UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

NATÁLIA GUBIANI RAMPON

A Viability Analysis of Low-end Devices for Low-cost Programmable LPWAN

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Engineering

Advisor: Prof. Dr. Weverton Cordeiro
Coadvisor: André Scheibe

Porto Alegre
May 2022

“I find the great thing in this world is not so much where we stand, as in what direction we are moving”

— OLIVER WENDELL HOLMES

ACKNOWLEDGMENTS

First and foremost, I would like to thank my parents, Beatriz Gubiani and Vitalino Rampon, whose support was the ladder I used to climb over any walls I faced during my graduation time. My regards also go to my twin brother Mateus who has been with me, quite literally, throughout my whole life and who did not doubt for even a moment I would prevail. Although I wavered, my family's support did not. For that, I am forever grateful.

I could not forget to thank the friends I made during my time at UFRGS, who heard me cry and complain that I would never graduate. My anxieties were unfounded, but mostly because you guided me through.

Finally, I give my heartfelt thanks to my co-advisor, André Scheibe, for the guidance he provided me with throughout the months I worked on this. Your kind words gave me hope when all seemed lost.

ABSTRACT

Internet access is crucial to our daily life in this day and age. However, it is not yet ubiquitous outside urban areas, such as those inhabited by native and rural communities. In this work, the feasibility of using Software-Defined Networking to manage a programmable Low-Power Wide-Area Network (LPWAN) composed of low-end LoRa radio devices to provide power affordable internet access is evaluated. A communication wrapper for a Ronoth LoStik antenna has been developed to this purpose, in order to provide an interface between the programmable forwarding devices and the incoming packets. We assess the real-life gains obtained by using a newly proposed theoretical protocol, Lightweight Tunnel Protocol (LTP), in a distributed SDN controller architecture using this setup. We conclude that the low-end devices are capable of providing a communication channel for forwarding devices in an LPWAN. However, the throughput is limited to 1 Kbps for this particular setup, which is a low speed for today's internet speed standards. Nevertheless, the LTP protocol has proved to provide up to 62.7% gain with 88-byte payloads and 34.1% with 128-byte ones.

Keywords: Low-Power Wide-Area Networks (LPWAN). Programmable forwarding planes. Software-defined networks. Community networks. LoRa. P4.

Uma Análise de Viabilidade de Dispositivos de Baixa Gama para LPWANs Programáveis de Baixo Custo

RESUMO

Atualmente, o acesso à internet é crucial para nossa vida diária. No entanto, ele ainda não é onipresente fora de áreas urbanas, tais como aquelas habitadas por comunidades indígenas e rurais. Neste trabalho, a viabilidade do uso de redes definidas por software para gerenciar redes de baixa potência e larga área (LPWAN) programáveis compostas por dispositivos de rádio LoRa de baixa gama é avaliada como alternativa para providenciar um acesso à internet de baixo consumo energético. Uma interface de comunicação para antenas Ronoth LoStik foi desenvolvida para este propósito, de forma a providenciar um canal de comunicação entre os dispositivos de encaminhamento programáveis e os pacotes recebidos. Avaliamos os ganhos concretos obtidos ao usar um recém proposto protocolo teórico de comunicação, Lightweight Tunnel Protocol (LTP), em uma arquitetura de rede definida por software com controladores distribuídos. Concluímos que os dispositivos de baixa gama são capazes de prover um canal de comunicação para os dispositivos de encaminhamento em uma LPWAN. No entanto, a taxa de transferência é limitada a 1 Kbps com o dispositivo utilizado, que é uma velocidade baixa para os padrões atuais de velocidade de internet. Assim mesmo, o protocolo LTP provou-se capaz de prover ganhos de até 62,7% com carga útil de 88 bytes e de 34,1% com cargas de 128 bytes.

Palavras-chave: LPWAN, planos de dados programáveis, redes definidas por software, redes comunitárias, LoRA, P4.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASIC	Application-specific Integrated Circuit
bps	bits per second
CLI	Command-line Interface
CR	Cognitive Radio
CSS	Chirp Spread Spectrum
FPGA	Field-programmable Gate Array
FSK	Frequency-shift Keying
IoT	Internet of Things
IP	Internet Protocol
ISM	Industrial, Scientific and Medical
Kbps	Kilobits per second
LPWAN	Low-power Wide-area Network
LTP	Lightweight Tunnel Protocol
M2M	Machine to Machine
MAC	Media Access Control
MTU	Maximum Transmission Unit
OS	Operating System
P4	Programming Protocol-independent Packet Processors
PISA	Protocol Independent Switch Architecture
PSA	Portable Switch Architecture
QoS	Quality of Service
REST	Representational State Transfer

SDN	Software-defined Networks
SF	Spreading Factor
TCP	Transmission Control Protocol
TVWS	TV White Spaces
UDP	User Datagram Protocol
USB	Universal Serial Bus
UHF	Ultra High Frequency
VHF	Very High Frequency

CONTENTS

1 INTRODUCTION

Universal internet access has been declared a key medium by which individuals may exercise their human right to freedom of speech and right to education (UNITED NATIONS GENERAL ASSEMBLY, 2016); however, many citizens of remote communities cannot yet enjoy internet access. Hurdles such as high maintenance cost, operation complexity, and prohibitively large infrastructure prevent the implantation of access networks by Internet Service Providers (ISP) in the areas where they live.

While cellular networks and satellite links may be available in some areas, those are often too expensive for the members of such communities. Many initiatives have been led by private enterprises and foundations, such as Google's Project Loon (X DEVELOPMENT LLC, 2020) and its successor Project Tara (X DEVELOPMENT LLC, 2021), IRTF's GAIA (IRTF GAIA, 2020) and the A4AI coalition (A4AI, 2020), to attempt to mitigate these obstacles and bring affordable internet connectivity to all.

To this end, low-end devices networking may help overpass the digital divide by providing a low-bandwidth, but also low-cost alternative to traditional internet connections for sparsely spread individuals in removed locations. These kinds of networks are in vogue today, due to the increasing popularity of IoT devices (RAZA; KULKARNI; SOORIYABANDARA, 2017), but the challenge of how to provide connectivity to this kind of device remains open. As a contender for solving this issue, Low-Power Wide-Area Networks (LPWAN) are a new generation of wireless technology, capable of achieving many of the target characteristics desirable for low-end networking, such as long-range transmission, low-power consumption, and scalability (IETF, 2020).

Alongside the rise of the LPWAN, we have seen in the past decade the emergence of new ways to define computer networks, enabling the conception of open standards and forwarding elements whose behavior is defined using languages such as P4 (BOSSHART et al., 2014). Through the use of Software Defined Networks (SDN) and Programmable Data Planes (PDP), low-end devices could be used to form a virtually programmable and open LPWAN, decoupling LPWAN integration with traditional networks from vendor-specific interfaces and standards.

Following the conceptual architecture for such a solution proposed previously (SCHEIBE et al., 2021), the aim of this research is to test the feasibility of implementing this conceptual work with devices equipped with Ronoth LoStik components with LoRa RN2483 antenna modules (RONOTH, 2021) as the LPWAN transceiver. A posi-

tive response to this evaluation could provide new grounds of research and possibilities of practical deployments for low-end, low-cost, and low-data-rate networks, taking us a little further into bridging the digital divide.

The contributions achieved through this work include the development of a communication interface between a LoRa antenna and a software-defined network of forwarding devices; the assessment of the real-life gains obtained by using a theoretical protocol based on data plane programmability proposed in a previous work (SCHEIBE et al., 2021); the testing and evaluation of the usage of a Ronoth LoStik antenna, with an analysis of its limitations for this use case; as a result, the verification of the viability of using such a deployment to bring internet access to remote communities; and finally an open-source implementation of this interface and test cases to evaluate its performance[1].

This study is organized as follows: Chapter 2 considers the technical background for a programmable LPWAN with LoRa radio devices, while Chapter 3 discusses the related academic work. Chapter 4 presents the interface developed so to use the devices as switches in an LPWAN, and also describes our experimental environment and test case scenarios. Chapter 5 presents and discusses the results of said experiments. Finally, Chapter 6 concludes this work and expounds on possibilities of future research on this topic.

[1] https://github.com/nataliarampon/LoRaStickComm

2 TECHNICAL BACKGROUND

This chapter presents the main technical concepts related to this work. Firstly, the definition of an LPWAN and its leading technologies are presented, followed by a deeper insight into the one chosen for this project, LoRa. Then, in section 2.3, we present one of the main ideas permeating our effort, that of programmable networks. A brief history of them is presented, followed by its evolution in some of the fields most pertinent to a programmable LPWAN: software-defined networking, data plane programmability, and network virtualization.

2.1 Low-Power Wide-Area Network (LPWAN)

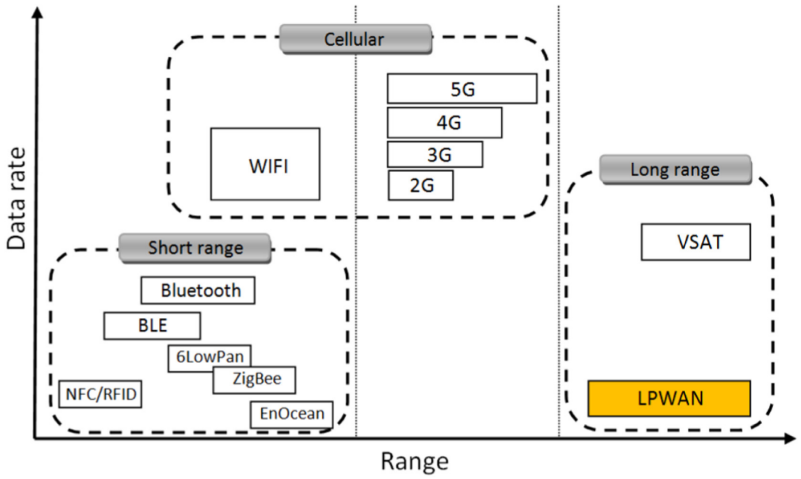
The Low-Power Wide-Area Network (LPWAN) is a type of wireless communication operating over a large geographic area. This kind of network contains desirable characteristics for inter-device communication in a network consisting of Machine to Machine (M2M) connections in areas where the maintenance of such devices would prove to be difficult, such as low-power consumption and long-range capacity of radio communication (up to 40km in rural zones and 5km in urban ones (CENTENARO et al., 2016)). However, these qualities are achieved through a compromise of having low data rate transmission and inconsistent quality of service (MEKKI et al., 2019).

Other widely used radio technologies such as Bluetooth could be used for the M2M communication, but since it is short-ranged, this would require a considerable amount of devices to be used for the establishment of the network. A more significant number of devices is undesirable because it makes maintenance more frequent and the overall operation more costly. A comparison between radio technology in regards to data rate and range of radio reach can be observed in Figure 2.1, with LPWAN made prominent in yellow.

Many different LPWAN technologies have been developed in order to provide the growing number of applications with a varied set of requirements with the best possible solutions. Such technologies differ in modulation technique, data rate, frequency used in communication (in licensed or unlicensed bands), range, maximum messages per data, and maximum payload length. The three leading technologies today are Sigfox, LoRa, and NB-IoT (MEKKI et al., 2019).

NB-IoT uses licensed frequency bands, which while providing great QoS, also

Figure 2.1: Comparison of radio communication technologies regarding data rate and range of radio reach



Source: (MEKKI et al., 2019)

incur a large financial overhead, with licensed LTE spectrum auctions costing hundreds of millions of dollars per MHz. This solution offers a prohibitively high cost, making it unfit for our purpose of a low-end and low-cost network.

The two other LPWAN technologies current in vogue are Sigfox and LoRa, both developed in France and employing unlicensed frequency spectra. Sigfox has a long range of communication (up to 40km in rural areas) and low noise levels. However, its maximum data rate is 100bps and it has a limitation on the number of messages to be sent in a day (each of those with a maximum payload length of only 12 bytes). Thus, it is a technology much better fit for devices that need only to communicate a few times per day (MEKKI et al., 2019).

The LoRa technology was thus chosen to be utilized in this project. For this reason, the next section will further elaborate on its characteristics.

2.2 LoRa

LoRa is a proprietary Chirp Spread Spectrum (CSS) modulation technique, which defines a physical layer of communication for signals in the sub-GHz ISM band. It operates under unlicensed bands (868 MHz in Europe, 915 MHz in North America, and 433 MHz in Asia), which do not require the acquisition of a license through auctions, but have no regulatory protection and are subject to a number of interfering signals. The CSS

modulation provides a signal with low noise levels, difficult to jam and having high interference resilience (REYNDERS; MEERT; POLLIN, 2016), thus making LoRa suitable for use in these unlicensed spectra.

The maximum payload length of a LoRa message is 255 bytes and its data rate falls between 300 bps and 27 Kbps, depending on the configuration of the CSS modulation. The technique uses six spreading factors, SF $\ni$ { 7, 8, 9, 10, 11, 12 }, with an increase in spreading factor having a lower data rate, but providing a stronger signal able to travel longer distances and less prone to errors, and vice-versa. It also affects battery life, because higher SFs need to stay active for longer and thus consume more energy. Along with the spreading factor, LoRa radio modules can also be configured to operate at different frequencies, going up to 2.4 GHz for higher data rates. Although outside of the sub GHz spectra, this frequency is also an ISM band used for device communication.

The LoRa Alliance has standardized a MAC layer protocol built on top of the LoRa modulation technique (LORA ALLIANCE, 2015) in order to provide better QoS for industry applications using it. The standard was first published in 2015, with new versions coming roughly every year since then, the latest being the one from October 2021. Acting as both a communication protocol and a system architecture, LoRaWAN provides a definition of how end-nodes communicate with gateways connected to a network server coordinating all exchanges. While this infrastructure provides security and reliability to communications, it doesn't rely on a mesh network architecture of interconnected devices, but rather on a star topology with devices connected to a central gateway node. This architecture made it impossible for our purposes to use LoRaWAN directly, on top of the fact the communication with the central node incurs an overhead and usage of an already limited bandwidth.

2.3 Programmable Networks

Computer networks are complex both in terms of management and heterogeneity. They are composed of a variety of devices, from the ubiquitous routers and switches, to middlebox equipment, such as firewalls and NAT translators. As networks and applications increased in complexity, the need to be able to manage and test a network node or groups of nodes as quickly and easily as possible arose. Programmable networks are such that make the devices in the traditional networks into active agents, being capable of executing code and actively participating in the processing functions of the data and control

planes.

This concept brings a multitude of advantages in comparison to traditional network management and development. Firstly, it decreases the cost to prototype and test network configurations and behaviors, as solutions can be deployed in any device with no attachments to vendor-specific mechanisms. It also reduces the frequency of errors, as all routers and switches are centrally managed through code configuration, making human intervention on single devices unnecessary and the network as a whole less error-prone. Lastly, programmable networks bring innovation within the network in fields such as traffic engineering and packet processing, which would not be possible with merely passive appliances (KREUTZ et al., 2014).

The idea of programmable networks was born from decades of research. The general idea can be traced back to the active networks developed in the 1990s (FEAMSTER; REXFORD; ZEGURA, 2014). These were the first efforts to extend the capabilities of devices such as routers and switches into active agents capable of executing custom functionality independently of the network node hardware (ALEXANDER et al., 1998; TENNENHOUSE; WETHERALL, 2007). The approach consisted of providing an API through which network resources could be manipulated at the network-node level.

The most widely spread manner of doing so was the *capsule model*, in which special data packets carried executable code to be used by the node. This approach was envisioned so to reduce the cost of computing by moving processing power to the network. It also gave traction to the idea of using programmable functions in the network, as a predecessor to virtual network functions. Other approaches included using packets for node configuration and network management (SCHWARTZ et al., 2000). However, the adoption of such technology was not widespread outside of the scientific community for a lack of clear implementation direction in real-world scenarios.

With the advent of active networks, which had much too broad a scope by focusing on the network nodes as a whole, the following research efforts focused on restricting the scope by clearly separating the data and control planes attributions. This was a novel approach, as traditional routers and switches present a tight coupling between these planes. Any customization tasks had to be implemented by making use of existing protocols or by proposing new standards to the industry. From these efforts, solutions that provided programmable control planes and visibility of the network as a whole emerged.

These were technologies that solved routing and traffic issues being faced by the engineers of the time, and thus were much better accepted by the community. The control

of the network packet forwarding was centralized by solutions such as SoftRouter (LAKSHMAN et al., 2004), which could in turn program the control plane through standardized interfaces with the data plane, such as ForCES (YANG et al., 2004) and Linux Netlink (KLEEN et al., 2003). By using these interfaces to replace forwarding-table entries in the switches and routers, these new approaches could effectively remove any control plane responsibility from the network devices and code them elsewhere on a much more powerful computer server with a holistic view of the network-wide traffic. However, since vendors had little incentive to adopt such technologies that would ease the life of new competitors, this technology was not widely adopted by the industry either.

These research efforts culminated in the network paradigm of Software-Defined Networks (SDN), which are characterized by two main principles:

- The separation of the control and data planes
- A logically centralized controller platform implementing the control logic

In the following section, the SDN overall architecture will be presented along with its use-cases and benefits.

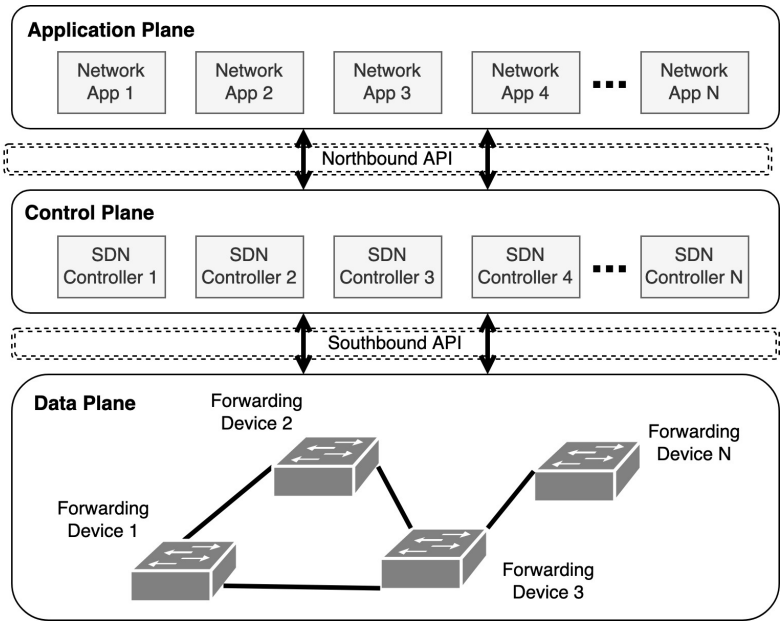
2.3.1 Software-Defined Networks

Relying on the road paved by the works on programmable networks, the network paradigm of the Software-Defined Networks evolved to embrace their core characteristics, while not straying far from the needs of the industry. The term Software-Defined Networks was coined in reference to Stanford University's OpenFlow API (MCKEOWN et al., 2008), which relied on existing switch hardware to program the data plane. This made the technology deployable from the start, which made research efforts gain traction.

The network architecture first envisioned for SDN decouples the forwarding devices' behavior (data plane) from the logic function by introducing a remote centralized controller (control plane). This external entity is responsible for determining the behavior of the network, since it has a view of the network as a whole and thus a better ability to create forwarding rules. This entity is called simply SDN controller, or Network Operating System (NOS). The NOS represents an abstraction of the network view, through which the programming of network behavior is much facilitated.

While other separation of concerns may exist where the management layer of the SDN has the ability to interact with both the control and the data plane (ALBOWARAB;

Figure 2.2: SDN Architecture



Source: the author (2022)

ZAKARIA; ABIDIN, 2018), we prefer to introduce the SDN architecture by focusing on the original three-plane model, seen in Figure 2.2, for the simplicity of reasoning. The definition and responsibilities of the elements seen on the diagram are outlined below, from top to bottom:

- **Application Plane:** Responsible for defining the policies and operation logic of the network that is going to be enforced by the control plane. It communicates such policies to the control plane by sending instructions to the northbound API. It relies on abstract information regarding the forwarding devices and network topology to programmatically make decisions regarding these policies (KREUTZ et al., 2014).
- **Network Applications:** These are the applications that will communicate their network requirements to the control plane. Examples of such applications are routing algorithms, load balancers, security monitoring, and firewalls. It is worth noting this includes not only responsibilities of routers and switches on traditional networks, but also functions executed by middleboxes, which provides a unified view of the appliances on the network (KREUTZ et al., 2014).
- **Northbound API:** Responsible for providing a common interface for the network applications. The kind and language used in the API depend on the NOS being used, some examples include REST APIs and programming languages such as Java and Python. It generally offers a higher level of abstraction than the southbound

API (KREUTZ et al., 2014).

- **Control Plane:** Responsible for establishing data handling policies along with populating the forwarding flow tables by communicating with the data plane. It collects information through the southbound API to have a holistic view of the network. Originally centralized, this plane may be organized into hierarchical or totally distributed approaches to improve network scalability (BENZEKKI; FERGOUGUI; ELALAOUI, 2016).
- **SDN Controller:** The controllers are responsible for dictating the behavior of the network and translating the high-level requirements received through the northbound API into southbound API instructions for the network devices. There are many controller platforms available for use today, like OpenDaylight (The Linux Foundation, 2021) and Hyperflux (TOOTOONCHIAN; GANJALI, 2010).
- **Southbound API:** Responsible for defining how the control and data planes interact. It provides the control plane with the ability to control the forwarding operations, and the data plane with the ability to inform its statistics and events (KREUTZ et al., 2014). Examples of southbound APIs are OpenFlow (MCKEOWN et al., 2008) and ForCES (YANG et al., 2004).
- **Data Plane:** Responsible for providing the network *de facto* infrastructure and forwarding capabilities, as well as data processing in some cases. This plane is where the network functionality is actually executed, thus defining its topology and connection between elements (KREUTZ et al., 2014).
- **Forwarding Devices:** A set of hardware or virtual software devices that use the network rules installed on them by the SDN controllers to perform operations on network packets (e.g.: rewrite a packet header, drop the packet, forward to one of its ports or to the controller). These devices may implement features usually performed on traditional networks by routers and switches, as well as those executed by middleboxes (KREUTZ et al., 2014).

Software-defined networking brings many advantages in comparison with traditional networks. It separates the concerns regarding networking into well-bounded layers, thus making issues on the definition of policies (application plane), the implementation of these policies in switching devices (control plane), and the actual forwarding of the traffic (data plane) easier to trace. This also results in a more flexible environment, with abstractions that make the definition of such behaviors easier for the engineers, who have to

deal with high-level languages on the southbound API instead of firmware and hardware updates. In fact, the forwarding abstraction provided by the data plane allows engineers to customize network behavior without any knowledge of the underlying hardware, providing a generalized view of network devices and functions.

They also make erroneous or malicious behavior easier to detect and act upon. Networks can remain intact by automatically reacting to such behaviors, which would maintain a higher QoS. Much different from issues with faulty switches on traditional networks, which are hard to detect and may isolate a whole section of the network (BUTLER et al., 2010).

Due to its qualities, software-defined networking has seen an increase in popularity in comparison with the earlier attempts at programmable networks. Some big industry players were early adopters of the paradigm, such as Google's B4, an SDN WAN used to connect data centers across the globe by bypassing vendor-specific solutions and significantly reducing costs (HONG et al., 2018), which has been active for 8 years.

2.3.2 Data Plane Programmability

With popular technologies such as cloud computing and machine learning needing massive amounts of data to function, it is no surprise that network operators were faced with the need to support new protocols and functions on top of existing functionality. However, existing hardware could not provide them with the promptness and flexibility desired, as we've seen.

Both software network switches operating on any CPU (PFAFF et al., 2015) and programmable hardware-accelerated devices offer primitives that can be assembled into processing pipelines and easily reconfigured through the use of domain-specific programming languages (BOSSHART et al., 2014; BIFULCO; RéTVáRI, 2018).

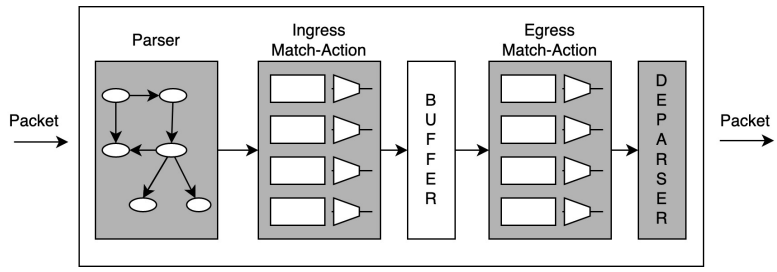
These forwarding devices may be configured to perform actions on the packets. Data plane programmability refers to the ability of the control plane on the SDN to dictate how this packet processing is going to be carried out by these devices. Some manner of influence over forwarding behavior could already be seen on traditional network devices, such as the forced addition of static IP routes on routing tables, but the ability to influence packet processing depending on arbitrary header fields or to modify these headers is exclusive to programmable switches.

One of the most popular languages to program the datapath of programmable data

plane devices nowadays is the P4 Language (BOSSHART et al., 2014), named after Programming Protocol-independent Packet Processors. It is, as the name suggests, protocol-independent, as well as platform-independent. Hence, it is not tied to a specific protocol (e.g.: OpenFlow) or to a target architecture, the P4 compiler can compile code to software-based switches or to hardware devices such as ASICs and FPGAs, given that the manufacturer provides the architecture model of the target device.

To process packets in this manner, P4 initially used the PISA (Protocol Independent Switch Architecture) (CASCAVAL; DALY, 2017). A model of the architecture can be seen in Figure 2.3. It consists of a parser, an ingress match-action pipeline, a buffer, an egress match-action pipeline, and a deparser. The gray elements in Figure 2.3 are programmable using the P4 language.

Figure 2.3: PISA Architecture



Source: The author (2022)

The parser is used to define headers according to either existing or new protocols and parse them using a parse graph. It can be thought of as a state machine that analyzes the packet headers and according to the ones found, tries to locate other headers inside the rest of the payload. The information of which headers are within a given packet is put into packet descriptors (also referred to as metadata).

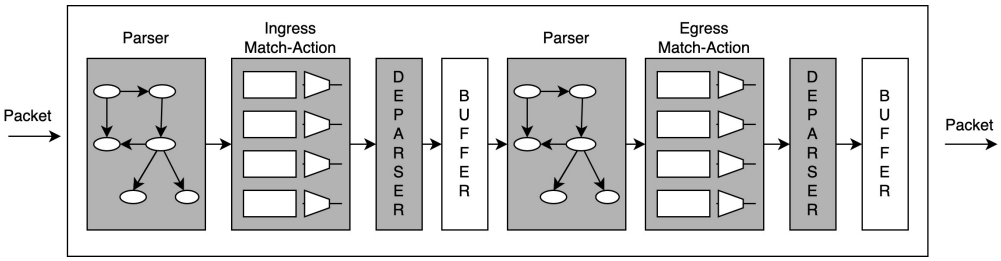
Sequentially, a subset of the packet header fields is used to perform a lookup (match) against a series of hierarchical lookup tables. Each of the entries on the tables has a corresponding action, which is then performed (e.g.: packet rewriting, dropping, forward, or nothing at all), alongside the possibility to alter the internal state of the device (e.g.: updating a packet counter). These table entries can be modified by the controller through the southbound API.

The ingress and egress match-action pipelines have similar purposes, with the difference that the ingress one performs modifications and egress output selection, whereas the egress pipeline modifies packets only. The buffer that intermediates both pipelines is used for packet queuing, replication, and scheduling. Finally, after passing through

the egress pipeline, the packet headers are reassembled in the deparser and serialized for transmission.

Due to a need to target more varied devices, such as NICs and FPGAs, the P4 language has outgrown the PISA architecture. The new model, called PSA (Portable Switch Architecture) extends the capabilities of the PISA one (CASCAVAL; DALY, 2017). It contains six programmable components, with a parser and deparser stage surrounding both the ingress and egress pipelines. The PSA model, with the programmable blocks in gray, may be seen in Figure 2.4.

Figure 2.4: PSA Architecture



Source: The author (2022)

The match-action model described was made popular by OpenFlow (MCKEOWN et al., 2008), but it was limited to a set of previously defined header fields, whereas P4 makes the usage of custom header fields possible. Similarly, other data plane programming technologies require the hardware to be modified with each new version, whereas P4 has the goal of platform and protocol independence. In summary, the P4 language represents an advancement in the extent of programmability of the data plane in comparison to earlier approaches, as well as to the range of target devices available for use.

Many programmable software switches can be emulated in a single machine by the use of network virtualization, which permits the evaluation and testing of network developments quickly and inexpensively. A virtual network consists of network devices and abstractions which are decoupled from their hardware counterparts. Network virtualization permits multiple networks to share a common infrastructure, with logical topologies differing from the hardware underneath (JAIN; PAUL, 2013).

Mininet (LANTZ; HELLER; MCKEOWN, 2010) is one of such environments, permitting the instantiation of multiple virtual hosts, switches, and SDN controllers to create any network topology and testing scenario. Each of those elements uses a different process on the host machine to emulate the corresponding device behavior with the aid of the operating system virtualization features. Due to its powerful CLI used for testing

and Python API used to build custom topologies, Mininet is one of the most popular open-source emulation platforms for virtual networks nowadays. In our study, we use this environment to emulate a diverse set of hosts and switches topologies without the need of multiple physical devices.

3 RELATED WORK

There already are a number of non-centralized and self-managed community networks that provide connectivity to remote communities. The most prominent ones use two technologies to provide the necessary infrastructure: Cognitive Radio (CR) and TV White Spaces (TVWS). While both technologies focus on making use of unexploited resources, both of them need licensing in order to be operated, alongside the proper equipment. Recent research pairs both approaches in order to better exploit the licensed-exempt spectra (BIAN et al., 2019).

Cognitive Radio refers to radio communication that adapts its use of resources to optimize the bandwidth across radio spectra. CR devices can alter their own transmission and reception parameters in order to avoid interference and over-utilization of a given spectrum frequency. Some of the parameters which can be adapted are radio power (which helps avoid interference), frequency (which is chosen based on spectrum sensing) and user-based cooperation (MITOLA, 2002). This is especially interesting for community networks given that many radio bands have been found to be underused (VALENTA et al., 2010) and could be utilized to transmit network packets.

TV White Spaces is the usage of the unused frequency spectra that is located between the band plans of TV channels in VHF and UHF spaces. These white spaces exist naturally between the frequency band allocated to each TV channel, since using directly adjacent frequencies would lead to destructive interference between channels. Since the frequencies allocated to TV broadcast are those below 1 GHz, there are many incentives to using TVWS for providing broadband internet access in remote communities. Because these frequencies are not as high as those on other license-exempt spectra, their signal is able to travel further without losing power, as well as penetrate obstacles easier (RAZA; KULKARNI; SOORIYABANDARA, 2017).

A comparison between this approach and the use of LPWAN may be seen in Table 3.1. Whereas TVWS with CR is capable of supplying a much higher throughput, the LPWAN technology is much better suited for community networks in rural and remote areas. The reason lies mainly in its more advantageous cost, use of unlicensed frequency spectra, and the size of infrastructure needed.

Furthermore, even though attempts to use SDN to manage LPWAN exist in the literature (GALLO et al., 2016; LUO; TAN; QUEK, 2012; BERA et al., 2016; BADDELEY et al., 2018), they are all focused on the IoT scenario and only bring programmability to

Table 3.1: TVWS+CR and LPWAN comparison

	TVWS + CR	*LPWAN*
Range	10-30 km	5-40 km
Cost	Thousands of dollars	Hundreds of dollars
License	Needs licensing of TVWS spaces	Unlicensed spectra
Infrastructure	Large base stations	Small devices
Throughput	2-45 Mbps	300 bps - 50 Kbps

Source: Based on (KHALIL et al., 2017; MEKKI et al., 2019)

the control plane. Thus, this work's proposal lies in bringing data plane programmability to internet access providing LPWAN.

4 VIABILITY ANALYSIS FOR LOW-END LPWAN

Based on the theoretical groundwork laid out in the article entitled "*Programmable Low-End Networks: Powering Internet Connectivity for the Other Three Billion*", an initial viability analysis of the real-life deployment of the low-cost LPWAN for internet access proposed is evaluated (SCHEIBE et al., 2021). The distributed SDN controller architecture for such scenarios proposed in following works is also taken into account (SCHEIBE; GASPARY; CORDEIRO, 2021).

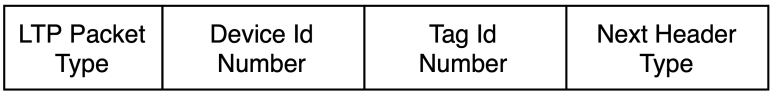
This chapter is organized as follows: firstly, the theoretical basis of the LTP protocol and the distributed controller architecture will be explained; followed by the description of the interface for the LPWAN transceiver developed for this study; finally, the wrapper to communicate incoming packets to the SDN is presented.

4.1 LTP Protocol

The Lightweight Tunnel Protocol was devised to maximize the goodput in unreliable and low-rate wireless connections. This is the case of the connections in a low-cost LPWAN, which have a data rate limitation due to the radio link capacity as well as unreliable connection due to interference in the long-distance transmission. LTP has shown to decrease the overhead in data communication with throughput gains of up to 23% in simulated scenarios (SCHEIBE et al., 2021).

The protocol works by replacing layer two and layer three headers (e.g.: Ethernet and IP) in a TCP/IP flow with a tag identifier. This relies on the assumptions that, *i.* the tags are single-hop only; *ii.* the reliability of the connection, such as packet retransmission and checksum, is provided by upper-layer protocols. This makes the protocol akin to a virtual channel between a pair of devices. So as not to lose information in this compression, the translation of the IP addresses of the communicating hosts into the tag (and vice versa) is done by the network devices through the match-action tables.

Figure 4.1: LTP Packet Header



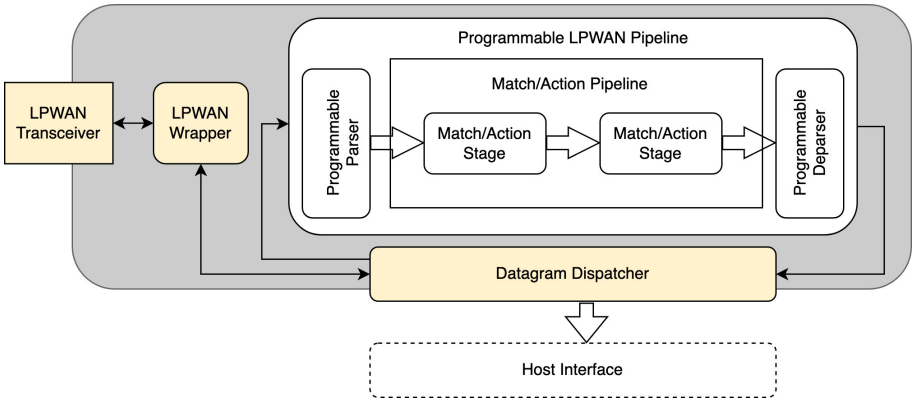
Source: (SCHEIBE et al., 2021)

The LTP packet header can be seen in Figure 4.1. It has a total of 4 fields, with 1 byte each. The fields are as follow:

- **Packet type**: used to differentiate the LTP packets from Ethernet frames
- **Device ID**: identification for the device initiating the tunnel (assuming each device has a unique ID)
- **Tag ID**: the unique identifier given to the tunnel established between a pair of hosts
- **Next Header Type**: the transport layer protocol (same information contained in the original IP header)

The LTP Protocol was designed around a conceptual architecture for programming low-end devices, which can be seen in Figure 4.2. This architecture was designed so to enable any LPWAN device to have data plane programmability easily available. The programmable LPWAN pipeline is implemented using P4 and follows the PISA architecture. The other architectural elements, which are highlighted in yellow, are part of the LPWAN transceiver interface with the SDN's data plane layer and were developed for this this work.

Figure 4.2: Conceptual low-end LPWAN architecture



Source: Based on (SCHEIBE et al., 2021)

The LPWAN Transceiver used in this work is a Ronoth LoStik (RONOTH, 2021) that sends packets over radio. The LPWAN Wrapper component consists of an interface between our programmable software and the antenna, through which it is possible to read incoming data and write outgoing packets. It is also through the wrapper that we are able to configure the antenna attributes (such as power, modulation, and baud rate). Finally, the Datagram Dispatcher is responsible for the packet exchange between the components (the wrapper, and the incoming and outgoing interfaces of the programmable pipeline) as

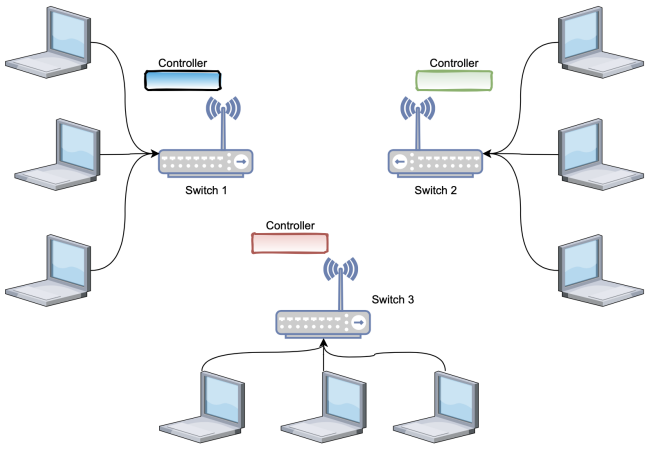
well as for sending the packet to the host.

4.2 Distributed SDN Controller Architecture

Given that LPWAN devices suffer from limited bandwidth, the traditional centralized SDN controller is not a feasible solution for managing a low-end network. This is due to the overhead of management instructions sent from the central control node, which would flood the capacity of the wireless link with admin packets, greatly affecting the goodput. It is then necessary to use distributed control algorithms, with independent and autonomous nodes, capable of relying on unstable communication links (SCHEIBE; GASPARY; CORDEIRO, 2021).

The structure proposed relies on the LTP protocol to build a control plane with an independent controller for each programmable low-end device, as seen in Figure 4.3, with communication between controllers done through the data plane layer itself. The architecture uses the LTP Packet Type field in the LTP Header to differentiate packets in the TCP/IP flow containing only data from packets containing match-action table configuration instructions. The proposed solution also limits the number of network events needed to write network status changes into the forwarding devices, once again, due to the resources constraint.

Figure 4.3: Conceptual SDN distributed architecture



Source: (SCHEIBE; GASPARY; CORDEIRO, 2021)

This structure has the benefit of scalability since it has independent, distributed controllers, as well as reliability, for a failure in one of such controllers would harm only

the device attached to it.

4.3 LPWAN Transceiver

The LPWAN transceiver used in this project is the one mentioned in the works by SCHEIBE et al.: a Ronoth LoStik, which can be seen in Figure 4.4. As of the presentation of this piece of work, one of such devices could be acquired on the company's website for under $60. It has support for both LoRaWAN and for packet-mode LoRa, the latter being the configuration used during this work. It also supports the FSK modulation in radio mode. It is based off Microchip's RN2483 LoRa module, which provides the user with an ASCII user interface that receives commands through a USB serial port. It has a radio range of 15 km in suburban areas and an energy consumption of 20 mA when idle and 140 mA when transmitting.

Figure 4.4: Ronoth LoStik device

Source: (RONOTH, 2021)

This device was firstly analyzed in order to ascertain its capabilities for this use-case, to check how to reproduce the conditions used in the emulated scenarios presented in (SCHEIBE et al., 2021). The first thing that could be ascertained is that the payload size used in the emulated scenarios would not be reproducible in real-life scenarios, as the maximum data size for transmission on LoRa mode with the RN2483 module is 255 bytes (MICROCHIP, 2015). Furthermore, the assumption of a 300 Kbps transmission rate could only be achieved with the FSK modulation, whereas the LoRa mode allows for a maximum of 10937 bps (MICROCHIP, 2017).

4.3.1 Device Interface

The device interface contains three groups of commands: system, MAC, and radio commands. The system commands are preceded by the keyword *sys* and are related to the hardware module, with commands such as resetting, reading the version, sleeping, and firmware upgrades. The MAC ones are related to the LoRaWAN protocol stack and are preceded by the keyword *mac*. Finally, the radio commands are the ones that command the LoRa stack; they are preceded by the keyword *radio* and are executed only after pausing the MAC operations.

Since the native ASCII device interface had many commands, which made their usage confusing, a functional interface was designed in order to more easily operate the antenna, which can be thought of as the LPWAN wrapper part of the conceptual architectural diagram. This interface has five commands, which are described in Table 4.1. It is specifically applicable to the LoRa module used during this project, but it was projected so to be easily replaced by any concrete implementation of a generic antenna.

Table 4.1: High-level antenna module commands

Abstract Command	Description	ASCII Interface Commands
setup	Setup steps for the antenna initialization	*mac reset* *mac pause* *radio set pwr 10* *radio set sf sf7*
enter_rx_mode	Commands to enter the radio receiving mode	*radio rx 0*
enter_tx_mode	Commands to enter the radio transmitting mode	*mac reset* *mac pause* *radio set pwr 10* *radio set sf sf7*
send_data	Steps to send data over radio	*sys set pindig GPIO11 0* *radio tx <data>* *sys set pindig GPIO11 1*
decode_received_data	Steps to decode data received through radio, evoked whenever new data comes through the serial interface	*radio_rx <encoded_data>*

Source: The author (2022)

It is worth noticing that both the incoming and outgoing data sent over radio are hexadecimal values encoded as ASCII strings. The size of the data packet for LoRa modulation is between 0 and 255 bytes.

The current *enter_tx_mode* implementation is below ideal, as it relies on resetting the radio configuration. This is so because the version of the LoRa module used is not the latest, thus missing a firmware upgrade that enabled the command to exit the continuous reception mode. This command, *radio rxstop*, would provide better performance than the current version of entering the transmission mode. This is particularly jarring because the communication is half-duplex, so any time spent resetting the antenna for transmission means missing incoming packets due to the antenna being currently in use.

A different approach considered for this problem involved time-division multiplexing, where a shared communication channel is used by parties in an alternating pattern over time. In this case, a receiving window would be open for an allotted time and after this period, a transmitting time window would be open for some time. This way, each device uses the channel for a fraction of the time in an alternating fashion. This solution would not be good for our use case, since it would significantly diminish the bandwidth available in asymmetrical cases (i.e. when a host downloads more content than it uploads, for example, which is a common occurrence on the internet); furthermore, since the LTP protocol gains rely on the use of the channel link to transmit smaller packets as opposed to their original versions, the usage of time multiplexing could negate its effects, as it would not matter much if a packet was smaller if it had the same time to be sent than a bigger one.

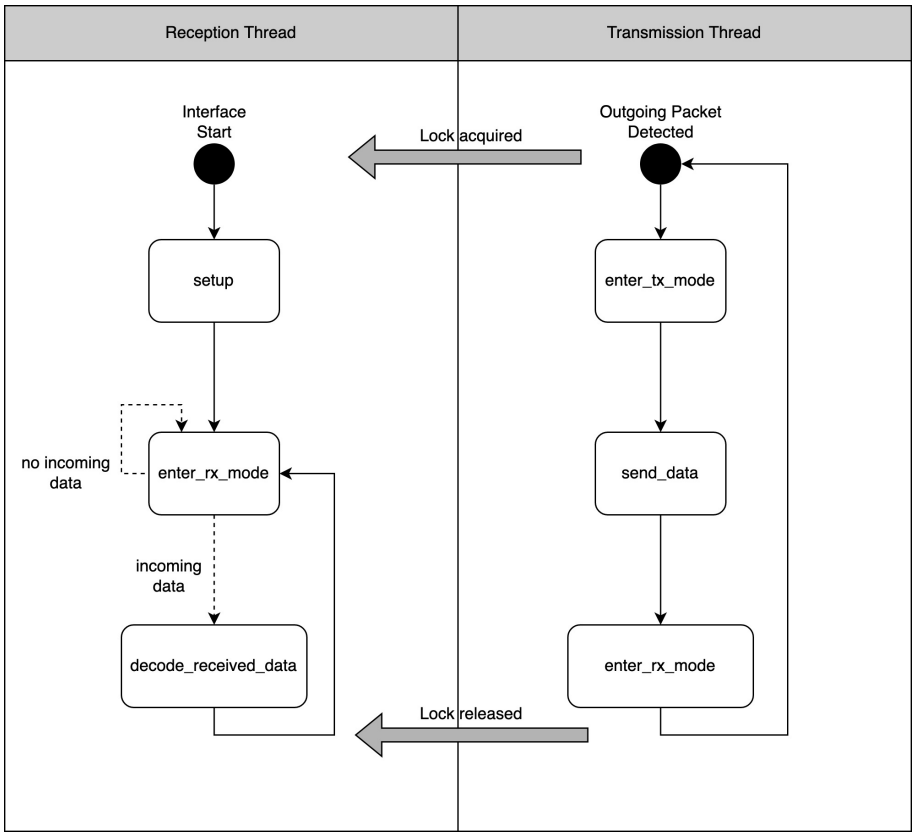
4.3.2 Half-duplex Communication

The transceiver module provided simplex, single-channel communications by default. Since these modules were to be used as antennas on the programmable devices, they had to provide duplex communication. The initial desire to bring full-duplex communication to the link was impossible due to the USB 2.0 connector available on the device. The USB 2.0 standard only provides half-duplex communication at best, which, being a limitation on the physical layer, renders the full-duplex communication unattainable in our interface.

The half-duplex communication is achieved through the use of two threads, with the serial port of the transceiver being a shared resource whose access is controlled through a locking mechanism. A diagram of the operations performed by each thread and the moment where the lock is acquired for the transmission can be seen in Figure 4.5.

The reception thread, also referred to as reader thread, is responsible for handling

Figure 4.5: Reception and transmission threads interaction diagram



Source: The author (2022)

any packets arriving in the network by the antenna. The default state of our LPWAN transceiver was chosen to be a listening mode, i.e. waiting for new packets to arrive by radio. Therefore, the reader thread is the main one, which is constantly running. It is responsible for first and foremost executing the antenna setup instruction upon the start of the interface. Afterward, it enters a loop of executing the instructions to enter the reception mode, which has a timeout duration of roughly a minute, after which, if no packet is received, the instructions to renew the reception state are once again executed. If a data packet is indeed received, however, the instructions to decode the hexadecimal string into a packet of bytes are executed, which finishes the packet reception from the view of the LPWAN wrapper, returning to the reception mode loop.

The transmission thread, also referred to as sender thread, is at first pooling the forwarding device's network interface to check if any outgoing packets are to be sent. While there are none, it does not send any instructions to the LPWAN transceiver. When a packet does arrive at the interface, it starts by acquiring a lock on the transceiver's serial port, as it cannot be accessed concurrently due to the half-duplex nature of the connection.

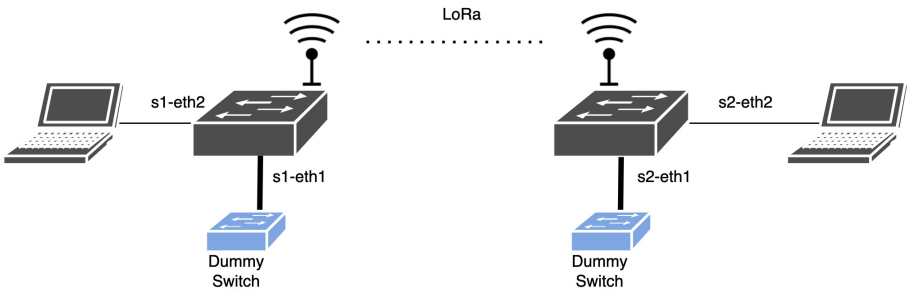
Once the lock is acquired, it executes the instructions to enter the transmission mode, encodes the data to be sent over radio into a hexadecimal string, and sends it over. Before releasing the lock to the reader thread, it executes the instructions to enter the reception mode, so as to make sure that the antenna re-enters the listening mode. Once the lock is released, it returns to the pooling of the network interface for new packets.

4.3.3 Datagram Dispatcher

The link between the virtualized software switch in Mininet and the LPWAN transceiver is done through the datagram dispatcher. This module was implemented through the Scapy Python library. The library is used to manipulate packets, being able to code and decode them, as well as send them through network interfaces. In our case, when we receive a packet through radio, we use the library to send it to a network interface dedicated to the antenna in the switch. Similarly, we use it to sniff the switch interfaces connected to our hosts.

However, since our antenna relies on a serial port, and not an actual network interface (even though it acts as such), it could neither be directly connected to the Mininet topologies, nor used by Scapy. This is why two fake interfaces had to be created on the software forwarding device, both of them connected to a dummy software switch, that drops all incoming packets (since they are actually processed by the datagram dispatcher).

Figure 4.6: Datagram Dispatcher dummy switch mechanism



Source: The author (2022)

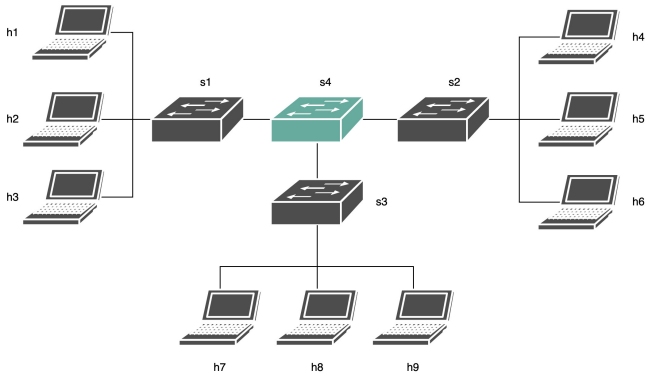
This mechanism is better seen in Figure 4.6, where a dummy switch is connected to each "real" one. This configuration only considers one host per switch for simplicity purposes, although many may be connected at a time. The dummy switch connected to *s1* permits the sniffing and interaction with the interface *s1-eth1*, while the one connected

to *s2* permits the sniffing and interaction with the interface *s2-eth1*. These can be thought of, effectively, as the network interfaces for the antennas.

4.4 Testing Scenarios

The test case scenarios for this work were based on the ones presented in the previous paper, "*Programmable Low-End Networks: Powering Internet Connectivity for the Other Three Billion*" by SCHEIBE et al.. These scenarios had to be modified, as they followed a star topology with a hub replicating switch (i.e. it replicates any packet received through a port into all the others). The hub switch acts as a stand-in for wireless communication, as in such scenarios a packet is not sent over a link to a specific address, but broadcasted through the frequency spectrum to any device in listening range to get.

Figure 4.7: Star topology with hub switch

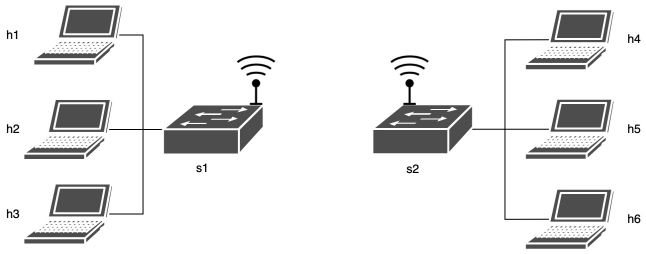


Source: The author (2022)

This configuration, with the hub switch in green, may be seen in Figure 4.7. This topology is a Mininet equivalent of the one in Figure 4.3, where the antennas on the forwarding devices make the use of such a hub irrelevant. Since we only have two LoStik devices available for use, our topology could not include three switches. Thus, the basic topology for our testing scenarios is the one in Figure 4.8, with two switches and three hosts connected to each switch.

Under this topology, the testing scenarios consider two cases. The first case is the standard one, where the protocol stack is either UDP or TCP, and the switches do not have controllers attached to them, but rather fixed forwarding tables. The second case is the one with the LTP protocol and the independent controllers on each switch. Having this in mind, we have a total of 3 testing scenarios in this topology:

Figure 4.8: Basic testing topology



Source: The author (2022)

- **Scenario 1:** host *h1* sending packets to host *h4*, payload size of 88 bytes. Data rate of 1 Kbps.
- **Scenario 2:** host *h1* sending packets to host *h4*, payload size of 128 bytes. Data rate of 1 Kbps.
- **Scenario 3:** host *h1* sending packets to host *h4*, payload size of 200 bytes. Data rate of 1 Kbps.

Considering the combined size of the payload and protocol headers, the payload was capped at 200 bytes because of the maximum data size for transmission on LoRa mode with the RN2483 module (MICROCHIP, 2015), which is 255 bytes. The maximum transmission rate for the RN2438 module is 10937 bps with LoRa mode (MICROCHIP, 2017), however, this considers a perfectly simplex, transmitting only node. This performance cannot be achieved in our case because of *i.* the nature of our half-duplex communication, which takes part of the transmitting capacity by receiving packets; and *ii.* the radio resetting that is needed to enter sender mode, which is composed of many instructions and thus increases the time needed to transmit a message by comparison with a sender-only scenario.

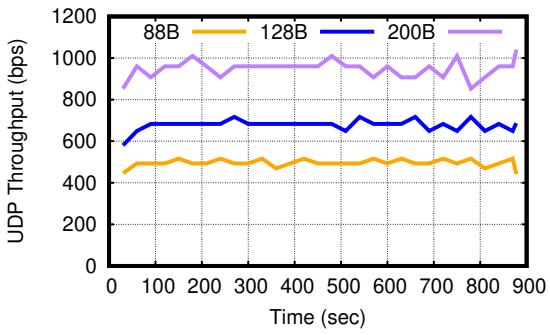
5 RESULTS

The testing scenarios mentioned before have been executed using two physical machines, each with a Ronoth LoStik device located at 10 meters from one another. The virtual machines used for testing had a Ubuntu 16.04 operating system, with 4096 MB RAM memory. Each test case scenario ran for 15 minutes, with measurements made each 30 seconds for a total of 30 measurement points in each experiment. The requests between hosts were made using the iperf3[1] utility program.

5.1 Standard Traffic

The plot of the throughput achieved using the standard switch and protocol headers may be seen below, in Figure 5.1, for the UDP protocol test cases. The test scenario using payloads of 88 bytes is in orange, 128 bytes in blue, and 200 bytes in purple.

Figure 5.1: Standard UDP traffic with varying payload sizes



Source: The author (2022)

It is possible to see in this graph that the throughput increases with the increases in the payload. The mean and median values for the throughput in each test case scenario can be seen in Table B.1 in the Appendix, with values of 496.72 bps, 680.72 bps, and 951.45 bps for the average, for the 88 bytes, 128 bytes, and 200 bytes scenarios respectively.

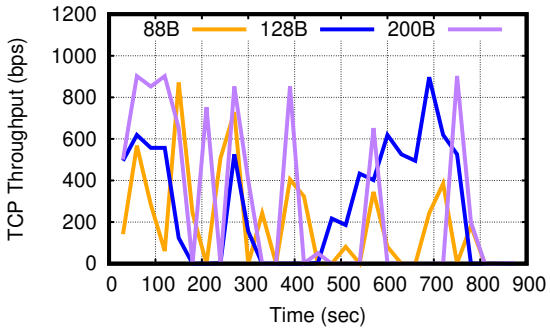
Due to the radio resetting having to be applied to the antenna every time a packet is to be sent, as detailed previously, there is a clear advantage to adding more data to each packet. For example, in the case of the packet with 88 bytes, the resetting process

[1]More information available on https://manpages.ubuntu.com/manpages/xenial/man1/iperf3.1.html

is executed in its entirety for that amount of data; which is the same case for a packet with 200 bytes. This way, it is clear a heavier payload would mean more time spent in productive commands, instead of in configuration.

The plot for the standard testbed with varying payload sizes can be seen in Figure 5.2, for the TCP protocol test cases. As for the UDP graph, the test scenario using payloads of 88 bytes is in orange, 128 bytes in blue, and 200 bytes in purple.

Figure 5.2: Standard TCP traffic with varying payload sizes



Source: The author (2022)

The data obtained in this test case varies too much, and thus, cannot be clearly analyzed. This is probably due to the mechanism this protocol uses to ensure reliable transmission. Put in simple terms, each data packet received must be acknowledged by the receiver by sending a packet with a positive "ACK" signal to the sender. This causes some problems due to the half-duplex nature of the communication. When the receiver device antenna is sending this "ACK" packet to the sender, it cannot receive any incoming packets for the duration of the transceiving operation, which means it has a higher probability of missing packets.

Furthermore, data packets are resent whenever they are missed in the TCP protocol, which makes the time spent on the previous try useless, as the data is only counted in the measurement if it was successfully transmitted and acknowledged by both parties. This mechanism can, therefore, explain the sudden drops in throughput observed in Figure 5.2, as they are probably caused by the packet transmission retrials.

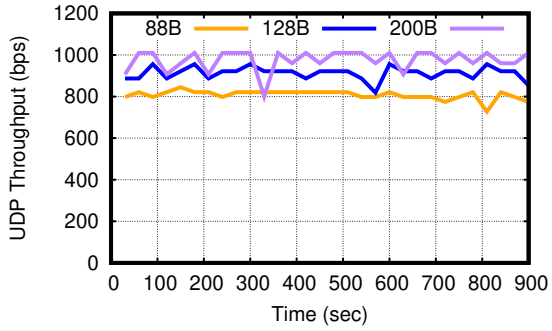
Although, even if we disregard this reliable transmission feature, the inconsistency in TCP throughput could also be due to its congestion control mechanism. This mechanism is put in place to ensure fair and reliable use of a link, effectively avoiding an overload of traffic. However, since it measures congestion by the number of unac-

knowledged packets in a connection, given a connection with a high packet loss rate, this algorithm ends up further limiting the already limited link capacity.

5.2 Using LTP

The plot of the throughput achieved using the LTP protocol with the independent SDN controllers may be seen below, in Figure 5.3, for the UDP protocol test cases. Similarly to the standard case, the test scenario using payloads of 88 bytes is in orange, 128 bytes in blue, and 200 bytes in purple.

Figure 5.3: LTP Protocol UDP traffic with varying payload sizes



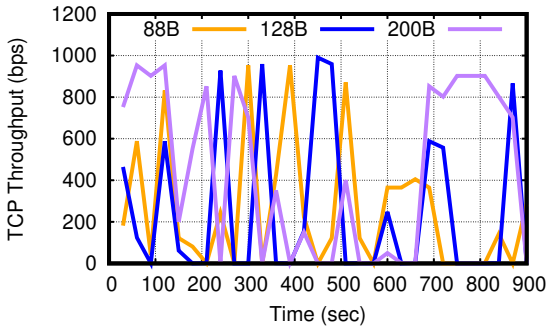
Source: The author (2022)

Similar to the standard test case, we see an increase in throughput with an increase in payload. Much the same logic can be applied here. However, with the usage of the LTP protocol, these metrics are higher for the smaller payload scenarios. The mean and median values for the throughput in each test case scenario can be seen once again in Table B.1 in the Appendix, with values of 808.21 bps, 913.8 bps, and 978.31 bps for the average, for the 88 bytes, 128 bytes, and 200 bytes scenarios respectively.

This perceived increase in performance can be accredited to the smaller size of packets in LTP protocol, since it substitutes the IP and Ethernet headers for a much smaller, 4-byte one. A smaller packet is transmitted more quickly through the communication channel, which in turn augments the throughput. This is especially beneficial in small payloads (such as the ones able to be transmitted through the LoRa modulation), since the header takes up a bigger part of the total packet size in these cases.

The plot for the LTP protocol setup with varying payload sizes can be seen in Figure 5.4, for the TCP protocol test cases. The same as for the other graphs, the test

Figure 5.4: LTP Protocol TCP traffic with varying payload sizes



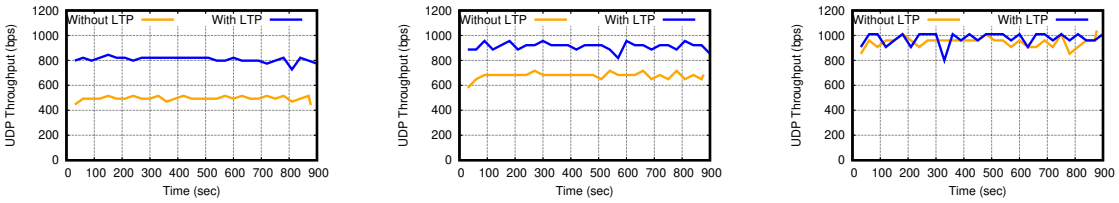
Source: The author (2022)

scenario using payloads of 88 bytes is in orange, 128 bytes in blue, and 200 bytes in purple. Much for the same reasons as in the TCP results for the standard setup, the experiment data varies too much in these tests for any conclusion to be made about the performance of the LTP protocol in comparison to the standard approach using the TCP protocol.

5.3 Comparison

In this section, we compare the performance of the SDN network connection using the LoStik antennas with and without the use of the LTP protocol.

Figure 5.5: LTP and Standard comparison of UDP traffic



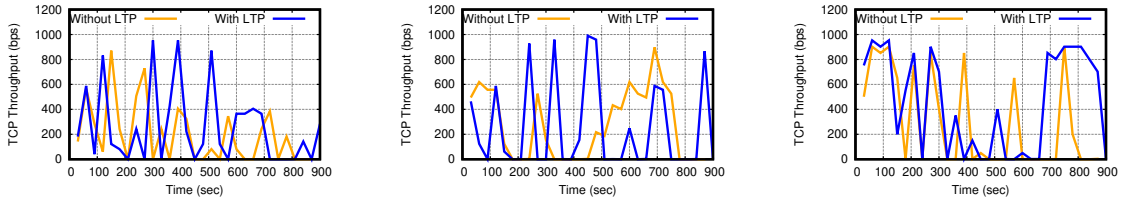
(a) Payload size of 88 bytes (b) Payload size of 128 bytes (c) Payload size of 200 bytes

Source: The author (2022)

5.3.1 UDP

We can see in Figure 5.5 the comparison between the throughput in UDP traffic with and without the LTP protocol. It is visible that the LTP protocol outperforms the standard one in scenarios with the smaller payloads, with a gain of 62.7% with 88-bytes payloads and 34.1% with 128-bytes payloads. This is due to a higher goodput when using LTP, as there is less header overhead. The 200-bytes payload test results are too interleaved for any conclusion on the gain or loss of performance to be made with statistical confidence.

Figure 5.6: LTP and Standard comparison of TCP traffic



(a) Payload size of 88 bytes (b) Payload size of 128 bytes (c) Payload size of 200 bytes

Source: The author (2022)

5.3.2 TCP

We can see in Figure 5.6 the comparison between the throughput in TCP traffic with and without the LTP protocol. As mentioned earlier, these test cases had high variance, so no conclusions can be made using this data.

It can be said, however, that both experiments presented much the same off-and-on characteristic to the throughput with time, which means LTP did not seem to negatively impact this scenario.

6 CLOSING REMARKS AND FUTURE WORK

The developed interface between the low-end LoStik antenna and a virtualized software-defined network was able to provide internet access, even though at low data rates. Furthermore, the results obtained from the interface developed as described in Chapter 4 were conclusive in the ability to improve the capacity of low-speed links through the use of new protocols powered by the programmable data plane.

In this final chapter, we discuss the results obtained throughout this work in Section 6.1, followed by the limitations found in the interface in Section 6.2; and possibilities of future work related to this study are presented in Section 6.3.

6.1 Discussion on Results

The usage of low-end, low-power devices to build programmable LPWAN was envisioned to provide internet access at limited data rates. This is indeed the case found through this viability analysis study, as the link speeds throughout the experiments did not often surpass the throughput of 1 Kbps. Such a throughput, while proving that it is indeed viable to use such a setup for internet access, does not provide enough speed to make it worthwhile. Such a setup could, however, be used in different use-cases, where a high throughput is not necessary, but a standard protocol stack for internet communication is.

On the paper where simulations of this setup were made, the nominal link speeds taken into account were up to 300 Kbps, a much higher speed, even if still low by today's internet standards. The LoStik antenna used here can provide up to 300 Kbps, but while using a different modulation technique, the Frequency-shift Keying (FSK). Nonetheless, this modulation technique had two features that made it unfit for our work: *i.* the maximum payload length transmittable was 64 bytes; *ii.* it does not have the same low-power and long-range characteristics of the LoRa modulation, which would technically put it out of the LPWAN definition.

Nevertheless, even with the limitation faced by the communication channel, it was still possible to reproduce the gain in throughput observed in the simulations of the setup with the advent of the LTP protocol. Since our real-life test case scenarios had more limitations in terms of link speed and payload size (both features that are more beneficial to the LTP protocol performance), we could observe even better gains than the original 23%. The LTP protocol had a gain in speed of 62,7% in comparison to the standard

testbed with 88-byte payloads, with an achieved average speed of 0.8 Kbps; and a gain of 34.1% with an achieved average speed of 0.91 Kbps for 128-byte ones. Thus, the theoretical gains have been successfully verified in this viability analysis, achieving in fact even better performance.

6.2 Interface Limitations

The developed interface had some limitations that were already mentioned previously during this work, but that we repeat here for conciseness' sake. The main limitation is in regards to the achieved throughput. The nominal maximum link speed with the LoRa modulation with the Ronoth LoStik antenna is 10 Kbps, a speed which our interface could not achieve because of the half-duplex nature of the communication established and also because of the antenna resetting having to be made to enter the transmission mode.

These characteristics could be overcome by the usage of a different antenna. The first hurdle by using a physical layer technology with the ability for full-duplex communication. The second, by using a different version of the Ronoth LoStik, which, as mentioned earlier, has a firmware upgrade that enables exiting the continuous reception mode for the transmission one without the need to reset and remake the radio setup.

6.3 Future Work

As for the effort to bring affordable low-power internet access through the use of LPWAN as community networks, there is still much to be done. Firstly, a performance improvement of the communication between forwarding devices is highly needed, which could involve testing other antenna modules and modulation techniques for the communication. Secondly, while the viability analysis was performed by using virtualized switches run in two computer hosts, the objective is to use embedded devices as the forwarding devices, such as a Raspberry PI device. Both the Python interface code and the P4 language run in such an appliance.

Moreover, further tests could be undertaken using the currently developed interface, as many behaviors are yet to be analyzed. Two such cases are the performance of the network link with many LoRa devices connected simultaneously in an area, and how the utilization of a LoStik antenna without the transmission resetting would behave (this

could be achieved through a firmware upgrade as mentioned earlier).

REFERENCES

A4AI. *A4AI 2020 Affordability Report*. 2020. Available from Internet: <https://a4ai.org/affordability-report/report/2020/>.

ALBOWARAB, M. H.; ZAKARIA, N. A.; ABIDIN, Z. Z. Software Defined Network: Architecture and Programming Language Survey. p. 12, 2018.

ALEXANDER, D. et al. The switchware active network architecture. **IEEE Network**, v. 12, n. 3, p. 29–36, 1998.

BADDELEY, M. et al. Evolving sdn for low-power iot networks. In: IEEE. **2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)**. [S.l.], 2018. p. 71–79.

BENZEKKI, K.; FERGOUGUI, A. E.; ELALAOUI, A. E. Software-defined networking (SDN): a survey: Software-defined networking: a survey. **Security and Communication Networks**, v. 9, n. 18, p. 5803–5833, dec. 2016. ISSN 19390114. Available from Internet: <https://onlinelibrary.wiley.com/doi/10.1002/sec.1737>.

BERA, S. et al. Soft-wsn: Software-defined wsn management system for iot applications. **IEEE Systems Journal**, IEEE, v. 12, n. 3, p. 2074–2081, 2016.

BIAN, K. et al. Heterogeneous Coexistence of Cognitive Radio Networks in TV White Space. **arXiv:1902.06035 [cs]**, feb. 2019. ArXiv: 1902.06035. Available from Internet: <http://arxiv.org/abs/1902.06035>.

BIFULCO, R.; RéTVáRI, G. A survey on the programmable data plane: Abstractions, architectures, and open problems. In: **2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)**. [S.l.: s.n.], 2018. p. 1–7.

BOSSHART, P. et al. P4: Programming Protocol-Independent Packet Processors. **arXiv:1312.1719 [cs]**, may 2014. ArXiv: 1312.1719. Available from Internet: <http://arxiv.org/abs/1312.1719>.

BUTLER, K. et al. A Survey of BGP Security Issues and Solutions. **Proceedings of the IEEE**, v. 98, n. 1, p. 100–122, jan. 2010. ISSN 0018-9219, 1558-2256. Available from Internet: <http://ieeexplore.ieee.org/document/5357585/>.

CASCAVAL, C.; DALY, D. **P4 Architectures**. 2017. Available from Internet: <https://p4.org/assets/p4-ws-2017-p4-architectures.pdf>.

CENTENARO, M. et al. Long-range communications in unlicensed bands: the rising stars in the iot and smart city scenarios. **IEEE Wireless Communications**, v. 23, n. 5, p. 60–67, 2016.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The Road to SDN: An Intellectual History of Programmable Networks. p. 13, 2014.

GALLO, P. et al. Sdn@home: A method for controlling future wireless home networks. **IEEE Communications Magazine**, v. 54, n. 5, p. 123–131, 2016.

HONG, C. yao et al. B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined wan. In: **SIGCOMM'18**. [s.n.], 2018. Available from Internet: <https://conferences.sigcomm.org/sigcomm/2018/program_tuesday.html>.

IETF. **IPv6 over Low Power Wide-Area Networks (lpwan)**. 2020. Available from Internet: <https://datatracker.ietf.org/wg/lpwan/about/>.

IRTF GAIA. **IRTF Global Access to the Internet for All Research Group (GAIA)**. 2020. Available from Internet: <https://irtf.org/gaia>.

JAIN, R.; PAUL, S. Network virtualization and software defined networking for cloud computing: A survey. **Communications Magazine, IEEE**, v. 51, p. 24–31, 11 2013.

KHALIL, M. et al. Feasibility, Architecture and Cost Considerations of Using TVWS for Rural Internet Access in 5G. In: . [S.l.: s.n.], 2017.

KLEEN, A. et al. **Linux Netlink as an IP Services Protocol**. [S.l.], 2003. Num Pages: 33. Available from Internet: <https://datatracker.ietf.org/doc/rfc3549>.

KREUTZ, D. et al. Software-Defined Networking: A Comprehensive Survey. **arXiv:1406.0440 [cs]**, oct. 2014. ArXiv: 1406.0440. Available from Internet: <http://arxiv.org/abs/1406.0440>.

LAKSHMAN, T. V. et al. The softrouter architecture. In: **In ACM HOTNETS**. [S.l.: s.n.], 2004.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: Rapid prototyping for software-defined networks. In: **Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks**. New York, NY, USA: Association for Computing Machinery, 2010. (Hotnets-IX). ISBN 9781450304092. Available from Internet: <https://doi.org/10.1145/1868447.1868466>.

LORA ALLIANCE. **What is LoRaWAN®**. 2015. Available from Internet: <https://lora-alliance.org/resource_hub/what-is-lorawan/>.

LUO, T.; TAN, H.-P.; QUEK, T. Q. Sensor openflow: Enabling software-defined wireless sensor networks. **IEEE Communications letters**, IEEE, v. 16, n. 11, p. 1896–1899, 2012.

MCKEOWN, N. et al. OpenFlow: enabling innovation in campus networks. **ACM SIGCOMM Computer Communication Review**, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Available from Internet: <https://dl.acm.org/doi/10.1145/1355734.1355746>.

MEKKI, K. et al. A comparative study of LPWAN technologies for large-scale IoT deployment. **ICT Express**, v. 5, n. 1, p. 1–7, 2019. ISSN 2405-9595. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S2405959517302953>.

MICROCHIP. **RN2483 LoRa Technology Module Command Reference User's Guide**. [S.l.], 2015. Available from Internet: <https://ww1.microchip.com/downloads/en/DeviceDoc/40001784B.pdf>.

MICROCHIP. **RN2483 Low-Power Long Range LoRa TechnologyTransceiver Module**. [S.l.], 2017. Available from Internet: <http://ww1.microchip.com/downloads/en/devicedoc/50002346c.pdf>.

MITOLA, J. Cognitive radio. an integrated agent architecture for software defined radio. 2002.

PFAFF, B. et al. The Design and Implementation of Open vSwitch. p. 15, 2015.

RAZA, U.; KULKARNI, P.; SOORIYABANDARA, M. Low Power Wide Area Networks: An Overview. **arXiv:1606.07360 [cs]**, jan. 2017. ArXiv: 1606.07360. Available from Internet: <http://arxiv.org/abs/1606.07360>.

REYNDERS, B.; MEERT, W.; POLLIN, S. Range and coexistence analysis of long range unlicensed communication. In: **2016 23rd International Conference on Telecommunications (ICT)**. [S.l.: s.n.], 2016. p. 1–6.

RONOTH. **LoSitck - USB LoRa Device – Ronoth**. 2021. Available from Internet: <https://ronoth.com/products/lostik>.

SCHEIBE, A.; GASPARY, L. P.; CORDEIRO, W. Managing programmable low-end wireless networks through distributed sdn controllers. In: IEEE. **2021 13th IFIP Wireless and Mobile Networking Conference (WMNC)**. [S.l.], 2021. p. 32–39.

SCHEIBE, A. et al. Programmable Low-End Networks: Powering Internet Connectivity for the Other Three Billion. **IEEE International Symposium on Integrated Network Management**, p. 9, 2021.

SCHWARTZ, B. et al. Smart packets: applying active networks to network management. **ACM Transactions on Computer Systems**, v. 18, n. 1, p. 67–88, 2000. ISSN 0734-2071. Available from Internet: <https://doi.org/10.1145/332799.332893>.

TENNENHOUSE, D. L.; WETHERALL, D. J. Towards an active network architecture. **ACM SIGCOMM Computer Communication Review**, v. 37, n. 5, p. 81–94, 2007. ISSN 0146-4833. Available from Internet: <https://doi.org/10.1145/1290168.1290180>.

The Linux Foundation. **OpenDaylight: a Linux Foundation Collaborative Project**. 2021. Available from Internet: <https://www.opendaylight.org/>.

TOOTOONCHIAN, A.; GANJALI, Y. HyperFlow: A Distributed Control Plane for OpenFlow. p. 6, 2010.

UNITED NATIONS GENERAL ASSEMBLY. **The promotion, protection and enjoyment of human rights on the Internet**. 2016. Available from Internet: <https://digitallibrary.un.org/record/845728>.

VALENTA, V. et al. Survey on spectrum utilization in Europe: Measurements, analyses and observations. In: **Proceedings of the 5th International ICST Conference on Cognitive Radio Oriented Wireless Networks and Communications**. Cannes, France: IEEE, 2010. ISBN 978-1-4244-5885-1. Available from Internet: <http://eudl.eu/doi/10.4108/ICST.CROWNCOM2010.9220>.

X DEVELOPMENT LLC. **Project Loon website**. 2020. Available from Internet: <https://loon.com/>.

X DEVELOPMENT LLC. **Expanding global access to fast, affordable internet with beams of light**. 2021. Available from Internet: <https://x.company/projects/taara/#>.

YANG, L. et al. **Forwarding and Control Element Separation (ForCES) Framework**. [S.l.], 2004. Num Pages: 40. Available from Internet: <https://datatracker.ietf.org/doc/rfc3746>.

APPENDIX A — SOURCE CODE REPOSITORY

The source code for the implementations and test case scenarios mentioned in this work are available on a public GitHub repository[1] from the day of publication of this work onward. The code is under a MIT License, to be viewed, reused and modified under its terms.

[1] https://github.com/nataliarampon/LoRaStickComm

APPENDIX B — TEST CASES SUMMARY STATISTICS

The following table contains the average and median values of the throughput in each test case scenario.

Table B.1: Summary of experiment measurements

Protocol	Payload Size (bytes)	Standard		LTP	
		Avg	Median	Avg	Median
UDP	88	496.72	493.00	808.21	821.00
UDP	128	680.79	683.00	913.48	922.00
UDP	200	951.45	960.00	978.31	1010.00
TCP	88	191.58	81.00	258.84	1.36
TCP	128	257.35	155.00	242.50	1.36
TCP	200	275.33	1.86	442.87	401. 00

Source: The author (2022)