets exchanged between them, at the same time classifying the flows and calculating the relevant metrics. This module can be enabled in the Flexcomm Simulator and will generate at the end of the simulation a XML file containing information on each network flow, like the average delay and jitter. This output file can later be parsed by one of the scripts referenced in Section 3.12 to generate compiled and aggregated data as demonstrated in Section 4.3.

3.10.4 Packet Captures

As described in Section 2.2.1, NS-3 is a packet based simulator, allowing it to generate Packet Captures (PCAPs) like in a real system. These captures are a helpful tool to study with further detail the behavior of each individual data flow, since they can later be inspected with other software programs. Because such files can become considerably large, depending on the amount of traffic being recorded, the Flexcomm Simulator allows to activate the captures individually for each link in the simulation, this way, is possible to only capture information in particular points of interest of the topology. In addition, it is also possible to enable captures for the communication between any SDN controller and the switches to help debug OpenFlow communications.

3.11 Distributed Simulations

NS-3's support for distributed simulations, mentioned in Section 2.2.1, relies on partitioning a network topology across multiple Logical Processs (LPs) that can be executed by different processors and MPI to ensure the communication between them. Each LP, or MPI rank, is in charge of simulating a portion of the network. To each simulation node, a system id is assigned, identifying the rank that belongs to. During simulation, if two nodes in distinct ranks need to communicate, the packets are encapsulated and exchanged using MPI messages.

The current implementation, as illustrated in Figure 7, replicates all the simulations nodes across the ranks, independently of their assigned system id. However, if the node does not actually belong to that rank, it is only considered as a *ghost* node abstraction, which helps to simplify internal routing and acts as a MPI gateway. At the moment, this implementation also requires links connecting nodes across different LPs to be PPP links. As Figure 7 further shows, when stabling a connection between two nodes from the same rank, a normal PPP NS-3 link is used, otherwise, a different *remote* variation is selected. This model is the one responsible for encapsulating and sending messages with MPI to other LPs. Moreover, to prevent ranks of receiving messages *from the past*, for distributed simulations NS-3 uses a different simulator implementation, that resorts to conservative synchronization algorithms with lookahead [54].

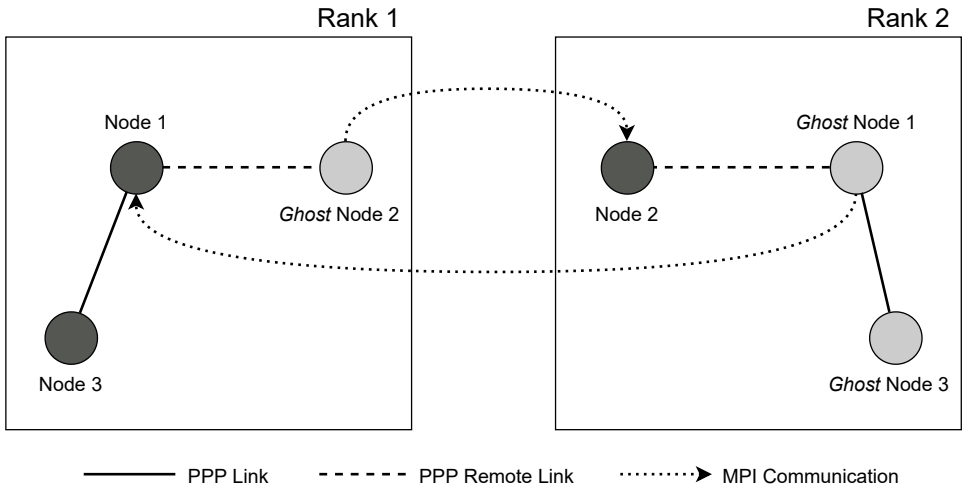


Figure 7: NS-3 MPI architecture

As discussed in Section 3.4.1, OFSwitch13 was initially developed to use CSMA links, thus, the proper support for MPI was never implemented, preventing the module from being compatible with distributed simulations. With the addition of *P2PEthernet* links into OFSwitch13, this became possible, since this new type of link also inherited the ability to use MPI messages to connect nodes in different ranks. To ensure full compatibility of internal controllers and switches with distributed simulations, this work further proposes modifications to the OFSwitch13's module core. With these changes, the OFSwitch13 module fully supports NS-3's MPI implementation, thus, allowing Flexcomm Simulator to run distributed simulations.

Besides the changes to OFSwitch13, the statistics generation modules also require modifications to work with distributed simulations. In a first instance, node specific data can only be collected in the LP where that node is simulated, therefore, ECOFEN's energy models and SwitchStats are installed exclusively for each node in the respective rank ensuring that *ghost* nodes do not interfere with data collection. Moreover, to avoid data races between LPs when writing information into logging files, instead of a single output file for each module as in normal simulations, an individual file is used for each node/link. Since only the rank responsible for the node writes to that file, this guarantees concurrency free access.

Although the introduction of the previously described changes allows the use of distributed simulations within the Flexcomm Simulator, some limitations still apply. The *Parser* component, described in Section 3.3.3, once again deals with all the necessary logic to ensure that the simulation is properly instantiated in all LPs. However, it is required to assign to each node a MPI rank in the configuration files, thus, the network partition needs to occur manually previous to the simulation. Since distributed simulations also require their own internal simulator implementation, external controllers can not be used in these scenarios due to incompatibilities with real-time synchronization. Lastly, the FlowMonitor module described in Section 3.10.3, requires global access to the simulation for classifying network flows, which makes it incompatible with distributed simulations.

3.12 Auxiliary Scripts

To help deal with different tasks related to the preparation of the simulations and interpretation of results produced, a set of example auxiliary scripts is included with the Flexcomm Simulator. These scripts can be divided into two groups, *parsers*, used to process the outputted files produced by the simulator and convert them in a more human friendly format, and *configuration helpers*, used to generate topologies and configurations input files.

Individual parsers exist for each different type of log file produced by the simulation tool. The data produced by ECOFEN can be parsed with *ecofen_parser.py*, the FlowLevel statistics with *flow_parser.py*, *LinkStats* with *links_stats_parser.py* and the *SwitchStats* with *switch_stats_parser.py*. The parsers are capable of producing aggregated data displayed in tables, or generating different plots using a combination of *matplotlib* and *gnuplot* to help illustrate the changes over the course of the simulation. Another script named *difference_parser.py* is also available, which helps illustrate how the actual energy consumption differs from the recommended flexibility. Examples of some of these outputs are shown in Section 4.3.

The configuration helpers are constituted by three scripts. The first one, *init_topology.py*, helps to generate the base structure and boilerplate for the TOML files of a topology with a given number of hosts, switches, links and applications desired. Then, only additional details are necessary to complete the configuration. However, for more common network topologies layouts, it is possible to generate more complete configuration files, with most links pre-configured. The second script, *topo_vis.py*, complements the previous one, by helping to visualize the network topology that a given configuration produces, it

achieves this reading the same TOML files as the simulator and using *networkx* and also *matplotlib* to create a visual representation of the network. At last the third script, *switches_config.py*, also reads the TOML configuration files of a topology and generates a JSON output containing information about the switches and links that compose the network, working as a replacement for external SDN controllers that cannot use LLDP to survey the network as mentioned in Section 3.4.2.

3.13 Summary

Starting with the considerations behind the tools used to develop this work, this chapter introduced the Flexcomm Simulator, the main contribution of this dissertation. Next to an architecture overview of the simulation environment proposed, a detailed description of the different components and modules that compose the tool was presented. Furthermore, the different changes and new behavior developed are explored, providing insights into the capabilities of the simulator. In the following chapter, those capabilities are demonstrated at work, providing a proof of concept for the Flexcomm Simulator and showcasing its functionality.

4

Proof of Concept

This chapter provides a proof of concept for the Flexcomm Simulator, the novel simulation environment proposed by this work and detailed in Chapter 3. The tool aims at aiding the development of new routing strategies that focus on exploiting energy flexibility to enhance energy consumption for large scale networks. Throughout the chapter multiple demonstrations showcase the tool's functionalities and capabilities to help with this process.

4.1 Main Goals

Developing and studying new routing policies at a large scale, requires a simulation environment capable of producing accurate data that allows different strategies to be compared and evaluated. Ensuring a fast pace of development, that allows to quickly iterate over different methods and approaches, is also a requirement for such tool, so the simulator also has to be capable of scaling and dealing with large size test scenarios.

To evaluate the Flexcomm Simulator's capability of achieving these goals, three different experiments were devised with the following objectives:

- Demonstrate the data generated by the tool to compare different routing strategies
- Study the tool's scalability when simulating large networks
- Demonstrate the tool's capacity to help develop energy-aware routing algorithms

4.2 Test Environment and Methodology

The tests presented in this chapter are conducted with a release build for the Flexcomm Simulator. For the majority of the experiments, the simulations were executed in a machine with an Intel i3-4170 at 3.70Ghz and 16Gb of Random Access Memory (RAM).

The tests focussed on MPI are conducted in one computing node from *BOB*, a supercomputer at Minho Advanced Computing Center (MACC), composed by two Intel 8-core "Sandy Bridge" Xeon processors clocked at 2.7Ghz with 32Gb of RAM.

Due to network simulator's architecture, described in Section 2.2.1, it is guaranteed for a simulation to be deterministic. This ensures that when using the same configuration files with the same routing strategy, no matter the number of test runs, the output results are exactly the same. Thus, obtaining simulation data only requires a single simulation run. Execution times, however, can suffer fluctuations based on the underlying host machine conditions, to mitigate this, results presented in the following sections addressing this topic, are calculated as an average of five distinct simulation runs.

4.3 Data Generated

This demonstration aims at showcasing the different information generated by the Flexcomm Simulator, and how it can be interpreted to evaluate the impact of routing strategies being tested.

For this particular demonstration, a simple simulation scenario with just two hosts (*H1* and *H2*) and two *10Gbps* switches (*Sw1* and *Sw2*) is replicated. As shown in Figure 8, Host *H1* is connected to switch *Sw1*, host *H2* is connected to switch Sw2, and both switches are connected between themselves. A *DijkstraController* (see Section 3.4.4) is also connected to each switch, in charge of applying the routing rules. All links in the simulation are configured with a *1Gbps* data rate and a *2ms* delay. A single *CONSTGen* application is installed in host *H1*, generating *TCP* traffic at *100Mbps* towards host *H2* and starting 5 seconds after the simulation begins. For this test, all the statistics sources described in Section 3.10 are enabled, including traffic captures for the link connecting both switches between themselves and with the controller. A five minutes real-time period is simulated.

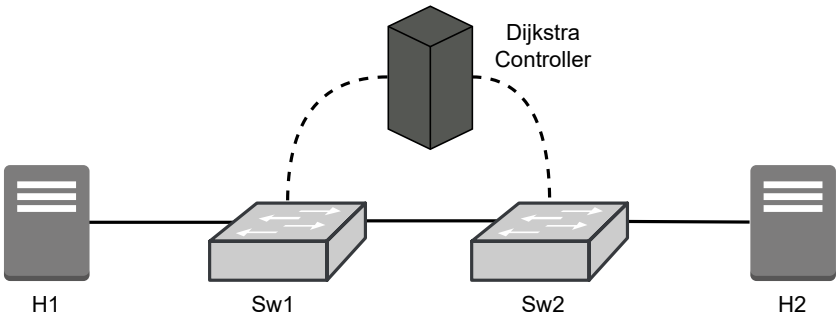


Figure 8: Test scenario

The plots shown in Figures 9, 10 and 11 show the data outputted respectively by the modules *ECOFEN*, *SwitchStats* and *LinkStats*. Figure 9 displays the energy consumption, in Watts, of both switches at discrete points in time. Similarly, the three plots in Figure 10 show the CPU usage, number of packets forwarded and number of bytes processed on those packets by each switch through the simulation respectively.

Lastly, Figure 11 also displays each link usage percentages, number of packets and bytes processed during the simulated time respectively.

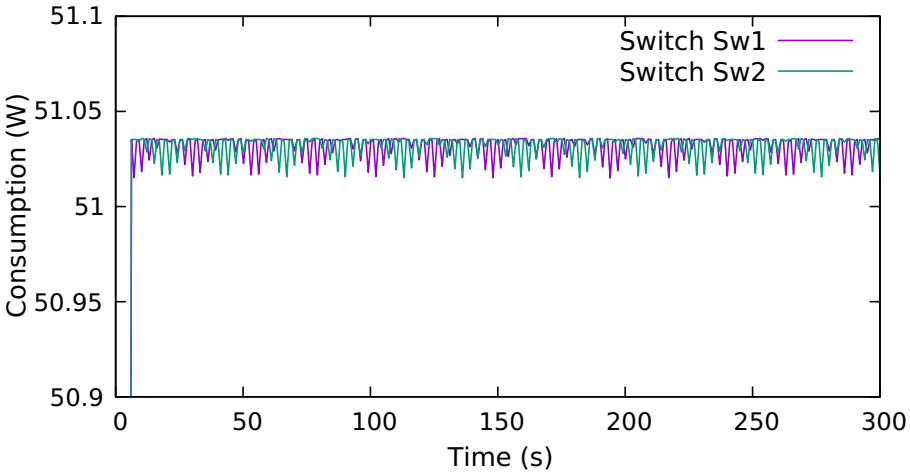


Figure 9: Energy consumption

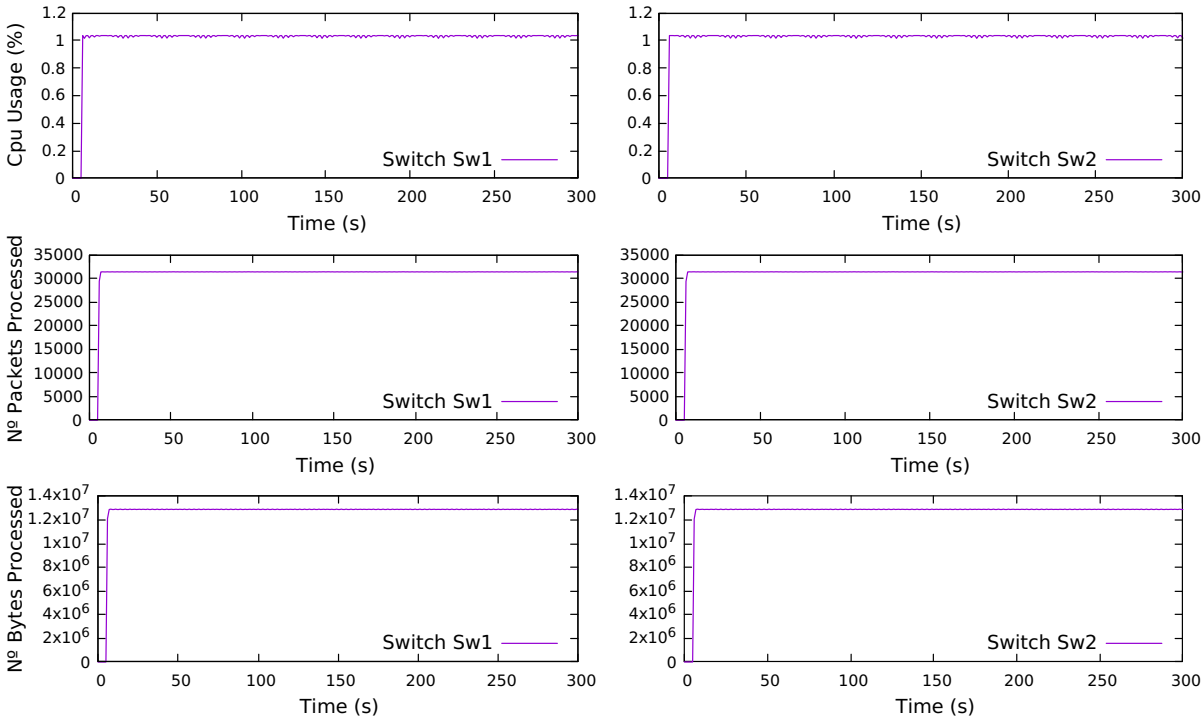


Figure 10: Switch statistics

Both Listings 4.1 and 4.2 show truncated *tcpdump*[1] outputs, obtained from the PCAPs produced respectively from the link connecting the switches between themselves, and the OpenFlow connection between switch *Sw1* and the controller. In Listing 4.1 is possible to observe TCP packets being exchanged

[1]Command-line packet analyzer

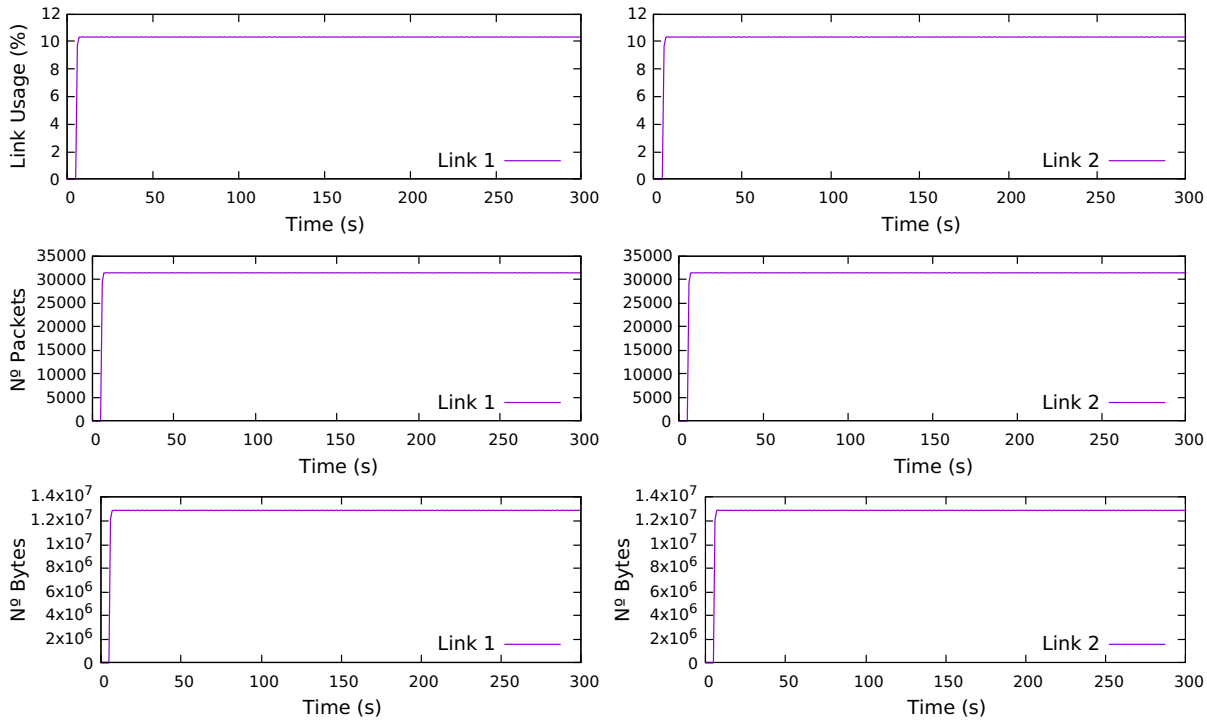


Figure 11: Link statistics

between hosts *H1* and *H2*, meanwhile, in Listing 4.2, the OpenFlow messages sent by the controller installing flow rules onto switch *Sw1* can be seen. Finally, Listing 4.3 demonstrates an output generated with the statistics produced by the FlowMonitor module. In this example, various IP Performance Measurement (IPPM) metrics about the simulated TCP network flow are presented.

```
01:00:06.394356 IP 10.1.1.1.49153 > 10.1.2.1.ndmp: seq 14285313:14285825, win 32768
01:00:06.394378 IP 10.1.2.1.ndmp > 10.1.1.1.49153: ack 14184961, win 32768
01:00:06.394397 IP 10.1.1.1.49153 > 10.1.2.1.ndmp: seq 14285825:14286337, win 32768
01:00:06.394438 IP 10.1.1.1.49153 > 10.1.2.1.ndmp: seq 14286337:14286849, win 32768
01:00:06.394460 IP 10.1.2.1.ndmp > 10.1.1.1.49153: ack 14185985, win 32768
01:00:06.394479 IP 10.1.1.1.49153 > 10.1.2.1.ndmp: seq 14286849:14287361, win 32768
```

Listing 4.1: Link capture

```
01:00:00.005000 IP 10.100.0.2.49153 > 10.100.0.1.openflow: Flags [.]
  version 1.3, type FEATURES_REPLY, length 32, xid 0x6b8b456a
  version 1.3, type BARRIER_REPLY, length 8, xid 0x6b8b456b
01:00:00.005000 IP 10.100.0.1.openflow > 10.100.0.2.49153: Flags [.]
  version 1.3, type FLOW_MOD, length 80, xid 0x6b8b456e
  version 1.3, type SET_CONFIG, length 12, xid 0x6b8b456f
  version 1.3, type FLOW_MOD, length 96, xid 0x6b8b4570
  version 1.3, type FLOW_MOD, length 96, xid 0x6b8b4571
01:00:00.207000 IP 10.100.0.2.49153 > 10.100.0.1.openflow: Flags [.]
```

Listing 4.2: OpenFlow capture

```
FlowID: 1 (TCP 10.1.1.1/49153 --> 10.1.2.1/10000)
  Bitrate:
    TX bitrate: 94295.82 kbit/s
    RX bitrate: 94296.13 kbit/s
  Delay:
    Mean Delay: 6.09 ms
    Max Delay: 6.00 ms
    Min Delay: 6.00 ms
  Jitter:
    Mean Jitter: 0.00 ms
    Max Jitter: 0.00 ms
    Min Jitter: 0.00 ms
  Packet Loss:
    Packet Loss Ratio: 0.00 %
```

Listing 4.3: Flow Monitor parsed output

These figures are meant as example representations, used to demonstrate how the log files outputted by the Flexcomm Simulator can be interpreted and analyzed. This data is, however, generated with generic formats as shown across Chapter 3, enabling it to be processed by different programs, and therefore represented using different methods. Information can also be processed using different approaches that convey different results, for example, instead of energy consumption in Watts, shown in Figure 9, the same values can be used to approximate the total energy consumption of each device during the simulation by calculating the product for each interval.

4.4 Scalability

Studying the Flexcomm Simulator's ability to simulate large scale networks, firstly requires an understanding of how the tool's performance scales depending on the scenario being simulated. Only then, the advantages of the MPI support and the ability for running distributed simulations can be explored. Thus, this section initially provides an overview on how increasing the number of devices and the amount of network traffic being simulated impacts the simulation performance, followed by an analysis of how MPI can be exploited to improve simulation duration.

4.4.1 Simulation size impact

The scale of a given simulation scenario can be affected mainly by two factors, either the number of physical nodes being simulated, or the amount of network traffic also being simulated. These should be considered distinctly, since depending on the particular use case, a large physical network can be simulated with low amounts of network traffic or vice-versa. Therefore, this section is subdivided into two smaller subsections that individually approach these topics.

Physical topology size

To determine how the simulation time is affected by different size topologies, this experimental scenario initially starts as the one described in Section 4.3, with just two switches connected by one link, and two hosts, each one connected to a different switch. Following the same approach, only one *100Mbps CONSTGen* traffic generation application is installed in one of the hosts and a five minutes real-time period is simulated.

Increasing the simulation's topology size, in each incremental step one new switch and a correspondent new host connected to it are added, raising the total number of nodes being simulated in each iteration by two. For a newly added switch, links to all the other previously existent switches are also added, forming a full mesh topology between them. Given that this test focuses on the number of network devices being simulated, the number of applications remains unchanged throughout the entire experiment, meaning that just one *100Mbps* traffic generator is active in each step. This process is repeated until a total number of forty nodes (twenty hosts and twenty switches) is achieved, a representative number for most Metropolitan Area Networks (MANs) [55].

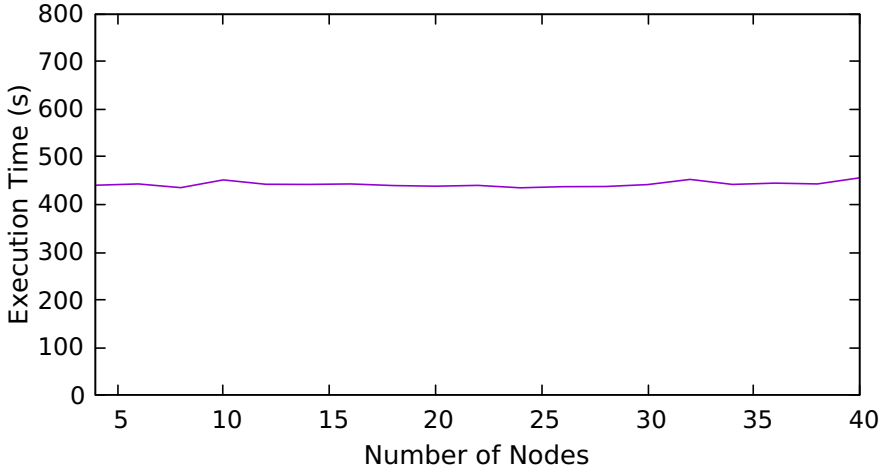


Figure 12: Simulation execution time based on the number of nodes

Figure 12 shows the execution time obtained at each incremental step. As the results demonstrate, the simulation time stays basically constant as the total number of simulated nodes increases. This leads to the conclusion that the number of hosts, switches and consequently links that compose the network topology being simulated does not impact the time required to run the simulation. Large topologies can still lead to larger configuration files that can affect the time required to parse them, however, thousands of nodes require just seconds to be parsed, a negligible time compared to simulation duration.

Simulated network traffic

After establishing that the total number of nodes being simulated does not impact the simulation time significantly (see Section 4.4.1), for this test, the topology size is fixed at forty nodes. The topology

configuration follows the same scheme as in the previous test, with twenty switches connected in a full mesh topology, and a correspondent host connected to each switch. Once again, the test starts with an *CONSTGen 100Mbps* application installed in one of the hosts. On each iteration step a new application is added, repeating the process until a total of twenty identical applications is achieved, ensuring that network traffic is generated and received by all the hosts in the simulation.

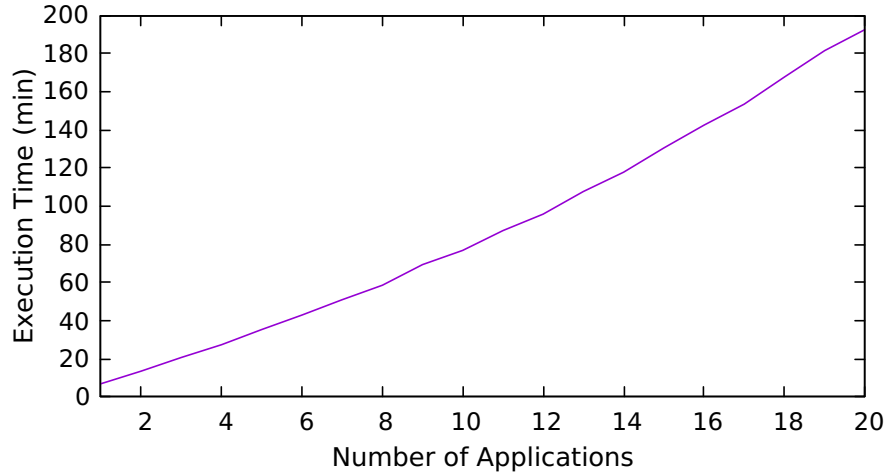


Figure 13: Simulation execution time based on the number of applications

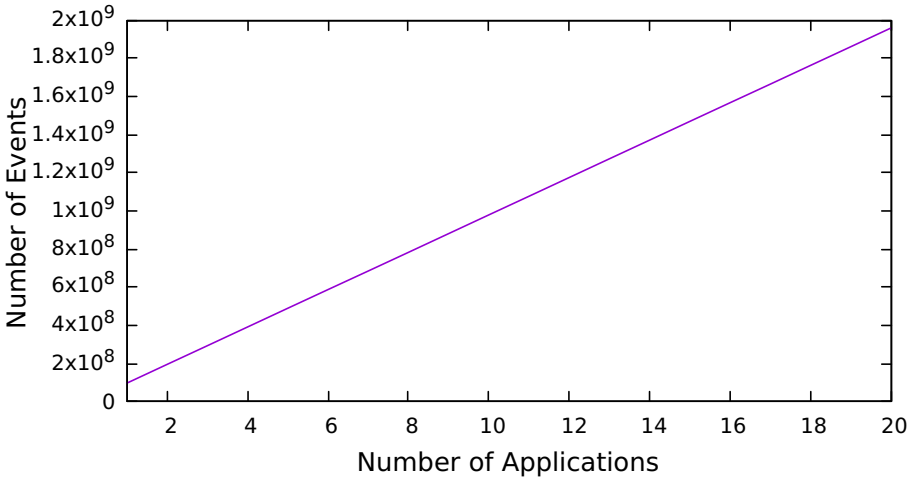


Figure 14: Number of events based on the number of applications

The simulation execution times presented in Figure 13 show an evident increase that follows the number of applications being simulated. Similarly, as shown in Figure 14, the total amount of events processed by the simulator also increases with the numbers of applications. This behavior further corroborates NS-3's packet level architecture described in Section 2.2.1. By replicating each packet in a network flow, each send, receive and forward actions are treated as individual events in the simulator, therefore, an increase in the amount of traffic being simulated, also causes an increase in the number of events that need to be processed.

This way, it is possible to conclude that when dealing with large scale networks, the most relevant aspect that affects the time required to run a simulation is the amount of traffic being replicated. Since each network packet translates into more simulation events, the increase in simulation duration is not related to the total number of individual simulated flows, but to the cumulative data rate across the entire simulation.

4.4.2 Distributed Simulations

The Flexcomm Simulator supports distributed simulations using MPI, allowing for the use of multicore processors or High Performance Computing (HPC) infrastructures to help reduce simulation duration. This test aims at demonstrating how distributed simulations can improve simulation time. In this scenario, a *fat-tree* network topology represented in Figure 15 is recreated. Data centers commonly deploy this kind o topology thanks to its efficient communication, and recently, SDN approaches that also target these infrastructures have been proposed. The simulated network is composed of sixteen hosts, representing servers in racks. Each host is connected to one of eight edge switches, which in turn, are connected to two of eight aggregation switches. At last, each aggregation device is connected to two of the four core switches. To simulate traffic, eight *CONSTGen* applications are installed. In a first scenario (*Scenario 1*), applications are installed generating traffic just within racks, meaning that the traffic is exchanged just using the edge switches. In the second scenario (*Scenario 2*), the applications are configured to generate traffic between racks, therefore, across the entire data center.

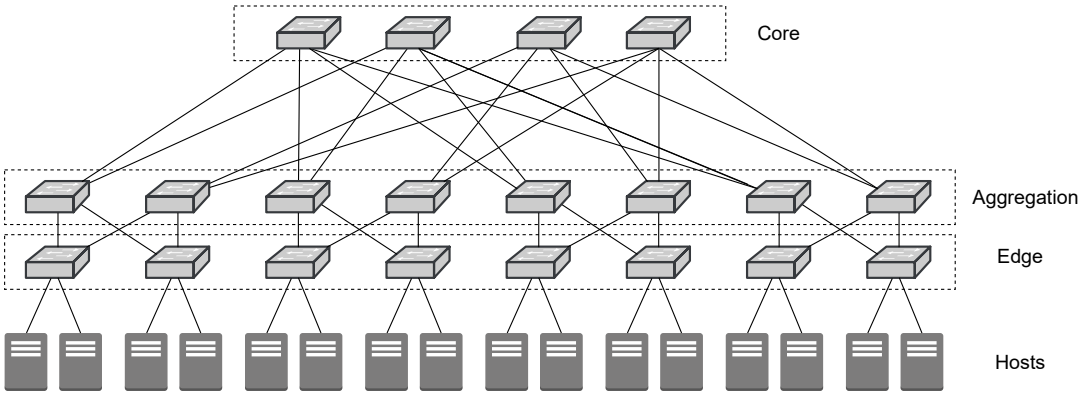


Figure 15: Fat tree topology

During the experiment, the number of MPI ranks is increased, from just one to sixteen. The topology is increasingly partitioned in *vertical slices*, until in the last iteration, where a different MPI rank is assigned to each host. Figure 16 shows the obtained speedups against the expected ones when increasing the number of LPs executing a simulation. Ideally, the speedup should increase proportional to the number of MPI ranks that share the total computational load, meaning that two ranks should take half the time to execute the simulation, three ranks one third and so on. Although both scenarios demonstrate a reduction

in simulation time when increasing the number of MPI ranks, the real speedup is noticeably lower than the expected one.

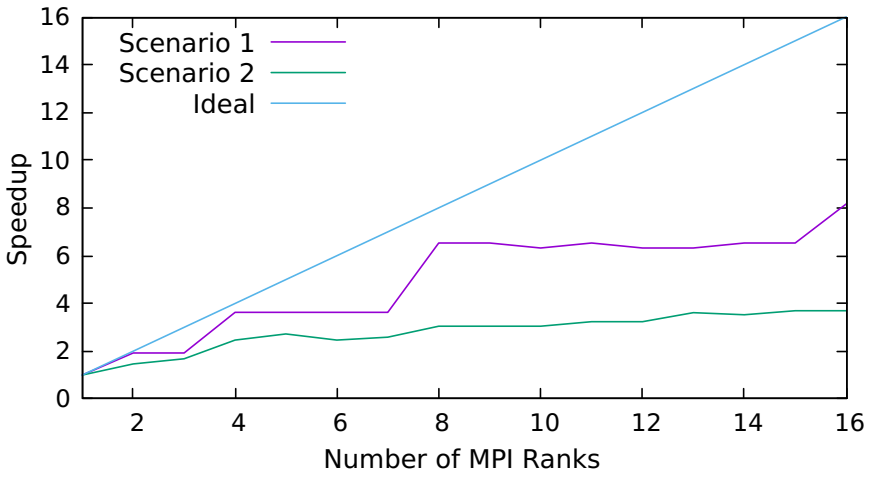


Figure 16: Simulation execution time based on the number of MPI ranks

Furthermore, in both scenarios the number of nodes and the total amount of network traffic replicated is the same, but *Scenario 1* clearly shows a better scaling when compared with *Scenario 2*, which stagnates at around three times speedup when reaching eight LPs. This behavior results from NS-3's MPI implementation described in Section 3.11 and the need for an optimal topology partition that isolates communication clusters in each LP, thus, minimizing the need for MPI communication. In *Scenario 1*, by confining the traffic to hosts at just two hops distance, the topology can be divided further until actual traffic started being exchanged through MPI messages. Contrarily, in *Scenario 2*, this effect is observed much sooner, preventing the simulation of furthering scaling.

This allows to conclude that the performance gains obtained with distributed simulations heavily depend on how the simulated network is partitioned. Ideally, each MPI rank should be as independent as possible of any other, thus reducing the MPI communications and benefiting from splitting the computation load. Since at the moment the partition process is manual, some knowledge of the simulation traffic pattern is required to determine the optimal solution for setting up the configuration. However, in some scenarios such solution can sometimes not be possible. Depending on the simulated network, isolated *communication clusters* can be nonexistent, preventing a network partition that can benefit from distributed simulations.

4.5 New routing policy

As stated in Section 2.1.2, energy flexibility can change over time depending on multiple factors, like flexible demands or variable productions from renewable distributed energy sources affected by weather conditions. Independently of the cause, these values behave differently across geographical areas, and

consequently across the electrical grid. In network topologies that expand over large areas, energy can be more abundant in certain zones than in others. Such examples are MANs.

To compare how purposely designed energy flexibility-aware routing algorithms behave against more classic approaches, in this experiment, Atlanta's MAN topology [55] shown in Figure 17 is recreated. For this scenario, faced with an increase in traffic demand, and with distinct energy flexibility values for multiple devices, a classic Open Shortest Path First (OSPF) routing algorithm and a novel energy-ware one are evaluated. The test simulates an increase in energy availability in a given location, while simultaneously an energy availability decrease in another.

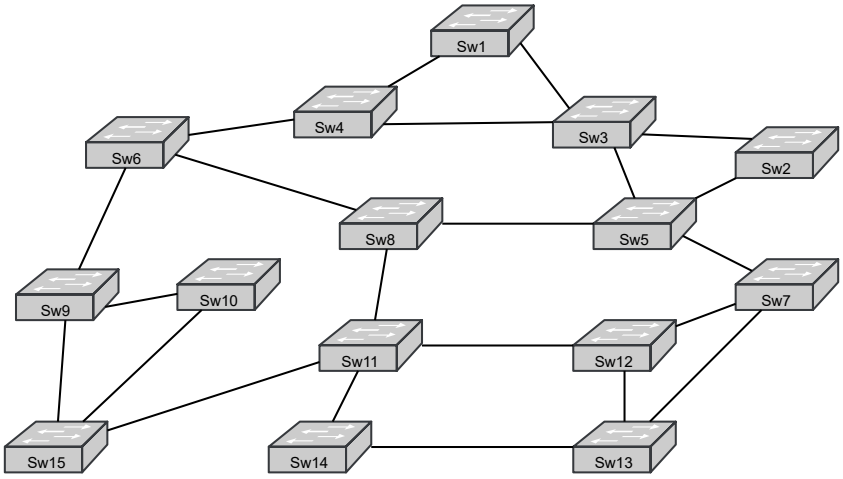


Figure 17: Atlanta's MAN toplogy

The results presented in Figures 18 and 19 show the average energy consumption in fifteen minutes intervals, across a simulated twelve hours period. The plotted recommended energy consumption corresponds to the energy consumption estimate, plus the energy flexibility, setting an ideal maximum value target for what a switch should consume.

As Figure 18 shows, during the mentioned period, the energy available to switch *Sw14* decreases over time. When using OSPF to apply the routing rules, the algorithm is not capable of reacting to such event, and eventually, the switch ends up operating above the recommended energy consumption. Meanwhile, the novel algorithm being developed minimizes this effect by anticipating its occurrence. Around time slot number twenty, as also seen in Figure 18, switch *Sw14*'s energy consumption drops to an inferior value, reflecting a decrease in its computational load. Such happens because the controller reacts to the decrease in availability and preemptively shifts some load away from the device, relocating network flows to alternative paths with more energy at their disposal. Contrarily, Figure 19 illustrates an increase in energy flexibility for switch *Sw5*. Again, OSPF can not adapt to such changes, and the switch remains idle during the entire period. With the novel routing logic, however, this increase in available energy is leveraged and flows are moved towards this device.

Listings 4.4 and 4.5 display FlowMonitor outputs containing IPPM metrics for a particular network flow,

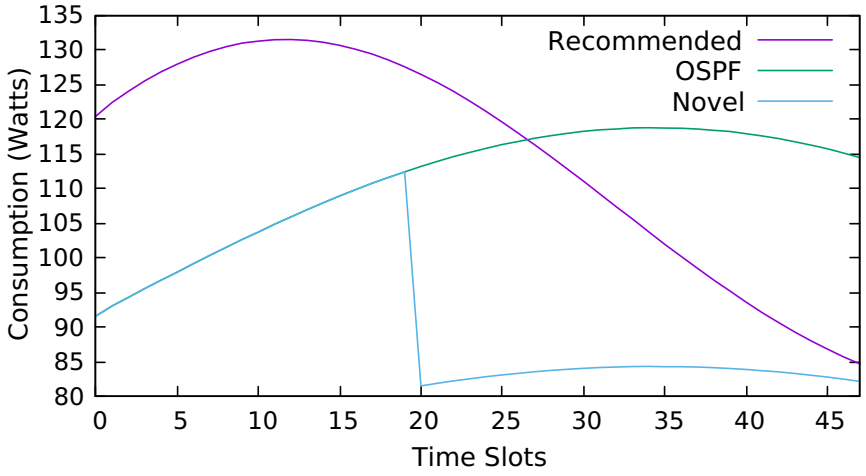


Figure 18: Switch Sw14 energy consumption

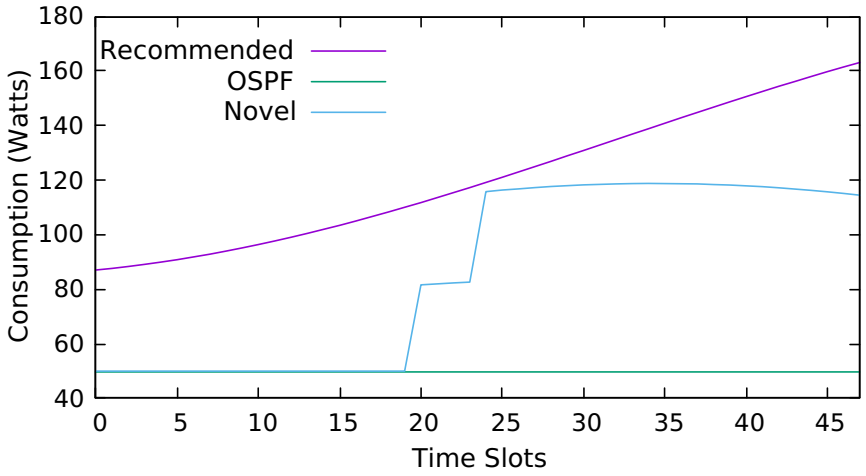


Figure 19: Switch Sw5 energy consumption

both when using OSPF and the novel algorithm respectively. The results show that when using the energy flexibility-aware routing algorithm, there is an increase in the mean and maximum delay experienced by this particular flow. They also show that there is a peak maximum jitter of 6 milliseconds, allowing to conclude that this flow suffers a relocation. This behavior is expected to happen, and depending on the nature of the communication, these values are still within acceptable values and do not affect the QoS.

```
FlowID: 13 (UDP 10.1.2.1/49153 -->
    10.1.5.1/10013)
  Bitrate:
    TX bitrate: 45254.21 kbit/s
    RX bitrate: 45254.21 kbit/s
  Delay:
    Mean Delay: 9.39 ms
    Max Delay: 9.00 ms
    Min Delay: 9.00 ms
  Jitter:
    Mean Jitter: 0.00 ms
    Max Jitter: 0.00 ms
    Min Jitter: 0.00 ms
  Packet Loss:
    Packet Loss Ratio: 0.00 %
```

Listing 4.4: Flow metrics with OSPF

```
FlowID: 13 (UDP 10.1.2.1/49153 -->
    10.1.5.1/10013)
  Bitrate:
    TX bitrate: 45254.21 kbit/s
    RX bitrate: 45254.21 kbit/s
  Delay:
    Mean Delay: 13.65 ms
    Max Delay: 16.00 ms
    Min Delay: 9.00 ms
  Jitter:
    Mean Jitter: 0.00 ms
    Max Jitter: 6.00 ms
    Min Jitter: 0.00 ms
  Packet Loss:
    Packet Loss Ratio: 0.00 %
```

Listing 4.5: Flow metrics with Novel algorithm

Although a simple example, this test demonstrates that energy flexibility can in fact be utilized to enhance energy consumption in networks by moving traffic across different alternative paths. Moreover, it shows that this can be achieved with minimal QoS penalties. This also still represents early work in the development of these techniques developed with the Flexcomm Simulator, with more developments expected to happen, leading to more advanced algorithms capable of prioritizing communications more susceptible to QoS (*e.g.*, Voice Over IP - VOIP).

4.6 Summary

This chapter started with an overall demonstration of the Flexcomm Simulator's ability to generate simulation data that allows comparing how different routing strategies perform in terms of energy consumption and QoS impact. It then proceeded to study how the simulation tool scales to deal with larger simulated networks, and how distributed simulations can help improve the execution duration. Lastly, it highlights early work on a novel routing algorithm, being developed resorting to Flexcomm Simulator, capable of prioritizing energy flexibility and help enhance the energy consumption of a given network. The final chapter that follows, presents the final conclusions on this work and the prospects for future work.

5 Conclusions

This chapter presents the conclusions based on the results obtained in Chapter 4 and on the overall development of the Flexcomm Simulator. It discusses the main contributions of this work and future work perspectives.

5.1 Conclusions

With the continuous development of the Internet, the energy consumption and environmental impact of wired networks have become a major concern. Pursuing more efficient traffic engineering techniques is now a higher priority for telecommunication operators and even governments. Developing such methods, however, requires ways to create and properly evaluate new policies without interfering with production environments, but that still produce accurate results.

The Flexcomm Simulator proposed by this work provides a platform with the purpose of helping to develop new strategies in the search for more sustainable large scale networks. This simulation tool, built upon NS-3 network simulator, combines and integrates some existent models with new ones to put together the programmability of SDN with today's Smart Grids energy flexibility in a single environment. This allows to evaluate how different routing strategies targeting these two concepts can help enhance energy consumption.

A proof of concept presented in Chapter 4 shows that detailed data can be generated to help compare how distinct routing strategies perform. Moreover, the Flexcomm Simulator is prepared to run distributed simulations that can assist in improving simulation duration, providing faster development rates. Finally, the tool has proved its abilities by helping to develop early work in energy flexibility-aware routing.

5.2 Contributions

Although the main contribution of this work is the Flexcomm Simulator as a whole, other contributions can be considered from the developments presented in Chapter 3. The changes and improvements proposed to NS-3's modules and core, represent by themselves relevant work, more importantly:

- The new proposed configuration format, described in Section 3.3.2, introduces a better method to describe simulations, portable across different NS-3 versions and other network simulators.
- *P2PEthernet* as a new link model, introduced in Section 3.4.1, enhancing NS-3 with a full duplex abstraction that better fits modern Internet.
- OFSwitch13 added support for MPI and detailed in Section 3.11, that unlocks the ability to simulate generic SDN topologies with distributed simulations, improving simulation duration.
- The advancements and improvements made with the ECOFEN module, enumerated in Section 3.5, that represent a continuation on the development of the framework.

5.3 Future Work

For future work, the continuous development of the Flexcomm Simulator is the main focus. Is expected for the information provided by Smart Grids to enrich in the future, providing more detailed data that can improve energy-aware routing algorithms. Moreover, SDN technology is continuing to evolve with improvements each year. Therefore, is required for the simulation environment to keep up with these advancements, ensuring an up-to-date platform to develop new routing strategies. Increasing support for more traffic generators can also help to evaluate with more detail the developed strategies and generate more accurate data on QoS impact. Moreover, the addition of more complementary simulation features (*e.g.*, selectively turning off devices [56], adaptive link data rates [57]) can help in the development of more advanced and innovative algorithms.

The results presented in Chapter 4 also show that the proposed simulation environment still has limitations when recreating large volumes of network traffic, a recurrent issue in network simulation. This creates the need to pursue more optimizations for the platform, and alternatives that help improve the speed of iteration in which new strategies can be tested.

Bibliography

[1] E. Ekudden, “Breaking the energy curve”, p. 12,

[2] G. Fettweis and E. Zimmermann, “ICT ENERGY CONSUMPTION – TRENDS AND CHALLENGES”, 2008.

[3] IEA, “Data centres and data transmission networks”, Paris, 2021. [Online]. Available: https://www.iea.org/reports/data-centres-and-data-transmission-networks.

[4] European Commission. Directorate General for the Information Society and Media., *Vision and challenges for realising the Internet of things.* LU: Publications Office, 2010. [Online]. Available: https://data.europa.eu/doi/10.2759/26127.

[5] Y. Mehmood, F. Ahmad, I. Yaqoob, A. Adnane, M. Imran, and S. Guizani, “Internet-of-things-based smart cities: Recent advances and challenges”, *IEEE Communications Magazine*, vol. 55, no. 9, pp. 16–24, 2017, issn: 0163-6804. doi: 10.1109/MCOM.2017.1600514. [Online]. Available: http://ieeexplore.ieee.org/document/8030479/.

[6] M. Agiwal, A. Roy, and N. Saxena, “Next generation 5g wireless networks: A comprehensive survey”, *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 1617–1655, 2016, issn: 1553-877X. doi: 10.1109/COMST.2016.2532458. [Online]. Available: http://ieeexplore.ieee.org/document/7414384/.

[7] ERICSSON, “6g - connecting a cyber-physical world”, Feb. 2022. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/white-papers/a-research-outlook-towards-6g.

[8] IEA-4E EDNA, “Intelligent efficiency for data centres and wide area networks”, 2019.

[9] 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”, *IEEE Communications Surveys & Tutorials*, vol. 13, no. 2, pp. 223–244, 2011, issn: 1553-877X. doi: 10.1109/SURV.2011.071410.00073. [Online]. Available: http://ieeexplore.ieee.org/document/5522467/.

[10] M. Pickavet, W. Vereecken, S. Demeyer, P. Audenaert, B. Vermeulen, C. Develder, D. Colle, B. Dhoedt, and P. Demeester, "Worldwide energy needs for ict: The rise of power-aware networking", in *2008 2nd International Symposium on Advanced Networks and Telecommunication Systems*, 2008, pp. 1–3. doi: 10.1109/ANTS.2008.4937762.

[11] A. B. Vieira, W. N. Paraizo, L. J. Chaves, L. H. A. Correia, and E. F. Silva, "An SDN-based energy-aware traffic management mechanism", *Annals of Telecommunications*, vol. 77, no. 3, pp. 139–150, Apr. 2022, issn: 0003-4347, 1958-9395. doi: 10.1007/s12243-021-00863-x. [Online]. Available: https://link.springer.com/10.1007/s12243-021-00863-x.

[12] L. Chiaraviglio, M. Mellia, and F. Neri, "Energy-aware backbone networks: A case study", in *2009 IEEE International Conference on Communications Workshops*, Dresden, Germany: IEEE, Jun. 2009, pp. 1–5, isbn: 978-1-4244-3437-4. doi: 10.1109/ICCW.2009.5208038. [Online]. Available: http://ieeexplore.ieee.org/document/5208038/.

[13] ONF, "Software-defined networking: The new norm for networks".

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, Mar. 31, 2008, issn: 0146-4833. doi: 10.1145/1355734.1355746. [Online]. Available: https://dl.acm.org/doi/10.1145/1355734.1355746 (visited on 09/12/2022).

[15] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors", *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, Jul. 28, 2014, issn: 0146-4833. doi: 10.1145/2656877.2656890. [Online]. Available: https://dl.acm.org/doi/10.1145/2656877.2656890 (visited on 09/12/2022).

[16] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? implementation challenges for software-defined networks", *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, Jul. 2013, issn: 0163-6804. doi: 10.1109/MCOM.2013.6553676. [Online]. Available: http://ieeexplore.ieee.org/document/6553676/.

[17] T. H. Vu, V. C. Luc, N. T. Quan, T. M. Nam, N. H. Thanh, and P. N. Nam, "The new method to save energy for openflow switch based on traffic engineering", in *2014 2nd International Conference on Electronic Design (ICED)*, Penang, Malaysia: IEEE, Aug. 2014, pp. 309–314, isbn: 978-1-4799-6103-0. doi: 10.1109/ICED.2014.7015820. [Online]. Available: http://ieeexplore.ieee.org/document/7015820/.

[18] Tran Hoang Vu, Pham Ngoc Nam, Tran Thanh, Le Thai Hung, Le Anh Van, Nguyen Duy Linh, To Duc Thien, and Nguyen Huu Thanh, "Power aware OpenFlow switch extension for energy saving in data centers", in *The 2012 International Conference on Advanced Technologies for Communications*, Ha Noi, Vietnam: IEEE, Oct. 2012, pp. 309–313, isbn: 978-1-4673-4352-7 978-1-4673-4351-0. doi: 10.1109/ATC.2012.6404282. [Online]. Available: http://ieeexplore.ieee.org/document/6404282/.

[19] T. H. Vu, V. C. Luc, N. T. Quan, N. H. Thanh, and P. N. Nam, "Energy saving for OpenFlow switch on the NetFPGA platform based on queue engineering", *SpringerPlus*, vol. 4, no. 1, p. 64, Dec. 2015, issn: 2193-1801. doi: 10.1186/s40064-014-0775-8. [Online]. Available: https://springerplus.springeropen.com/articles/10.1186/s40064-014-0775-8 (visited on 09/15/2022).

[20] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An Exhaustive Survey on P4 Programmable Data Plane Switches: Taxonomy, Applications, Challenges, and Future Trends", en, *IEEE Access*, vol. 9, pp. 87 094–87 155, 2021, issn: 2169-3536. doi: 10.1109/ACCESS.2021.3086704. [Online]. Available: https://ieeexplore.ieee.org/document/9447791/ (visited on 08/12/2022).

[21] USEF, "Flexibility Value Chain", 2018.

[22] E. Lochin, T. Perennou, and L. Dairaine, "When should i use network emulation?", *annals of telecommunications - annales des télécommunications*, vol. 67, no. 5, pp. 247–255, Jun. 2012, issn: 0003-4347, 1958-9395. doi: 10.1007/s12243-011-0268-5. arXiv: 1002.2827[cs]. [Online]. Available: http://arxiv.org/abs/1002.2827.

[23] M. Besta, M. Schneider, S. Di Girolamo, A. Singla, and T. Hoefler, *Towards million-server network simulations on just a laptop*, May 26, 2021. arXiv: 2105.12663[cs]. [Online]. Available: http://arxiv.org/abs/2105.12663.

[24] T.-S. Chou, S. Baker, and M. Vega-Herrera, "A comparison of network simulation and emulation virtualization tools", in *2016 ASEE Annual Conference & Exposition Proceedings*, New Orleans, Louisiana: ASEE Conferences, Jun. 2016, p. 26 285. doi: 10.18260/p.26285. [Online]. Available: http://peer.asee.org/26285.

[25] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: A generic framework for large-scale distributed experiments", in *Tenth International Conference on Computer Modeling and Simulation (uksim 2008)*, Cambridge, UK: IEEE, 2008, pp. 126–131, isbn: 978-0-7695-3114-4. doi: 10.1109/UKSIM.2008.28. [Online]. Available: http://ieeexplore.ieee.org/document/4488918/.

[26] K. Fujiwara and H. Casanova, "Speed and accuracy of network simulation in the SimGrid framework", in *Proceedings of the 2nd International ICST Conference on Performance Evaluation Methodologies and Tools*, Nantes, France: ICST, 2007, isbn: 978-963-9799-00-4. doi: 10.4108/nstools.2007.2010. [Online]. Available: http://eudl.eu/doi/10.4108/nstools.2007.2010.

[27] L. Guegan, B. L. Amersho, A.-C. Orgerie, and M. Quinson, "A large-scale wired network energy model for flow-level simulations", in *Advanced Information Networking and Applications*, L. Barolli, M. Takizawa, F. Xhafa, and T. Enokido, Eds., vol. 926, Series Title: Advances in Intelligent Systems and Computing, Cham: Springer International Publishing, 2020, pp. 1047–1058, isbn: 978-3-030-15032-7. doi: 10.1007/978-3-030-15032-7_88. [Online]. Available: http://link.springer.com/10.1007/978-3-030-15032-7_88.

[28] J. Pelkey and G. Riley, "Distributed simulation with MPI in ns-3", in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, Barcelona, Spain: ACM, 2011, isbn: 978-1-936968-00-8. doi: 10.4108/icst.simutools.2011.245585. [Online]. Available: http://eudl.eu/doi/10.4108/icst.simutools.2011.245585.

[29] S. Nikolaev, E. Banks, P. D. Barnes, D. R. Jefferson, and S. Smith, "Pushing the envelope in distributed ns-3 simulations: One billion nodes", in *Proceedings of the 2015 Workshop on ns-3*, Barcelona Spain: ACM, May 13, 2015, pp. 67–74, isbn: 978-1-4503-3375-7. doi: 10.1145/2756509.2756525. [Online]. Available: https://dl.acm.org/doi/10.1145/2756509.2756525.

[30] L. J. Chaves, I. C. Garcia, and E. R. M. Madeira, "OFSwitch13: Enhancing ns-3 with OpenFlow 1.3 support", in *Proceedings of the Workshop on ns-3 - WNS3 '16*, Seattle, WA, USA: ACM Press, 2016, pp. 33–40, isbn: 978-1-4503-4216-2. doi: 10.1145/2915371.2915381. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2915371.2915381.

[31] A.-C. Orgerie, L. Lefevre, I. Guerin-Lassous, and D. M. Lopez Pacheco, "ECOFEN: An end-to-end energy cost mOdel and simulator for evaluating power consumption in large-scale networks", in *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, Lucca, Italy: IEEE, Jun. 2011, pp. 1–6, isbn: 978-1-4577-0352-2. doi: 10.1109/WoWMoM.2011.5986203. [Online]. Available: http://ieeexplore.ieee.org/document/5986203/.

[32] B. F. Cornea, A.-C. Orgerie, and L. Lefevre, "Studying the energy consumption of data transfers in clouds: The ecofen approach", in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, Luxembourg, Luxembourg: IEEE, Oct. 2014, pp. 143–148, isbn: 978-1-4799-2730-2. doi: 10.1109/CloudNet.2014.6968983. [Online]. Available: http://ieeexplore.ieee.org/document/6968983/.

[33] A.-C. Orgerie, B. L. Amersho, T. Haudebourg, M. Quinson, M. Rifai, D. L. Pacheco, and L. Lefevre, "Simulation toolbox for studying energy consumption in wired networks", in *2017 13th International Conference on Network and Service Management (CNSM)*, Tokyo: IEEE, Nov. 2017, pp. 1–5, isbn:

978-3-901882-98-2. doi: 10.23919/CNSM.2017.8256037. [Online]. Available: http://ieeexplore.ieee.org/document/8256037/.

[34] A. Varga, "THE OMNET++ DISCRETE EVENT SIMULATION SYSTEM", p. 8,

[35] L. Mészáros, A. Varga, and M. Kirsche, "Inet framework", in. May 2019, pp. 55–106, isbn: 978-3-030-12841-8. doi: 10.1007/978-3-030-12842-5_2.

[36] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms", *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Oct. 2014, issn: 07437315. doi: 10.1016/j.jpdc.2014.06.008. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0743731514001105.

[37] R. Jansen, J. Newsome, and R. Wails, "Co-opting Linux Processes for High-Performance Network Simulation", en,

[38] R. Jansen and N. Hooper, "Shadow: Running Tor in a Box for Accurate and Efficient Experimentation:" en, Defense Technical Information Center, Fort Belvoir, VA, Tech. Rep., Sep. 2011. doi: 10.21236/ADA559181. [Online]. Available: http://www.dtic.mil/docs/citations/ADA559181 (visited on 12/28/2022).

[39] A. Fernandez-Fernandez, C. Cervello-Pastor, and L. Ochoa-Aday, "Achieving energy efficiency: An energy-aware approach in sdn", in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–7. doi: 10.1109/GLOCOM.2016.7841561.

[40] B. G. Assefa and Ö. Özkasap, "A survey of energy efficiency in sdn: Software-based methods and optimization models", *Journal of Network and Computer Applications*, vol. 137, pp. 127–143, 2019, issn: 1084-8045. doi: https://doi.org/10.1016/j.jnca.2019.04.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804519301134.

[41] J. Winick and S. Jamin, "Inet-3.0: Internet topology generator", p. 19,

[42] P. Mahadevan, C. Hubble, and D. Krioukov, "Orbis: Rescaling degree correlations to generate annotated internet topologies", p. 12,

[43] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with rocketfuel", *IEEE/ACM Transactions on Networking*, vol. 12, no. 1, pp. 2–16, Feb. 2004, issn: 1063-6692. doi: 10.1109/TNET.2003.822655. [Online]. Available: http://ieeexplore.ieee.org/document/1268075/.

[44] *Standard for the transmission of IP datagrams over IEEE 802 networks*, RFC 1042, Feb. 1988. doi: 10.17487/RFC1042. [Online]. Available: https://www.rfc-editor.org/info/rfc1042.

[45] W. A. Simpson, *The Point-to-Point Protocol (PPP)*, RFC 1661, Jul. 1994. doi: 10.17487/RFC1661. [Online]. Available: https://www.rfc-editor.org/info/rfc1661.

[46] M. Fedor, M. L. Schoffstall, J. R. Davin, and D. J. D. Case, *Simple Network Management Protocol (SNMP)*, RFC 1157, May 1990. doi: 10.17487/RFC1157. [Online]. Available: https://www.rfc-editor.org/info/rfc1157.

[47] S. Troia, Gao Sheng, R. Alvizu, G. A. Maier, and A. Pattavina, "Identification of tidal-traffic patterns in metro-area mobile networks via matrix factorization based model", in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Kona, HI: IEEE, Mar. 2017, pp. 297–301, isbn: 978-1-5090-4338-5. doi: 10.1109/PERCOMW.2017.7917576. [Online]. Available: https://ieeexplore.ieee.org/document/7917576/.

[48] D. Ammar, T. Begin, and I. Guerin-Lassous, "A new tool for generating realistic internet traffic in NS-3", in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, Barcelona, Spain: ACM, 2011, isbn: 978-1-936968-00-8. doi: 10.4108/icst.simutools.2011.245548. [Online]. Available: http://eudl.eu/doi/10.4108/icst.simutools.2011.245548.

[49] M. Zukerman, T. Neame, and R. Addie, "Internet traffic modeling and future technology implications", in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, vol. 1, San Francisco, CA, USA: IEEE, 2003, pp. 587–596, isbn: 978-0-7803-7752-3. doi: 10.1109/INFCOM.2003.1208709. [Online]. Available: http://ieeexplore.ieee.org/document/1208709/.

[50] R. Pries, Z. Magyari, and P. Tran-Gia, "An HTTP web traffic model based on the top one million visited web pages", en, in *Proceedings of the 8th Euro-NF Conference on Next Generation Internet NGI 2012*, Karlskrona, Sweden: IEEE, Jun. 2012, pp. 133–139, isbn: 978-1-4673-1634-7 978-1-4673-1632-3 978-1-4673-1633-0. doi: 10.1109/NGI.2012.6252145. [Online]. Available: http://ieeexplore.ieee.org/document/6252145/ (visited on 01/03/2023).

[51] J. Morley, K. Widdicks, and M. Hazas, "Digitalisation, energy and data demand: The impact of internet traffic on overall and peak electricity consumption", *Energy Research & Social Science*, vol. 38, pp. 128–137, Apr. 2018, issn: 22146296. doi: 10.1016/j.erss.2018.01.018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2214629618301051.

[52] J. L. Garcia-Dorado, A. Finamore, M. Mellia, M. Meo, and M. Munafo, "Characterization of ISP traffic: Trends, user habits, and access technology impact", *IEEE Transactions on Network and Service Management*, vol. 9, no. 2, pp. 142–155, Jun. 2012, issn: 1932-4537. doi: 10.1109/TNSM.2012.022412.110184. [Online]. Available: http://ieeexplore.ieee.org/document/6158423/.

[53] G. Carneiro, P. Fortuna, and M. Ricardo, "FlowMonitor - a network monitoring framework for the network simulator 3 (NS-3)", in *Proceedings of the 4th International ICST Conference on Performance Evaluation Methodologies and Tools*, Pisa, Italy: ICST, 2009, isbn: 978-963-9799-70-7. doi: 10.4108/ICST.VALUETOOLS2009.7493. [Online]. Available: http://eudl.eu/doi/10.4108/ICST.VALUETOOLS2009.7493.

[54] R. Fujimoto, "PARALLEL AND DISTRIBUTED SIMULATION",

[55] M. K. Awad, Y. Rafique, and R. A. M'Hallah, "Energy-aware routing for software-defined networks with discrete link rates: A benders decomposition-based heuristic approach", *Sustainable Computing: Informatics and Systems*, vol. 13, pp. 31–41, 2017, issn: 2210-5379. doi: https://doi.org/10.1016/j.suscom.2016.11.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210537916301251.

[56] L. Chiaraviglio, M. Mellia, and F. Neri, "Reducing power consumption in backbone networks", in *2009 IEEE International Conference on Communications*, Dresden, Germany: IEEE, Jun. 2009, pp. 1–6. doi: 10.1109/ICC.2009.5199404. [Online]. Available: http://ieeexplore.ieee.org/document/5199404/.

[57] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, "Reducing network energy consumption via sleeping and rate-adaptation", in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08, San Francisco, California: USENIX Association, 2008, pp. 323–336, isbn: 1119995555221.

I

Configuration Examples

This annex presents examples for Flexcomm Simulator's configuration files. This specific configuration corresponds to the test scenario described in Section 4.3. The network topology is made of two hosts, and two switches connected in chain.

```
#### HOSTS ####
[H1]
type = "host"
rank = 0

[H2]
type = "host"
rank = 1

#### SWITCHES ####
[Sw1]
type = "switch"
rank =  0
cpuCapacity = "10Gbps"

[Sw1.chassis]
template = "demo-template"

[Sw2]
type = "switch"
rank = 1
cpuCapacity = "10Gbps"

[Sw2.chassis]
template = "demo-template"
```

Listing I.1: nodes.toml

```
[link1]
edges = ["H1", "Sw1"]
dataRate = "1Gbps"
delay = "2ms"

[link2]
edges = ["H2", "Sw2"]
dataRate = "1Gbps"
delay = "2ms"

[link3]
edges = ["Sw1", "Sw2"]
dataRate = "1Gbps"
delay = "2ms"
```

Listing I.2: links.toml

```
[simulator]
stopTime = "5min"

[flowMonitor]
enable = true

[ecofen]
traceFile = true
interval = "1s"

[switchStats]
enable = true
interval = "1s"

[mibLogger]
enable = false
interval = "5s"

[linkStats]
enable = true
interval = "1s"
```

Listing I.3: configs.toml

```
[app1]
startTime = "5s"
node = "H1"
remote = "H2"
type = "CONSTGen"
protocol = "TCP"
dataRate = "100Mb/s"
```

Listing I.4: applications.toml

```
################################
########## Chassis #############
################################
[chassis]

[chassis.demo-template]
model = "loadBased"
percentages = [0.0, 1.0]
consumptions = [50.0, 150.0]

################################
########## Interfaces ##########
################################
[interface]

[interface.demo-template]
model = "basic"
onConso = 1.0
offConso = 0.5
```

Listing I.5: energy-templates.toml