UNIVERSIDADE FEDERAL DA BAHIA
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANDRÉ LUIZ ROMANO MADUREIRA

IOTP: ON SUPPORTING IOT DATA AGGREGATION THROUGH PROGRAMMABLE DATA PLANES

Salvador
23 de dezembro de 2020

ANDRÉ LUIZ ROMANO MADUREIRA

IOTP: ON SUPPORTING IOT DATA AGGREGATION THROUGH PROGRAMMABLE DATA PLANES

Esta Dissertação de Mestrado foi apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Leobino Nascimento Sampaio

Salvador
23 de dezembro de 2020

Sistema de Bibliotecas - UFBA

Madureira, André Luiz Romano.

IoTP: On Supporting IoT Data Aggregation Through Programmable Data Planes / André Luiz Romano Madureira – Salvador, 2020.

80p.: il.

Orientador: Prof. Dr. Leobino Nascimento Sampaio.

Dissertação (Mestrado) – Universidade Federal da Bahia, Instituto de Matemática e Estatística, 2020.

1. *Software Defined Networks*. 2. *Internet of Things*. 3. *Data Aggregation*. 4. *Data Plane Programming*. 5. *P4*. I. Sampaio, Leobino Nascimento. II. Universidade Federal da Bahia. Instituto de Matemática e Estatística. III Título.

CDD – XXX.XX

CDU – XXX.XX.XXX

TERMO DE APROVAÇÃO

ANDRÉ LUIZ ROMANO MADUREIRA

IOTP: ON SUPPORTING IOT DATA AGGREGATION THROUGH PROGRAMMABLE DATA PLANES

Esta Dissertação de Mestrado foi julgada adequada à obtenção do título de Mestre em Ciência da Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia.

Salvador, 23 de dezembro de 2020

Prof. Dr. Leobino Nascimento Sampaio
Universidade Federal da Bahia

Prof. Dr. Cássio Vinicius Serafim Prazeres
Universidade Federal da Bahia

Prof. Dr. Rodolfo da Silva Villaca
Universidade Federal do Espirito Santo

Dedico este trabalho aos meus pais (Luiz e Cátia) e madrasta (Conceição), que sempre estiveram presentes, fornecendo ensinamentos, amor e incentivos valiosos.

A minha irmã (Ana Luiza), que me apoiou e me ajudou sempre que precisei, mesmo estando distante.

Ao meu irmãozinho (Huguinho), que é a alegra dos meus dias.

ACKNOWLEDGEMENTS

Agradecer não é tarefa simples e fácil, principalmente em se tratando de uma conquista tão grande e sonhada. Não pela falta de motivos, mas sim pelo excesso de razões para ser grato. Faltam-me palavras para expressar minha gratidão a todos àqueles que de alguma forma, direta ou indiretamente, fizeram parte desta trajetória.

Ao meu orientador, Prof. Leobino, pelo suporte, incentivo e orientação vitais para o sucesso dessa jornada.

Aos meus colegas do grupo de pesquisa INSERT, pelos direcionamentos e sugestões que fizeram toda a diferença.

À Fundação de Amparo à Pesquisa do Estado da Bahia (FAPESB) pelo apoio financeiro essencial para a conclusão desta dissertação.

Ao meu pai, Luiz, pelos conselhos, orientação e incentivo. Agradeço muito por ter tido a oportunidade de ser seu filho.

À minha mãe, Cátia, que me faz rir sempre e se fez presente quando mais precisei. Agradeço muito por ser seu filho.

À minha madrasta, Conceição, que acompanhou minha adolescência sendo sempre muito carinhosa. Não pudia ter tido madrasta melhor...

Aos meus irmãos, Ana e Hugo, que eu amo demais! Tem um cantinho reservado em meu coração.

À Leda, por estar sempre presente dando todo o suporte que precisávamos.

À Universidade Federal da Bahia, que me proporcionou um ensino de qualidade, de alto nível e sem custos.

A trajetória do desenvolvimento dessa dissertação foi uma grande caminhada de crescimento pessoal, que se pavimentou com o carinho, apoio e compreensão de muitas pessoas, nem todas puderam ser citadas. Obrigado a todos que, de alguma foram, colaboraram para a concretização deste trabalho.

“ (...) Hoje me sinto mais forte
Mais feliz, quem sabe
Só levo a certeza
De que muito pouco sei
Ou nada sei (...) ”

—ALMIR SATER (Tocando Em Frente)

ABSTRACT

IoT devices generate large continuous data streams, which causes congestion that compromises the scalability of IoT systems. To face this problem, techniques for data aggregation propose to reduce recurring packet headers, through assembly of packet data coming from different sources. Due to the energy constraints and limitation of computational resources of devices, most proposals adjust data aggregation according to their features following multilayered-based approaches or coupling the solution to a given network protocol, but overlooking the properties of the communication link. In this work, we introduce the Internet of Things Protocol (IoTP). An L2 communication protocol for IoT programmable data planes that supports the implementation of data aggregation algorithms inside hardware switches, at the network level. Through these features, IoTP provides support for the design of efficient and adaptable aggregation schemes that can function according to network status and based on the different communication technologies used by IoT devices. We implemented IoTP using the P4 language and conducted emulation-based experiments through the Mininet environment. Our findings show that IoTP accomplishes a 78% improvement in network efficiency, as well as allowing control over the average delay generated by data aggregation techniques. Besides that, it was able to reduce the number of packets sent over the network, while also reducing the consumption of network devices computational resources.

Keywords: Programming Protocol-Independent Packet Processors. Programmable Data Plane. Data Aggregation. Internet of Things. Protocol for Programmable Data Planes.

LIST OF ACRONYMS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CONTENTS

Chapter 1

INTRODUCTION

1.1 CONTEXTUALIZATION AND MOTIVATION

Internet of Things (IoT) devices are increasingly used for various applications (ALKHAMISI; NAZMUDEEN; BUHARI, 2016). Forecasts in this area indicate that hundreds of billions of IoT devices will be connected to networks worldwide by 2020 (YOKOTANI et al., 2017). These devices produce continuous data streams (WISSINGH; D'ACUNTO; TRICHIAS, 2017), generating a large volume of information (WISSINGH; D'ACUNTO; TRICHIAS, 2017; ALKHAMISI; NAZMUDEEN; BUHARI, 2016).

Rahman, Ahmed e Hussain (2016) claim that most of the energy consumption of IoT devices occurs in sending and transmitting packets. Consequently, packet aggregation on the network has traditionally been carried out within IoT devices to reduce the number of communications with the base station, as illustrated in Figure 1.1. Packet aggregation on the network is a technique that combines several packets with heterogeneous requirements and characteristics. This combination allows a single packet to be transmitted to the next network node (AKYUREK; ROSING, 2018). By reducing the number of packets on the network, this aggregation technique also manages to reduce the energy consumption of the devices, increase the efficiency of the network (AKYUREK; ROSING, 2018; AKYUREK;

ROSING, 2016) and eliminate the recurring headers of the packets. There is also a reduction in latency, jitter, and end-to-end delays (AKYUREK; ROSING, 2018).

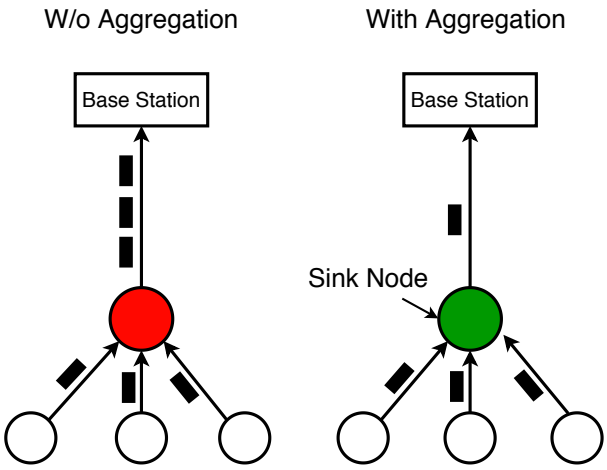


Figure 1.1: Legacy IoT networks (RAHMAN; AHMED; HUSSAIN, 2016)

1.2 PROBLEM STATEMENT

In Internet of Things (IoT) scenarios, the majority of device power consumption occurs in transmitting packets (RAHMAN; AHMED; HUSSAIN, 2016). Packet aggregation on such networks is traditionally performed within the IoT devices in order to reduce the number of communications with base stations. Network packet aggregation is a technique that combines multiple packets, possibly from different sources, with heterogeneous requirements and characteristics. This combination allows a single packet to be transmitted to the next network node. By reducing the number of packets on the network, aggregation techniques can also increase network efficiency, and eliminate recurring packet headers. In addition, packet aggregation can reduce end-to-end latency and jitter (AKYUREK; ROSING, 2018).

When performing aggregation, IoT devices that operate as sink nodes require more computational resources and present a higher power consumption pattern than other nodes in the network (AKYUREK; ROSING, 2018). Therefore, hierarchical (i.e., tree topology) and clustering strategies carry out the aggregation through a set of interconnected IoT devices (RAHMAN; AHMED; HUSSAIN, 2016; Tortonesi et al., 2016;

HEINZELMAN; CHANDRAKASAN; BALAKRISHNAN, 2002; SAPIO et al., 2017; KARIM; AL-KAHTANI, 2016; SHEN et al., 2017; WISSINGH; D'ACUNTO; TRICHIAS, 2017), instead of solely sink nodes. Besides energy constraints, the limitation of computational resources also restricts the execution and efficiency of aggregation algorithms.

Solutions like the Priority-based Dynamic Data Aggregation scheme (PDDA) face resource constraint-related problems by adopting multilayered hierarchical big data aggregation. They enable the deployment of different aggregation schemes according to the available computational resources (e.g., memory, data processing capability, and processing delay) of the devices in each layer of the network (KARIM; AL-KAHTANI, 2016). Although being able to adopt different techniques, these approaches, however, overlook the specifics of the communication technologies involved (e.g., BLE, ZigBee, 802.11, and LoRa). Conversely, other proposals implement aggregation mechanisms that adapt themselves to the communication link by coupling the solution to the supported network protocol. But, the degree of dependence on one specific network protocol makes them harder to apply in different networking scenarios. Therefore, the literature reports some attempts to use generic architectures through which different communication technologies and heterogeneous IoT devices are integrated to facilitate both data transmission and aggregation (SHEN et al., 2017).

Despite the contributions of the initiatives mentioned above, current data aggregation strategies can reap the benefits of data plane programmability through technologies that can provide a richer set of network information that had not been available so far. Based on the features of the communication link, algorithms can rely on optimization techniques to accomplish a higher data aggregation efficiency. For example, by using the network's Maximum Transmission Unit (MTU), the aggregation can exploit the available payload to reduce the total amount of packets sent to the network. The reduction of the number of packets sent results in less network congestion. Thus, this work's main problem statement is **how to process network and device information to support data aggregation on IoT networks?**

1.3 HYPOTHESIS

Hence, based on the understanding that the network can provide a rich set of information for data aggregation solutions, our hypothesis is that a link-layer network protocol (Layer 2) can empower data aggregation strategies with network properties information, such as MTU, link bandwidths, delays, and underlying communication technology, embedding them into networking devices. In doing so, highly efficient data aggregation can be deployed within the programmable data plane of network hardware to mitigate delays and congestion in IoT networks.

1.4 GOALS

1.4.1 Main goal

This work aims to develop an adaptive network protocol that supports multiple communication technologies and empowers data aggregation strategies with network information, aiming to mitigate congestion and improve the efficiency of IoT networks.

1.4.2 Specific goals

- Enable the implementation of a distributed data aggregation architecture within the network to allow the execution of data aggregation strategies before, during, and after sending data from IoT devices;
- Allow the implementation of data aggregation strategies directly on switches based on programmable data plane hardware, capable of reducing the number of packets that travel through the network and the overhead caused by repeated packet headers, while improving network efficiency, as measured by the ratio of total payload by total data transmitted;
- Control the average delay associated with the aggregation strategy implemented in the network switches;

1.5 METHODOLOGY

Initially, an extensive bibliographic study was carried out in the main databases of the computer science research field (e.g., IEEE[1], ACM[2] and Elsevier[3]) about Internet of Things (IoT) and Software Defined Network (SDN). Throughout this study, it can be verified that current research efforts have geared toward SDN solutions for IoT wireless sensor environments (BERA; MISRA; VASILAKOS, 2017). After obtaining this first information, the research focused on such scenarios to list the main problems of relevant research. Thus, it was observed that the problems tend to group around typical issues, such as network management, network function virtualization, ubiquitous information access, resource utilization control, energy management, security, and privacy, among others (BERA; MISRA; VASILAKOS, 2017).

Among the main open research topics, it was decided to investigate IoT scalability, due to the facts already presented in the previous sections. Some studies have already been proposed with this theme, however, several others suggest that the issue of scalability in IoT still needs to be further investigated. With the research progress, it was noted that the massive amount of data transmitted through IoT networks cause several network issues, resulting in high network latency and congestion. This issue was raised by Karlsson, Kassler e Brunstrom (2009) and also studied in Akyurek e Rosing (2018).

Once the problem was identified, studies began to develop a solution, based on data aggregation, that would provide an improvement in the network's performance regarding the overall network congestion. Also, a scenario planning was started to validate the proposal, covering IoT devices, in this case, as in Karim e Al-kahtani (2016), we opted for an environment of Wireless Sensor Network (WSN) infrastructure.

To validate the research, the feasibility of the three basic techniques for validating scientific research was considered: measurement, analytical modeling, and simulation/emulation (JAIN, 1990). A high amount of IoT devices is required to statistically assess

[1]IEEE Xplore Digital Library. Available at: ⟨https://ieeexplore.ieee.org/Xplore/home.jsp?reload=true⟩. Last accessed on July 01, 2020.

[2]ACM Digital Library. Available at: ⟨https://dl.acm.org/⟩. Last accessed on July 01, 2020.

[3]Compendex Engineering Index. Available at: ⟨https://www.engineeringvillage.com/search/quick.url⟩. Last accessed on July 01, 2020.

the network performance degradation associated with their data transmission in wireless scenarios and, for this reason, the measurement technique has become unviable. The analytical modeling technique, in turn, is complex to model dynamic network environments and it also has a high abstraction from reality. On the other hand, the emulation technique presents a low cost, high accuracy, in addition to high portability of simulated algorithms for the real system. For these reasons, the proposal was validated by means of emulation.

After choosing the validation technique of the proposal, a research was carried out on network emulators based on SDN, which are characterized by lightweight nodes composed of virtual NICs constructed using Linux kernel inherent features. Subsequently, an extensive bibliographic study was carried out in the main databases in the computer science area to establish which emulator is more frequently employed in SDN network scenarios. The Mininet emulator was considered the most suitable for the development of this research, as it is by far the most frequently cited emulator regarding SDN researches (KREUTZ et al., 2015). Then, there was a thorough study in the documentation of Mininet (LANTZ; HELLER; MCKEOWN, 2010) and from that, the specifications and the development of the scenario and the proposal started. At the end of this stage, a careful performance analysis (JAIN, 1990) of the solution was planned. After that, the experiments were started to generate the network pcap dump files, which then were submitted to an evaluation procedure under a quantitative data analysis.

1.6 CONTRIBUTIONS

The main contributions of this dissertation are the following:

- A protocol that allows data aggregation solutions to be deployed directly in the data plane through programmable hardware-based switches;
- A protocol that provides support for data aggregation algorithms that can use the network information provided by the proposed protocol to reduce network communication overhead, while improving network's efficiency, as measured by the ratio

of total payload by total data transmitted;

- A protocol that can prioritize the IoT data retrieval, by controlling data aggregation delays.

1.7 ORGANIZATION

The rest of the work is organized as follows: Chapter 2 presents the theoretical framework that underlies the research. The methodology and implementation details are discussed in Chapters 3 and 4 respectively. The evaluation environment, proposal's evaluation and results achieved are described in Chapter 5. Finally, this work is concluded in Chapter 6.

Chapter

2

FUNDAMENTALS AND RELATED WORKS

2.1 LEGACY NETWORKS

For the Internet to work, many devices like routers, switches, and middleboxes are required (GUPTA; KAUR; KAUR, 2016; LAISSAOUI et al., 2015). Most of these devices contain proprietary and vendor-specific network operating systems, and implementing new features and protocols involves years of effort and testing. Each of them must be configured individually and manually according to the guidelines provided by the suppliers (GUPTA; KAUR; KAUR, 2016; LAISSAOUI et al., 2015). In this scenario, network operators are responsible for configuring the network, imposing several high-level policies to respond to the wide range of events that occur in it (for example, traffic changes, intrusions) (KIM; FEAMSTER, 2013). In this context and taking into account the vastness of equipment used, contemporary networks have become very difficult to manage and scale, becoming very complex (GUPTA; KAUR; KAUR, 2016; STANCU et al., 2015). Besides, the initial cost of purchasing devices and running costs are very high (GUPTA; KAUR; KAUR, 2016). There was also a significant increase in the number of mobile devices, along with the content accessed (STANCU et al., 2015).

The network protocols are generally organized into three planes: data, control, and management. The data plane consists of all messages generated by users. To carry these

messages, the network needs to perform some tasks such as finding the shortest path, using L3 routing protocols such as Open Shortest Path First (OSPF) or L2 as Spanning Tree Protocol (STP). The messages used for this purpose are called control messages and are essential for the operation of the network. Additionally, the network administrator may wish to monitor traffic statistics and the status of the various network devices. Such monitoring is performed by the management plane (JAIN; PAUL, 2013).

In traditional switches and routers, the control and data planes are embedded in the same device (GUPTA; KAUR; KAUR, 2016; SEZER et al., 2013). Due to the high integration of these planes, the introduction of new applications and features becomes very difficult (GUPTA; KAUR; KAUR, 2016; KREUTZ et al., 2015). Since there is no common control interface to handle all devices (GUPTA; KAUR; KAUR, 2016), the implementation of new features needs to be performed in a device-by-device manner. To perform such modifications, it is necessary to install new firmware and, in some cases, updates to the device hardware. Thus, new network resources are traditionally introduced through expensive, specialized, and complex configuration equipment, known as middleboxes. Examples of these middleboxes are load balancers, Intrusion Detection Systems (IDSs), firewalls, among others. These devices need to be strategically placed on the network, making it even more difficult to change the topology, configuration, and functionality of the latter (KREUTZ et al., 2015).

2.2 SOFTWARE-DEFINED NETWORKING (SDN)

SDN aims to mitigate the limitations of traditional networks, such as complexity, dependence on suppliers, inconsistencies in network policies, scalability problems (STANCU et al., 2015), and configuration errors (XIA et al., 2015). SDNs are programmable networks, whose goal is to simplify network management (GUPTA; KAUR; KAUR, 2016), which provides several benefits, such as:

- Centralized control (KREUTZ et al., 2015; SEZER et al., 2013);
- Facilitated evolution, innovation (KREUTZ et al., 2015) and network management

(HU; HAO; BAO, 2014);

- Facilitated network policies modifications through high-level languages (KREUTZ et al., 2015);
- Improvements in the coherence of network policies (XIA et al., 2015), keeping them intact as changes in the state of the network occur (KREUTZ et al., 2015);
- Better usage of network bandwidth and resources (GUPTA; KAUR; KAUR, 2016).

Unlike traditional networks, in SDN there is no need to solve trivial problems over and over again, such as network topology discovery, network state distribution, and network resilience. It is significantly easier to develop applications for these cohesive control platforms than to do the same for a multitude of standalone applications, which work on heterogeneous routing elements (YEGANEH; TOOTOONCHIAN; GANJALI, 2013). This characteristic derives from the fact that SDNs are based on a centralized intelligence (TANTAYAKUL et al., 2017), which collects information about the network status and transmits this data to the SDN applications, as illustrated in Figure 2.1 (XIA et al., 2015). In this scenario, the key component of an SDN architecture is the controller, which coordinates and manages all devices on the network. In SDNs, the network operation is divided into two parts: a control plane and a data plane (TANTAYAKUL et al., 2017).

An SDN network proposal consists of:

- **Separate control and data planes**: Control and data planes are decoupled (XIA et al., 2015; KREUTZ et al., 2015). Control functionality is removed from network devices, making them easily replaceable packet forwarding machines (KREUTZ et al., 2015).
- **Centralized control plane**: The control plane is now centralized, with a global view of the network (GUPTA; KAUR; KAUR, 2016; RASTOGI; BAIS, 2016). For this, a Network Operational System (NOS), also called an SDN controller, is introduced in the network, taking responsibility for the control plane decisions (GUPTA;

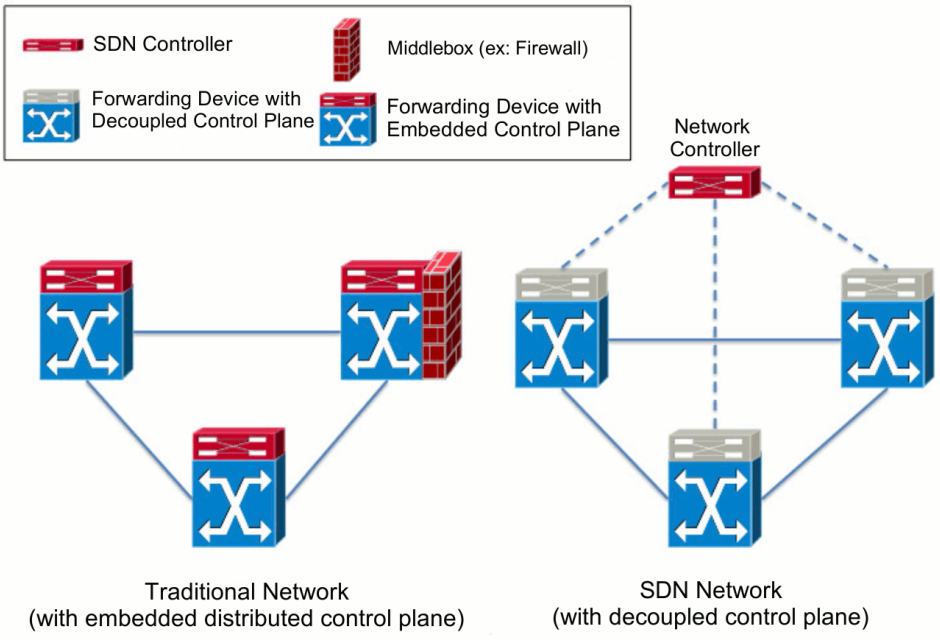


Figure 2.1: Comparison between Legacy Networks and Software Defined Networks (NUNES et al., 2014)

KAUR; KAUR, 2016; KREUTZ et al., 2015). This external entity is a software platform that, based on the global view of the network, provides essential resources and abstractions to facilitate the programming of forwarding devices (KREUTZ et al., 2015).

- **Dynamic Network Adjustment**: In SDNs, the network traffic flow is dynamically adjusted by the control plane to meet the current network needs (GUPTA; KAUR; KAUR, 2016). For this, applications that run on top of NOS (GUPTA; KAUR; KAUR, 2016; KREUTZ et al., 2015) interact with the programmable control plane (GUPTA; KAUR; KAUR, 2016; RASTOGI; BAIS, 2016) via the Northbound API. Then, the control plane interacts with the underlying data plane devices (KREUTZ et al., 2015; STANCU et al., 2015).

The SDN architecture was subdivided into three layers: Application Layer, Control Layer and Infrastructure Layer, as depicted in Figure 2.2 (RASTOGI; BAIS, 2016). Such layers are also called application plane, control plane (GUPTA; KAUR; KAUR, 2016) and data plane (GUPTA; KAUR; KAUR, 2016; ARBETTU et al., 2016), respectively.

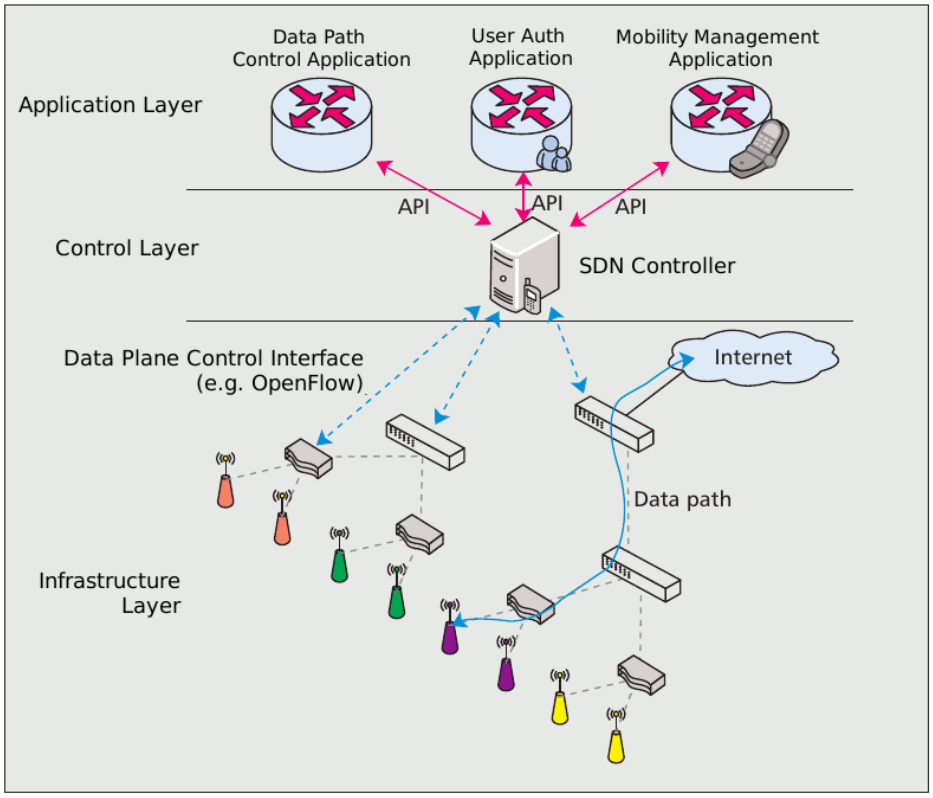


Figure 2.2: SDN Architecture Overview (BERNARDOS et al., 2014)

2.2.1 Infrastructure layer

This is the layer responsible for delivering packets using the Forwarding Information Base (FIB) tables (JAIN; PAUL, 2013). This layer is mainly composed of routing devices, either physical, such as switches and routers (RASTOGI; BAIS, 2016), or virtual, such as software (KREUTZ et al., 2015).

In SDN networks, routing devices receive instructions from SDN controllers via Southbound Interface (SBI) API (KREUTZ et al., 2015). These instructions are stored in flow tables (ARBETTU et al., 2016), in the form of rules that represent a set of elementary operations to be performed on packets received by the forwarding devices (KREUTZ et al., 2015). Examples of operations that can be performed are forwarding, drop, or rewriting a packet header (KATTA et al., 2016; KREUTZ et al., 2015).

2.2.2 Control Layer

In Software-Defined Networks, network control is performed by a separate and centralized SDN controller (KATTA et al., 2016; GUPTA; KAUR; KAUR, 2016). This controller represents the control layer (KREUTZ et al., 2015), responsible for defining the behavior of the devices present in the infrastructure layer (RASTOGI; BAIS, 2016). For this, the SDN controller collects information about the network (SEZER et al., 2013), and, based on this information, it sets up forwarding rules in the data plane, using the Southbound Interface (SBI) API (PAKZAD; PORTMANN; HAYWARD, 2015; XIA et al., 2015).

2.2.3 Application Layer

This is the top layer of an SDN architecture where the network administrator can operate (RASTOGI; BAIS, 2016). The requirements of the entire network can be configured dynamically at this layer using custom SDN applications (RASTOGI; BAIS, 2016; XIA et al., 2015). Through this programmable platform, SDN applications can access and control switching devices in the infrastructure layer (XIA et al., 2015). Consequently, different functionalities can be implemented in the (GUPTA; KAUR; KAUR, 2016; STANCU et al., 2015) network, allowing all the control requirements of current networks to be implemented in SDN (GUPTA; KAUR; KAUR, 2016) networks. As an example of SDN applications we have dynamic access control, continuous mobility and migration, server load balancing, and network virtualization (XIA et al., 2015).

2.2.4 OpenFlow Protocol

The OpenFlow (OF) was designed to achieve the purposes of SDN networks, such as dynamic routing configuration and abstracting the details of the underlying hardware (KREUTZ et al., 2015). This protocol allows the network controller to install or delete switch forwarding rules, as well as request flow or port statistics (PAKZAD; PORTMANN; HAYWARD, 2015). Consequently, OF is a protocol that implements the SBI of SDN networks (PAKZAD; PORTMANN; HAYWARD, 2015; NUNES et al., 2014).

However, the OpenFlow community quickly recognized that some specific operations, which depend on the state of the network, need to be performed directly by the network switch itself, within its data plane. This approach reduces the overhead caused by excessive switch communication with the SDN controller, mitigating the associated latency problems (DARGAHI et al., 2017).

Additionally, OF is highly coupled with other network protocols. That is, the OF match-action tables are designed for specific network protocols, supported by a specific OF version, such as the IP protocol, Ethernet, and among others. To provide support for a greater number of network protocols, OF has increased the number of supported header fields. In its most recent version (v1.5.1), the protocol supports more than 40 packet header fields (Layers 2, 3, and 4). However, this approach creates barriers to network innovations, as it limits the progress of OpenFlow since it's highly coupled with other network protocols (BAKTIR; OZGOVDE; ERSOY, 2018).

2.2.5 P4 Data Plane Programming Language

In this scenario, the P4 data plane programming language (BOSSHART et al., 2014) was developed in order to increase the degree of flexibility of SDN networks, mitigating their limitations (BAKTIR; OZGOVDE; ERSOY, 2018). Recent studies prove that P4 is able to act as a facilitator for the innovation of network protocols (ZHANG et al., 2017) since this technology is a language that facilitates the data plane programming of network switches (SVIRIDOV et al., 2018; HYUN; HONG, 2017). Thus, the way in which each packet will be processed within the switches can be defined and implemented within P4, allowing this technology to adapt to multiple protocols (BAKTIR; OZGOVDE; ERSOY, 2018; ZHANG et al., 2017) as they are designed. For this reason, P4 is able to make switches independent of network and vendor protocols (BAKTIR; OZGOVDE; ERSOY, 2018; SVIRIDOV et al., 2018). In addition, P4 also allows a switch to perform actions based on information previously stored in its memory and facilitates in-field reprogrammability (SVIRIDOV et al., 2018) and In-band Network Telemetry (INT).

Unlike OpenFlow, P4 features a programmable packet parser, which allows switches

to analyze new header fields (DARGAHI et al., 2017; HYUN; HONG, 2017). Such a feature makes the deployment of custom protocol headers easier (SVIRIDOV et al., 2018; HYUN; HONG, 2017) while at the same time it allows packet payload processing within the switch (SVIRIDOV et al., 2018).

The P4 language is designed to operate at line rate and allow more efficient use of the computational resources available in switches (SVIRIDOV et al., 2018). For this reason, P4 allows parallel, sequential, and hybrid (sequential and parallel) execution of actions and match-action tables. In OpenFlow, the execution of these tables and actions is mandatory sequential (DARGAHI et al., 2017).

2.2.6 Simulators / Emulators

To perform tests on physical computer networks, multiple hosts, switches, and controllers need to be interconnected (GUPTA; KAUR; KAUR, 2016), which represents significant costs (TANTAYAKUL et al., 2017; KREUTZ et al., 2015). Thus, simulation and emulation tools are used to reduce these associated costs related to computer network testing. Another important factor is the ease of data collection and analysis within virtualized environments such as those described above (TANTAYAKUL et al., 2017).

There are several options available for network emulators and simulators (TANTAYAKUL et al., 2017; GUPTA; KAUR; KAUR, 2016), such as NS3, EstiNet, Mininet and OpenNet (TANTAYAKUL et al., 2017). Mininet is the first system that provides a quick and easy way to prototype and evaluate SDN protocols and applications (KREUTZ et al., 2015). This characteristic, together with its powerful network emulation environment (HU; HAO; BAO, 2014), has made Mininet one of the most used emulator tools in research related to SDN (XIA et al., 2015).

2.2.6.1 Mininet

The Mininet (LANTZ; HELLER; MCKEOWN, 2010) emulator allows the creation of many topologies, being able to emulate hundreds of thousands of nodes and switches

(GUPTA; KAUR; KAUR, 2016). To perform the emulation, Mininet uses a lightweight virtualization technique, combining many of the optimal features of emulators, testbeds, and simulators (GUPTA; KAUR; KAUR, 2016). In this way, Mininet is a more feasible and better available solution than traditional testbeds (GUPTA; KAUR; KAUR, 2016). Additionally, although Mininet was initially developed to perform experiments using OF, it can also be used for experiments with other technologies, such as P4 (BAKTIR; OZGOVDE; ERSOY, 2018).

There are several advantages to using Mininet, such as:

- **Custom Topologies:** In the environment provided by Mininet different topologies can be created, from large topologies similar to smaller ones (GUPTA; KAUR; KAUR, 2016);
- **Running Real Programs:** The Mininet hosts can run any software that is executable on a Linux system. That way, tools like *tcpdump*, *curl*, *elinks*, *wireshark* and etc can be used without any modification (GUPTA; KAUR; KAUR, 2016);
- **Shareable Code and Replicable Results:** The codes used in the experiments can be shared and the results are readily replicable, which facilitates the validation of results obtained in experiments (GUPTA; KAUR; KAUR, 2016);

2.3 INTERNET OF THINGS (IOT)

The management of Internet of Things networks (IoT) is essential to enable its operation since such networks have billions of IoT devices, generating large volumes of data. Thus, it is of fundamental importance to distribute and control traffic flows on the network, performing load balancing, and minimizing the communication delay. Such requirements can be met by SDN technology, as it can obtain a global view of the network in a logically centralized way (BERA; MISRA; VASILAKOS, 2017). In this sense, there have been several studies on the adoption of SDN in IoT networks (KIM; MIN; HAN, 2017).

Typically, an IoT network is composed of a set of embedded systems, equipped with sensors and actuators. These large-scale distributed systems support real-time applica-

tions, such as smart healthcare, transport, and energy systems (BERA; MISRA; VASILAKOS, 2017), generating continuous data flows (WISSINGH; D'ACUNTO; TRICHIAS, 2017).

However, IoT networks have significant requirements and challenges, such as heterogeneity/interoperability and scalability (ARSHAD et al., 2019).

2.3.1 Heterogeneity and Interoperability

IoT devices are heterogeneous in nature (ARSHAD et al., 2019; BERA; MISRA; VASILAKOS, 2017), and usually have varying computational resources such as memory amount, processing capacity, and battery life. Also, communication between these sensors is carried out by different underlying technologies (wired, wireless, Bluetooth, 4G, CRN, opportunistic networks among others) (ARSHAD et al., 2019). The data format sent by IoT devices is also different, due to the specificities of each manufacturer (BERA; MISRA; VASILAKOS, 2017). Therefore, an IoT network architecture needs to be designed to support the inherent heterogeneity in IoT devices (ARSHAD et al., 2019; BAKTIR; OZGOVDE; ERSOY, 2017), taking into account the computational resources available in each system (ARSHAD et al., 2019) and the different communication technologies used (ARSHAD et al., 2019; BAKTIR; OZGOVDE; ERSOY, 2017). In this way, the IoT network can achieve interoperability (ARSHAD et al., 2019), facilitating the exchange of information between devices in a unified way (BERA; MISRA; VASILAKOS, 2017).

To achieve interoperability between IoT devices, a unified data collection mechanism (BERA; MISRA; VASILAKOS, 2017) and a vendor-independent environment, such as the one proposed by SDN networks, are necessary. In SDNs, different types of sensors can communicate in a single environment since the network can manage different technologies, such as WSNs and Body Area Networks (BANs) (BAKTIR; OZGOVDE; ERSOY, 2017).

2.3.2 Scalability

The IoT architecture possesses great challenges regarding scalability (ARSHAD et al., 2019) since it is expected that the number of devices will reach trillions in the years to come (BAKTIR; OZGOVDE; ERSOY, 2017). Thus, IoT networks need to support a huge number of devices efficiently (ARSHAD et al., 2019; YOKOTANI et al., 2017). Although there are solutions, such as IPv6, for addressing the vast number of IoT devices, this problem is not the only obstacle to the scalability of these networks (ARSHAD et al., 2019).

In 2015, approximately 97 million portable devices generated 15 petabytes of traffic per month and this large volume of transmitted data is expected to grow exponentially over the years (BAKTIR; OZGOVDE; ERSOY, 2017). In addition to the enormous volume of transmitted data, IoT devices also transmit a large number of packets at the edges of the network. Consequently, this pattern generates an increase in the processing load on switching devices (YOKOTANI et al., 2017). Despite recent advances in the transmission capacity of networks and the processing capacity of switches, it is debatable whether these advances are sufficient to provide an efficient communication service on IoT networks (YOKOTANI et al., 2017). So, the huge amount of data being produced by IoT devices needs better and more efficient scalability management (ARSHAD et al., 2019; ALKHAMISI; NAZMUDEEN; BUHARI, 2016). In particular, it is necessary to reduce the number of packets on the network to mitigate the processing overhead inflicted on the network switches (YOKOTANI et al., 2017).

One of the current approaches that aim to manage the huge amount of data produced by IoT devices is the use of specialized gateways. These machines collect data from IoT devices and store them, usually in an aggregate form in data centers, where the data can be accessed by users' applications (WISSINGH; D'ACUNTO; TRICHIAS, 2017).

The use of Fog Computing techniques, as described in the previous paragraph, is essential for wireless IoT networks, since IoT devices generally have limited capacity for storage and processing (BAKTIR; OZGOVDE; ERSOY, 2017). Besides, the communication of IoT devices can be intermittent, as they operate on batteries and can suffer

network attacks such as Denial Of Service (DoS) (ARSHAD et al., 2019).

Despite the importance of gateways for managing IoT data, the fact that they are specialized and work with data at the application layer level makes the aggregation scheme inflexible. Such inflexibility stems from the fact that the aggregated data is excessively customized to meet the needs of specific IoT applications and networks. For example, such gateways can be designed to operate with a specific IoT protocol, such as 802.15.4, and with a specific IoT application, such as forest fire detection. Consequently, data aggregation performed by gateways is tailored to meet the requirements of specific IoT networks and applications, which also limits how this data can be used by other applications. Thus, such gateways become inefficient in heterogeneous IoT networks, endowed with protocols, applications, and devices with different characteristics (WISSINGH; D'ACUNTO; TRICHIAS, 2017; SHEN et al., 2017).

Taking into account the challenges mentioned above, there is a real need for a network protocol stack capable of natively supporting adaptive and general-purpose data aggregation. Such a stack will allow IoT applications to request and obtain data according to their needs, in an adaptive way (WISSINGH; D'ACUNTO; TRICHIAS, 2017).

2.4 DATA AGGREGATION

Within the IoT network scenario, described earlier, data aggregation has become an important feature due to its ability to conserve the limited battery of IoT devices (ALGHAMDI et al., 2016; ALKHAMISI; NAZMUDEEN; BUHARI, 2016), reduce the volume of data transmitted over the network (BAKTIR; OZGOVDE; ERSOY, 2017; ALGHAMDI et al., 2016) and improve the WSN-IoT network's throughput (ALGHAMDI et al., 2016; ALKHAMISI; NAZMUDEEN; BUHARI, 2016).

However, traditionally, data aggregation techniques are performed by the base station of wireless networks. Thus, all data generated by the IoT devices are routed from device to device towards the base station, where the data aggregate is calculated. This makes the sensors use a good part of their battery transmitting data over the network (ALGHAMDI et al., 2016).

In addition to the problems mentioned above, the amount of payload sent by IoT devices is small when compared to the size of packet headers. Consequently, the communication protocols used in IoT networks have a major impact on communication efficiency and network overhead (YOKOTANI et al., 2017). Also, small payload packets experience varying delays, which creates network jitter (AKYUREK; ROSING, 2016).

To solve the aforementioned problems, recent research on data aggregation in IoT networks has managed to develop conservative energy approaches. Such techniques propose that the data produced should be aggregated in each IoT device and the resulting aggregates forwarded to the base station (ALGHAMDI et al., 2016).

Another complementary approach is the aggregation of packets within the networks. Such a strategy is very efficient since multiple data from different applications or even different nodes are aggregated in a single packet to maximize efficiency. When transmitting a single packet instead of a set of packets, one can save energy and decrease the overall end-to-end average delay (AKYUREK; ROSING, 2016).

2.4.1 Packet aggregation within the network

The aggregation of packets within the network (in-network aggregation) is the process of collecting and combining data payloads as packets are routed within a multi-hoc network (SRUTHI; GEETHAKUMARI, 2016). This aggregation strategy combines the data from several packets into a single one that is transmitted to the next network node (AKYUREK; ROSING, 2018). Such an approach eliminates recurring lower layer headers, such as IP and 802.11, and it also reduces the number of packets transmitted. Consequently, this strategy reduces the energy consumption of IoT devices, increases network efficiency, and indirectly reduces latency and interference (contention and collision) in shared media (AKYUREK; ROSING, 2018).

The effects of packet aggregation within a network depend directly on when the transmission of the aggregated packet will occur. If the aggregator node waits for the packet reception for a long period, more packets can be combined. However, the packet delivery latency will increase during this process, which may be undesirable for the IoT

application. On the other hand, if the aggregator waits for a short period, fewer packets will be aggregated, and the advantages of packet aggregation techniques will not be that significant. Thus, the optimum point for packet transmission depends on several factors, ranging from the communication flow of individual applications to the current state of the network (AKYUREK; ROSING, 2018).

2.5 RELATED WORKS

Several studies in the literature analyze packet aggregation within the network (AKYUREK; ROSING, 2018; AKYUREK; ROSING, 2016). In one of these studies, Karlsson, Kassler e Brunstrom (2009) showed that packet aggregation increases the efficiency of TCP flows and reduces end-to-end delays. Similar advantages were also observed in the study conducted by Kim et al. (2006), where the authors show that the aggregation of data in a Voice over Internet Protocol (VoIP) system decreases jitter and increases the number of simultaneous calls on the network.

According to Akyurek e Rosing (2018), one of the most popular and oldest data aggregation solutions is LEECH, proposed by Heinzelman, Chandrakasan e Balakrishnan (2002). This solution proposes that the Media Access Control (MAC) protocol should use temporal division techniques to aggregate data through clusters. Heinzelman, Chandrakasan e Balakrishnan (2002) provided simulations that show the trade-off between the number of clusters and the performance achieved.

Yokotani et al. (2017) proposed a new data aggregation strategy at the application level and compared it with traditional communication strategies. In the traditional strategy, IoT devices send a packet for each data obtained by the IoT sensors. However, Yokotani et al. (2017) proposes that the data of several IoT sensors should be combined in a single ZigBee packet. To do so it segments data into blocks called data blocks, each one occupying a byte of the ZigBee packet payload. Also, this proposal suggests that IoT devices should send packets only if there is a change in the data value over time. In other words, they do not sent the same piece of data twice. As each data block has a size of a byte and the ZigBee protocol has a total size of 133 bytes, it is possible to send up to

67 data blocks using this new proposal. The results obtained with this approach showed that the aggregation of data blocks offers a better result than the traditional approach regarding the total network traffic.

Another approach similar to Yokotani et al. (2017) was proposed by Zechinelli-Martini, Bucciol e Vargas-Solar (2011), also addressing the aggregation problem from the point of view of IoT devices. In this context, the multiple measurements of the IoT devices are aggregated in a single packet to save energy. For this, Zechinelli-Martini, Bucciol e Vargas-Solar (2011) considered that IoT devices will produce only a single type of data stream with known characteristics. Thus, the proposed solution is optimized for a specific platform, endowed with the restrictions associated with the data flow.

Hu, Cao e May (2005) proposed an aggregation based on timed adjustments, which uses network congestion heuristics to determine the exact moment to send a data aggregate. For this, the proposed solution achieved high energy efficiency by adjusting the sending time of the data aggregate, according to fluctuations in network traffic congestion. Xu, Zeng e Qu (2006) proposed a solution similar to the previous work, in which the transmission times are adjusted according to the number of network hops. This study also aims to improve the energy consumption of the network. However, both proposals are based on software, which undermines the scalability of these solutions for large networks.

The most recent research has been trying to integrate the different aggregation techniques mentioned above into a generic IoT data aggregation architecture, such as the one described by Karim e Al-kahtani (2016) and Shen et al. (2017). In the PDDA architecture proposed by Karim e Al-kahtani (2016), data aggregation occurs hierarchically, on three levels: on sensors (level 1), on the base station (level 2), and on the gateway fog (level 3), as illustrated in Figure 2.3. The results showed that PDDA provided better performance than the traditional strategies (tree-based and cluster-based).

The proposal of Shen et al. (2017) presents a general-purpose IoT architecture that facilitates interoperability between different IoT devices, endowed with different communication technologies and computational resources. In this proposal, IoT devices connect to the front-end ports of the switches, while the back-end ports are used to communicate

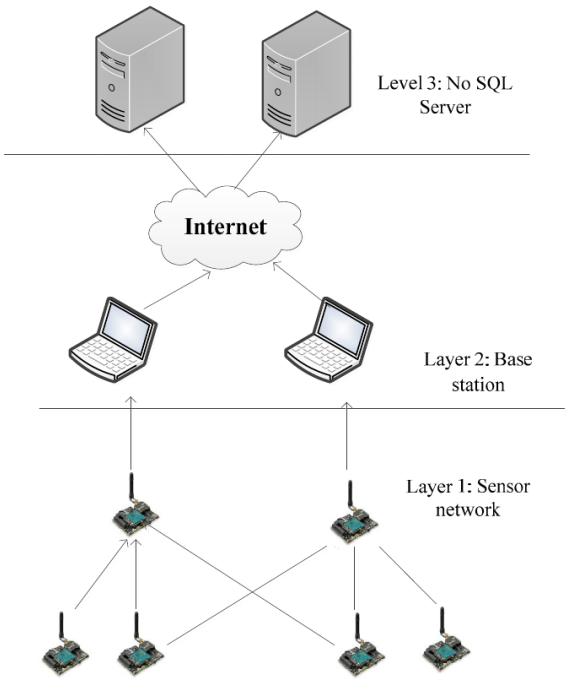


Figure 2.3: PDDA IoT Data Aggregation Architecture (KARIM; AL-KAHTANI, 2016)

with the rest of the network (gateways and aggregation servers).

However, most of the solutions proposed above assume that the network environment is homogeneous and the data flow comes from a single application. That is, such proposals are based on the premises: (i) IoT devices have similar computational capacities and communication technology, (ii) the temporal behavior of the data flow to be transmitted on the network and its type of load (linear, Poisson, etc.) are known a priori. Besides, most of the aforementioned studies aim to optimize only a single performance metric, with the main focus of most of these studies being energy consumption. However, IoT devices and applications may have different characteristics, requirements, and optimization metrics. Associated with these facts, 5G cellular networks are increasingly using the Machine-to-Machine (M2M) communication model, which requires more flexible aggregation solutions (AKYUREK; ROSING, 2018).

Given the above, we propose the IoTP protocol that allows the implementation of a generic network architecture for aggregating hierarchical data. IoTP provides support for data aggregation before, during, and after data transmission from IoT devices, similarly to the proposals of Karim e Al-kahtani (2016) and Shen et al. (2017). However, unlike

Karim e Al-kahtani (2016) and Shen et al. (2017) proposals, the IoTP protocol can be implemented directly in the network switches' hardware, in conjunction with data aggregation strategies that adapt to the needs of IoT and network applications. Thus, our proposal is capable of providing: (i) greater energy efficiency for the network and IoT devices, (ii) greater efficiency in data aggregation, (iii) less use of network links, without reducing the total payload transmitted, and (iv) less use of the computational resources of the switches.

2.6 CHAPTER'S SUMMARY

In this chapter, SDN, IoT and data aggregation are discussed alongside several works related to this dissertation. In addition to the works presented, there are still others in the literature. It can be noted that some studies have already addressed the issue of data aggregation in IoT WSN networks. However, the works proposed so far do not consider the network's current status to improve data aggregation efficiency, while other studies have investigated the data aggregation from the IoT device perspective, aiming at a single optimization metric, mainly IoT device energy autonomy. Thus, the present work is a pioneer in considering, at the same time, the IoT device constraints and the network current status effect upon the overall data aggregation efficiency and network congestion to assist data aggregation algorithms in achieving higher efficiency and/or low latency. The present work is also a pioneer in regards to the embedding data aggregation algorithms within the programmable data plane of hardware-based switches to achieve higher aggregation efficiency and packet forwarding performance.

In the next chapter, the methodology of this work is presented, important IoT-related definitions and concepts are established and the proposal's operation is detailed.

Chapter

3

INTERNET OF THINGS PROTOCOL

Considering the features of the proposals discussed in the previous section, we proposed the IoTP protocol herein. The architecture proposed aims to mitigate congestion and improve the efficiency of IoT networks by performing aggregation of IoT data as it traverses the network switches. Such aggregation improves network efficiency[1] as it reduces the overhead caused by the repeated packet headers of network protocols. To improve the interoperability and make the protocol more generic, we designed IoTP to be independent of routing protocols and link technologies. Due to these features, IoTP can coexist with any other IoT communication technologies, such as BLE and ZigBee, due to the programmable data plane low-level header processing[2]. So, our design has no impact in the data transmission of other IoT technologies. For those devices that use IoTP, the aggregation method described herein will be carried out, harnessing the IoTP features.

As we discuss in the next sections, IoTP helps to achieve higher performance since it adopts a fixed header. The protocol also has been implemented in the link layer (L2) of the P4 switches. Such design choices allow IoTP's execution to take place within the

[1]In this work, network efficiency indicates the communication overhead associated with a protocol, being defined as the ratio of total payload by total data transmitted. Section 5.1.1 provides further details about this network metric.

[2]The coexistence is possible due to the port-by-port configuration of the switch's packet parser.

network data plane of intermediate network nodes, such as switches. In doing so, IoTP increases interoperability between heterogeneous IoT devices and allows the development of more efficient aggregation strategies. The next section formalizes IoTP's essential concepts and definitions for the protocol's design.

3.1 CONCEPTS AND DEFINITIONS

- **IoT Application**: it is an application designed to solve a problem that requires IoT devices to exchange data with the network infrastructure. Ex: Detection of forest fires, control of industrial and metallurgical processes such as iron foundation, etc. In IoTP, an IoT Application is identified by the header field "Service ID", further described in Section 3.4.

- **Multi-level Aggregation Architecture**: it is a network, in which each node of the network can execute a certain aggregation strategy as the packets traverse the network nodes. These networks typically have a hierarchical topology to allow more nodes to aggregate data, but it is not a strict requirement. The IoTP protocol allows the deployment of such networks, as each node can execute a different aggregation algorithm alongside the IoTP protocol, according to its computational resources.

- **IoT Device**: it is an embedded system, with limited computational resources (memory and CPU), powered by batteries, and equipped with sensors that collect and transmit data periodically to a machine or set of machines on the network. Such devices are responsible for capturing and transmitting IoT data to the IoTP gateway, using the network's protocol.

- **Aggregation Strategy**: it is an algorithm that, given a dataset as input, produces a smaller dataset as output, called "aggregated data" or "data aggregate". Different algorithms can be used throughout IoTP network, according to the available computational resources of each node.

3.2 SDN CONTROLLER ROLE

In an IoTP network, the Software-Defined Networks (SDN) controller is responsible for: (i) establishing the packet routes, and (ii) choosing and controlling the data aggregation algorithm that will be executed by the network. To reach this goal, the SDN controller take into account the IoT application requirements and the current network status. The IoTP network stack, header and operation is described in further details in next sections.

3.3 NETWORK STACK

Unlike traditional network stacks, such as TCP/IP, in which there are 5 to 7 layers in the network stack, the IoTP network is based on only 4 layers, as depicted in Figure 3.1. In this way, IoTP brings with it the following benefits, given the scenario of IoT networks: i) the IoTP simple network stack requires less computational resources to implement than a more complex stack, making IoTP useful for a wide range of heterogeneous IoT devices, ii) since IoTP is a protocol that provides support for multi-level data aggregation techniques, taking into account the current network status, it can improve the efficiency of data transmission over the network, making it possible to send the same amount of payload with less usage of the network links, and iii) IoTP is able to control the delay involved in data aggregation and, in doing so, it can prioritize IoT data retrieval.

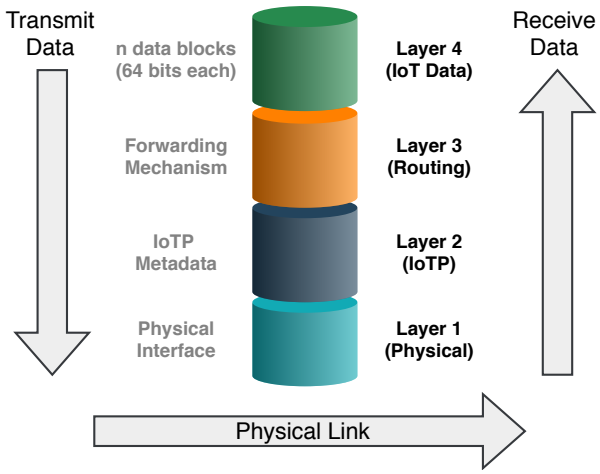


Figure 3.1: IoTP Network Stack (MADUREIRA; ARAúJO; SAMPAIO, 2020)

In the IoTP architecture, the physical layer (i.e., Layer 1) is adaptable, which means that IoTP switches can operate with different types of communication technologies, as required by their IoT scenario. Therefore, the IoTP protocol can be used by a wide range of IoT devices. To accomplish this behaviour, the network switches, in which IoTP will be executed, must provide the necessary hardware support to such communication technologies.

The link layer (i.e., Layer 2) is where the IoTP header is located. The IoTP header contains metadata that can be used by the aggregation algorithms to increase the efficiency of the data aggregation strategies implemented throughout the network. This metadata is constructed by the IoTP switches as the packets pass through them. With such information, the IoTP protocol stores part of this metadata in the header of each packet, in order to transmit this information to the rest of the network. In doing so, the IoTP protocol can provide information about the IoT data contained in each packet, and can also be used to infer the status of the network. The IoTP header is described in more detail in the Subsection 3.4.

The network layer (i.e., Layer 3) is comprised by a header of a given protocol, which indicates to the network how to route the packet. Such header is called "forwarding mechanism" in the IoTP architecture and it must be present in every single IoTP packet so that they can be forwarded throughout the network. IoTP packets can have different "forwarding mechanisms", which allows each packet to take a different route from each other. In this way, the protocol favors the implementation of traffic engineering techniques on a packet-by-packet basis. Examples of forwarding mechanisms are the Ethernet header, IPv4, IPv6, Bluetooth Low Energy, among others.

The application layer (i.e., Layer 4) is comprised by the IoT data, in the form of a set of n data blocks, as described in Subsection 3.4.1. Each data block store up to 64 bits of information, thus allowing the aggregation algorithms to know the structure of the data transmitted in the IoTP packets payload. Therefore, it becomes possible to improve the performance and efficiency of the data aggregation performed throughout the IoTP multilevel architecture.

3.4 PROTOCOL HEADER

To make L2 processing feasible, the protocol is based on packet processing consisting of a fixed header made up of five fields and a stack of data blocks, as illustrated in Figure 3.2. Through this header, IoTP becomes generic and independent of routing protocols and communication technologies. Consequently, interoperability between heterogeneous IoT devices becomes more feasible.

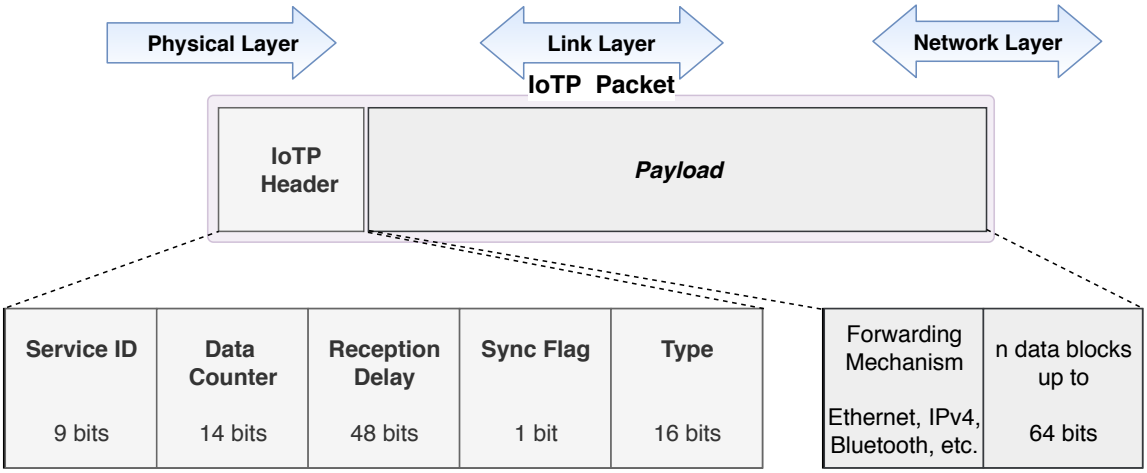


Figure 3.2: IoTP Packet Structure (MADUREIRA; ARAúJO; SAMPAIO, 2020)

The header of the IoTP protocol described in Figure 3.2 is formed by the following fields:

- **Service ID**: This is a 9 bits field that identifies a set of similar data belonging to the same IoT application. With this approach, network switches can group incoming data into a single packet in case of the total size of the data sent is less than the network MTU.
- **Data Counter**: IoTP enables IoT devices to send multiple blocks of data in a single packet, as long as the limit set by the network's MTU is respected. To this aim, this field identifies the number of data blocks a packet carries. This field is 14 bits in size. Therefore, IoTP is capable of carrying up to 16383 blocks of data per packet, each containing 8 bytes, totaling 128 KB of data.
- **Reception Delay**: This 48-bits field indicates the time taken from the data cap-

ture by the IoT sensor to the reception of this information by the IoTP gateway. Thus, the IoTP switch stores IoT data internally and, when the condition to send the packet is met, the IoTP switch updates the packet delay and sends the aggregated data. Through this approach, the IoTP gateway is able to estimate the time elapsed from capturing the first data to receiving this information. Figure 3.5 and Algorithm 5 provide further details about the "Reception Delay" field.

- **Sync Flag**: When an IoTP packet with Sync Flag enabled is received by the switch, it sends all the internally stored data that is associated with the service ID of the Sync Flag packet to the IoTP gateway. By periodically sending packets with Sync Flag, the gateway is able to control the trade-off between the delay in receiving data and the efficiency of the data aggregation strategies.

- **Type**: This 16-bits field informs which forwarding mechanism should be used to forward IoTP packets. Therefore, IoTP can be used on any network, regardless of the underlying communication and routing technologies. For instance, the forwarding mechanism can be another protocol header such as Ethernet, Bluetooth 4.0, 5G, IPv4, and IPv6.

3.4.1 Packet payload

Each IoTP packet, described in Figure 3.2, contains a finite set of up to n blocks of data. Each block carries the data collected by the IoT devices, and can contain up to 64 bits (i.e., 8 bytes) of information. Thus, all elements that compose an IoTP network (such as switches, gateways and IoT devices) are aware of the amount and format of the data that each packet carries, which facilitates the use of aggregation strategies.

3.5 EXAMPLE OF DATA AGGREGATION IN IOT SCENARIOS

Figure 3.3 illustrates an example of IoTP-based data aggregation. It describes how a specific strategy could be supported by the protocol's functions. In this example, Service

ID 1 has a Trigger Counter[3] equal to 6. Service ID 2 has a Trigger Counter equal to 3. IoT devices H_1, H_3 and H_2 send the respective packets A, B and C to the IoTP switch S_1. As S_1 receives these packets, the data contained in them is aggregated by the switch, creating the new D and E packets. These new packets are then forwarded to the IoTP gateway G_1.

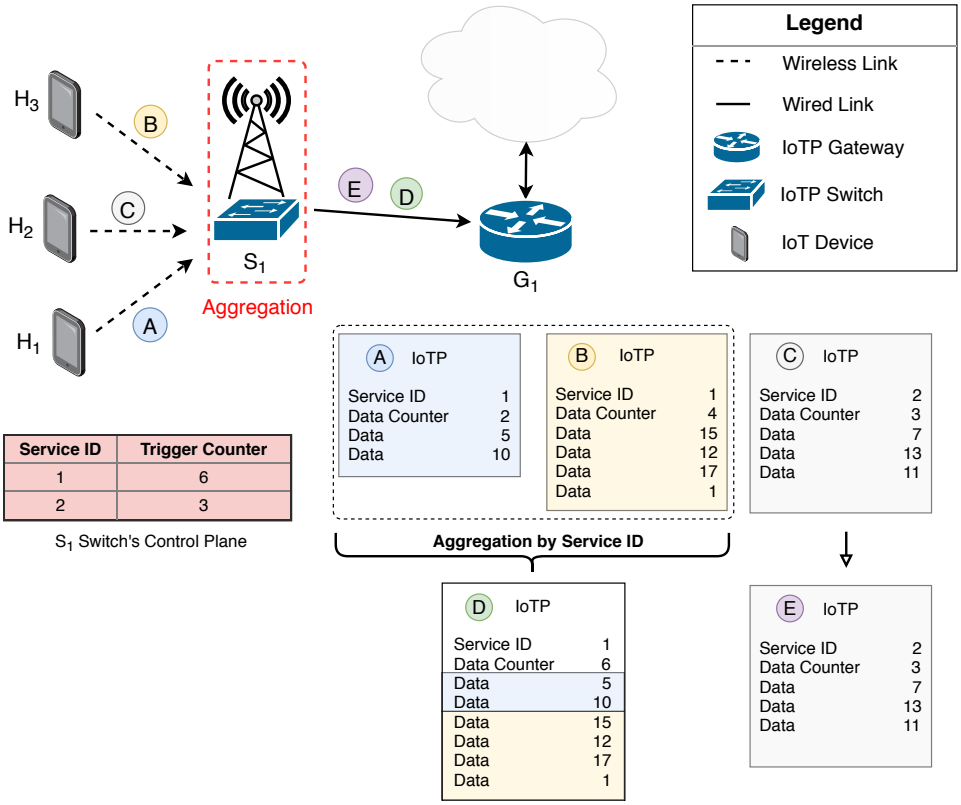


Figure 3.3: Example of aggregation supported by an IoTP-based P4 Network (MADUREIRA; ARAúJO; SAMPAIO, 2020)

Packet D, in this example, is composed of the data contained in packets A and B, since both have the same Service ID 1. This packet is created by the switch and sent to IoTP gateway. This happens because the switch received an amount of data equal to or greater than the Trigger Counter of the Service ID 1. That is, the switch received at least 6 IoTP data blocks with Service ID 1. Packet E is composed only of data contained in packet C, since no other packet with Service ID 2 was received by switch. The switch

[3]The trigger counter is the minimum number of IoTP data blocks that each node has to store prior to performing data aggregation and transmitting the data aggregate. Section 3.6 provides further details about the trigger counter.

creates and sends this packet E to the IoTP gateway, since it has received an amount of data equal to or greater than the Trigger Counter of the Service ID 2 (i.e., the switch received 3 blocks of IoTP data with Service ID 2).

Bearing in mind that IoTP headers provide metadata (i.e., information about the transmitted data), IoTP can also be used to implement other aggregation algorithms. Such strategies can be implemented directly in the hardware of the network switches, being executed by the data plane. That is, the aggregation algorithms can be processed at line rate speed and within the switching devices. Therefore, we remark that the example illustrated in Figure 3.3 is just one example of aggregation strategies that can be implemented using IoTP.

IoTP can also be associated with other aggregation solutions that execute inside IoT devices and within IoTP gateway, allowing sophisticated aggregation algorithms and caching strategies to be employed to improve overall network performance and data storage compression. Thus, the IoTP network architecture enables the implementation of multilevel data aggregation strategies, as depicted in Figure 3.4, such as the ones described by (KARIM; AL-KAHTANI, 2016; SHEN et al., 2017).In such architectures, multilevel data aggregation occurs as the data travels through the various elements of the architecture, allowing different distributed aggregation algorithms to be executed according to the available computational resources. In IoTP architecture, we implement a similar multilevel data aggregation, but we also take into account the current network status and information, like MTU, bandwidth and delays, to perform the data aggregation. That feature allows IoTP to improve data aggregation efficiency, mitigate network congestion, and lower network link usage when compared to the other architectures.

3.6 PROTOCOL OPERATION

In order to assess the effectiveness of IoTP in supporting aggregation techniques, we implemented a version derived from the aggregation algorithm Accretion (KIM et al., 2006). In our implementation of the algorithm, IoTP devices receive various packets from other devices on the network and aggregate their data according to the service ID

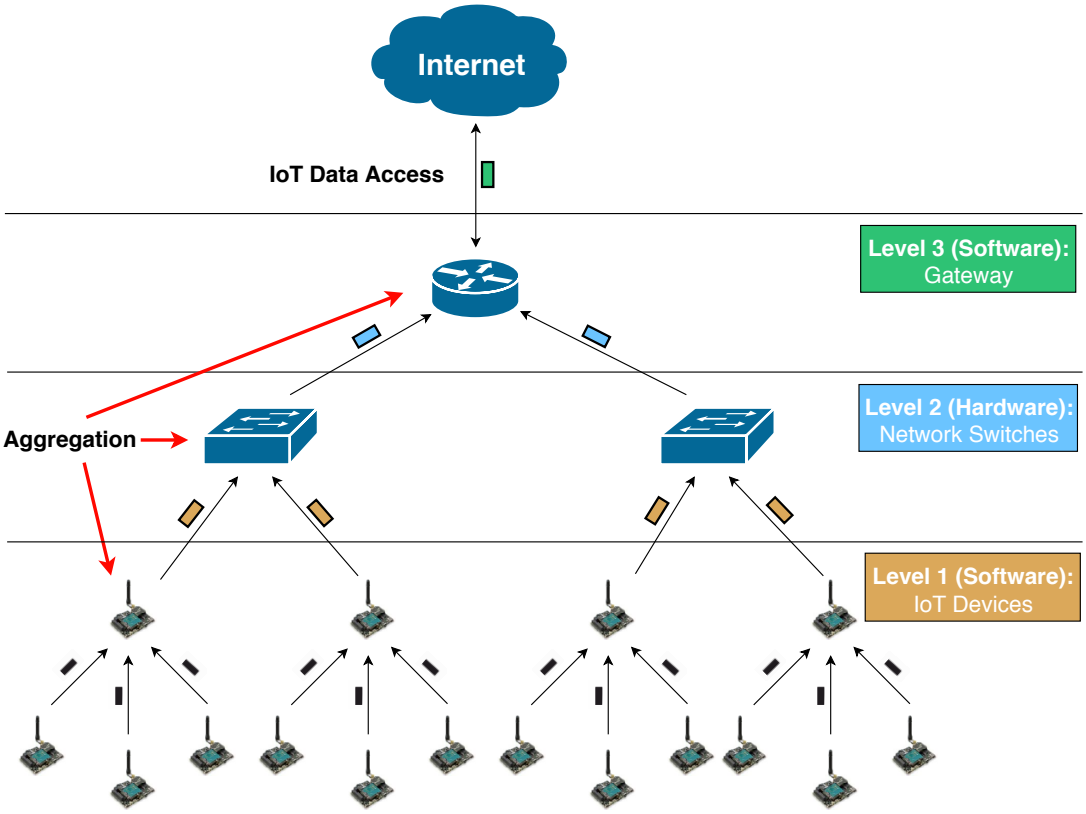


Figure 3.4: IoTP Network Architecture. Adapted from Karim e Al-kahtani (2016)

of the packets. At the end of the aggregation process, the IoTP device temporarily stores this information in its internal memory (see Algorithm 1) until this information is sent to the IoTP gateway (see Algorithm 2)[4].

As described by the Algorithm 1, the aggregation strategy implemented within the switch stores the data sent by the IoT devices, as well as the information regarding the delays that the data suffered from the moment it was captured to time it has reached the IoTP switch. In addition to that, the IoTP switch stores delays caused by the aggregation strategy that executes alongside the IoTP protocol, within the IoTP switch. To this end, the switches store the exact moment when the first data for a given service ID is received[5]. This information is then used to calculate how long the data remained

[4]Most data aggregation strategies, IoTP included, temporarily store data in order to improve efficiency. However, that approach creates a trade-off between efficiency and reliability. Noneless, IoTP manages this trade-off through Sync Flags, further described in Section 3.6. Still, IoT devices have to retransmit any lost data, as needed by the IoT application.

[5]To keep track of time inside a P4 switch, we used an internal P4 timer. This timer keeps track of how much time (in microseconds) has elapsed since the IoTP switch has been turned on.

within the switch. Based on this information, it calculates the total delay induced by the chosen aggregation strategy. Thus, the total reception IoTP delay, described in the IoTP's delay field, consists of the sum of the IoT device delay and the IoTP switch delay, as depicted in Figure 3.5. So, IoTP does not require real-time synchronization to calculate the reception delay. In Section 4.3, Algorithms 4 and 5 provide further details about the delay storage and calculation.

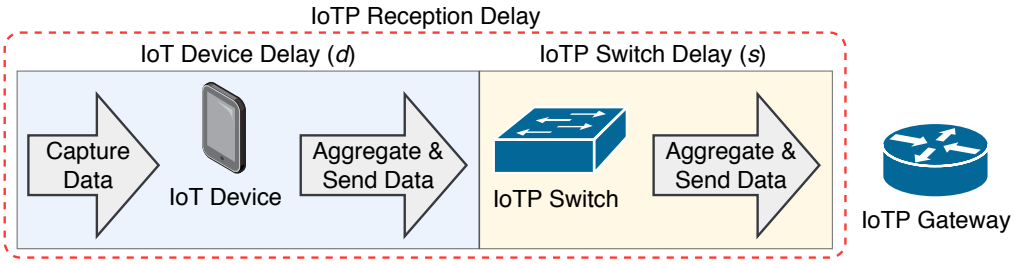


Figure 3.5: IoTP Reception Delay (MADUREIRA; ARAúJO; SAMPAIO, 2020)

Algorithm 1: Data Storage.

```
1 Function AggregateReceivedData(pkt):
2     if pkt.dataCounter > 0 then
3         StoreDelay(pkt.serviceID, pkt.delay)
4         foreach data in pkt.dataList do
5             StoreData(pkt.serviceID, data)
6         end
7     end
8 end
```

After the storage phase ends, the conditions for sending the data are verified (see Algorithm 2). The switch calculates the cumulative data aggregation delay, adds this information to the delays reported by the IoT devices, and sends the stored data. The format in which the stored data is sent depends on the aggregation strategy implemented in the IoTP's switches. If the switch implements, for example, the moving-average strategy, a set of n stored data blocks is compressed into a single data block. If the stacking aggregation strategy is used, as illustrated in Figure 3.3, the entire set of n data is sent in the same packet. Thus, the IoTP protocol is responsible for controlling the delays related to the implemented aggregation strategy and defining the moment when the stored IoT

data will be sent by the switch.

Algorithm 2: Aggregation and Data Send.

```
1  Function SendAggregatedData(pkt):
2      id = pkt.id
3      sf = pkt.syncFlag
4      len = LenStoredData(id)
5      if len >= MinData(id) or sf == 1 then
6          pkt.delay = StoredDelays(id)
7          pkt.delay += TimeDataInSwitch(id)
8          pkt.dataCounter = len
9          foreach data in StoredData(id) do
10             pkt.dataList.push(data)
11         end
12         SendPacket(pkt)
13     end
14 end
```

The aggregation strategy implemented within the IoTP's switches considers two conditions for sending the stored data: (i) minimum amount of data blocks reached or (ii) an IoTP packet with Sync Flag enabled is received by the switch.

In the first condition, each service ID of the IoTP protocol is associated with a minimum amount of data, called "Trigger Counter", which indicates to the aggregation strategy when to send the data stored within the switch. When the minimum amount of storage is reached, the switch applies the aggregation strategy to the locally stored data and sends the aggregated data to the IoTP gateway. The minimum number of data blocks for each service ID is defined by the SDN controller, according to the characteristics of the data received and the requirements of the IoT applications.

The second condition enables the IoTP to control delays related to the aggregation strategy. However, unlike the End-to-End, Hop-by-Hop and Accretion algorithms (KIM et al., 2006), IoTP does not use timers to perform such control. Instead, the delays are controlled by sending packets with Sync Flag enabled (i.e., pkt.syncFlag == 1) to the IoTP switch (see Algorithm 3). As such packets are received, the switch retrieves all stored data associated with the service ID. Then, the aggregation strategy is applied over the data and the resulting data aggregate is sent to the IoTP gateway. So, the IoTP

gateway can control the frequency of data reception by sending IoTP packets with Sync Flag periodically. This approach, therefore, facilitates the controlling of the trade-off between reception delay and efficiency of the aggregation strategy.

Algorithm 3: SyncFlag Packet Sending

```
1 Function SendSyncFlag(pkt):
2     for id in serviceIDList do
3         receptionTime = TimeLastReception(id)
4         if ActualTime - receptionTime >= ReceptionTimeout(id) then
5             pkt.id = id
6             pkt.delay = 0
7             pkt.dataCounter = 0
8             pkt.syncFlag = 1
9             SendPacket(pkt)
```

3.7 CHAPTER'S SUMMARY

In this chapter, IoTP was presented, a new solution to support the in-network data aggregation in IoT WSN scenarios. The proposed solution benefits from the characteristics of the IoT devices, programmable data plane hardware-based switches, and wireless networks, to support the deployment of network-aware data aggregation algorithms within the data plane of the network nodes. In this way, in-network data aggregation algorithms can consider both the network current status and IoT device constraints to improve network congestion, latency, and/or data aggregation, while executing at line rate. In the next chapter, the implementation details of the proposal will be presented and important aspects of data plane programming will be discussed.

Chapter

4

P4 LANGUAGE IOTP IMPLEMENTATION

4.1 PROGRAMMABLE DATA PLANE (PDP)

The network data plane must be programmed to understand and execute the IoTP protocol proposed in this study. To this end, we adopted the P4 technology, which is a high-level language for data plane programming[1] based on the SDN paradigm that allows the network engineer to program the data plane of hardware switches. Thus, using P4, one can program the hardware similarly to Verilog and VHDL languages, without having to understand complex electrical engineering design patterns, commonly required by VHDL and Verilog. By programming the data plane, network designers can describe the behaviors and characteristics of the network with more granularity. Thus, the IoTP-based P4 network herein presented is based on switches programmed to interpret the IoTP protocol and carry out actions on packets traversing their communication interfaces. To accomplish this goal, we also programmed an SDN controller to populate the

[1]Data plane programming is different from normal computer programming languages, as it is similar to hardware description languages (e.g., P4, Verilog and VHDL). With hardware description languages one can construct a CPU (code is compiled to logic gates inside the programmable hardware), but with computer programming languages, one can only construct software, that runs inside CPUs. Generally, software implementations are slower than the hardware-based ones (SEZER et al., 2013), as they are executed over the available hardware.

match-action tables of the switches to inform which actions should be taken and when they should be performed.

In this study, we used the P4 Very Simple Switch (VSS) architecture to implement IoTP, as depicted in Figure 4.1. This implementation considered the language restrictions, presented in P4's official specification (P4 LANGUAGE CONSORTIUM, 2019b). Among these restrictions, the language does not provide loop structures and has no complex data structures, such as matrices. IoTP itself does not require such programming resources, to be implemented within a P4 switch. Conversely, the aggregation strategy that runs alongside the IoTP protocol, within the P4 switch's data plane, does require such programming resources. Thus, to circumvent such issues, programming techniques such as loop unroll and matrix linearization were required to implement the aggregation strategy described by Algorithms 1 and 2.

In the next sections, we further describe how each component of the VSS architecture was implemented, providing details about the IoTP protocol and the data aggregation algorithm design. Finally, we also discuss P4 language restrictions, how we were able to circumvent them, and their implications upon IoTP and data aggregation.

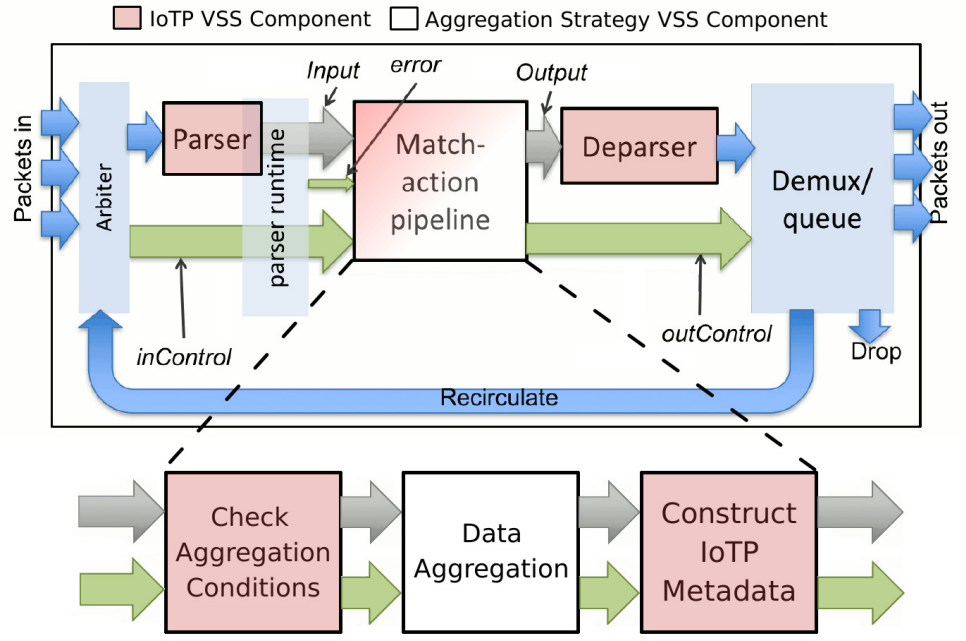


Figure 4.1: Key IoTP VSS components (MADUREIRA; ARAúJO; SAMPAIO, 2020)

4.2 PARSER AND DEPARSER

To process any packet in P4 we are required to build a packet parser for each protocol we want to support inside the switch. A packet parser is a state machine responsible for the interpretation of the serialized data received from the switch's Network Interface Controllers (NICs). Once this step is completed, we have access to each packet header field, allowing the execution of protocols in a structured manner.

As P4 is a language for protocol header processing, we had to consider the packet payload as a set of P4 headers to allow data aggregation to take place in the switch's data plane. In this way, we can access each IoTP data block within P4 control blocks. Furthermore, IoTP's parser has to be configured in a way that it provides interoperability between different IoT link-layer technologies. With these goals in mind, we designed an IoTP packet parser according to the state machine described in Figure 4.2. First, the packet parser interprets the IoTP fixed header fields. Then, we verify which forwarding mechanism the IoTP packet should use, based on the IoTP *Type* field. In this particular implementation, we developed the forwarding mechanism for the Ethernet link-layer technology. Once the forwarding mechanism is identified, our packet parser iterates over the packet's payload to identify each IoTP data block.

To construct the IoTP deparser, we used the same approach of the parser, with the difference that the deparser traverses the same steps in reverse order. Hence, IoTP deserialization (deparser) has a similar performance to the serialization (parser).

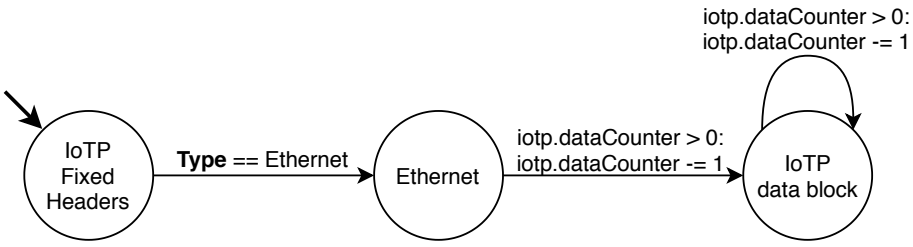


Figure 4.2: IoTP Packet Parser

4.3 MATCH-ACTION PIPELINE

In this section, we present relevant implementation adaptations that were required to deploy a P4-based IoTP switch with data aggregation support. The match-action pipeline in a P4-based switch is responsible for the programming logic of network protocols, being the only P4 control block that allows conditional code execution (if conditions). In IoTP, this component is implemented within the P4's VSS ingress control block and is responsible for delay calculation and data aggregation, following the steps: i) delay storage, ii) data storage and aggregation, and iii) delay calculation.

4.3.1 Delay Storage

To implement the aggregation delay calculation within P4 switches, first, we need to store the accumulated delay, associated with the data aggregation that has taken place within previous network nodes (e.g. IoT devices, other IoTP switches, or IoTP gateway). Then, we calculate the processing time of the IoTP switch, and, finally, we sum the accumulated delay with the processing time. In this section, we detail the first step related to accumulated delay storage.

We used P4 registers[2] to store the accumulated delays. As each IoTP packet transports a set of n IoT data blocks, we considered that these data blocks contained within the packet are the result of a prior aggregation that has happened in a previous network node, either an IoTP switch, IoT device, or IoTP gateway. Therefore, each IoTP packet has a 48-bit "Reception Delay" field that represents the elapsed time since the IoT data capture, measured in microseconds (μs). As described by the Algorithm 4, when a packet arrives at the switch, the following actions take place: i) read the IoTP service ID to identify the IoT application associated with the packet payload, ii) if no data was stored within the switch, store the current time[3] in the start time registers to calculate the switch's

[2]Each P4 register represents an array that can store 32-bit words in each vector position.

[3]To obtain the current P4 switch time, we used the *standard_metadata.ingress_global_timestamp* internal P4 timer, that represents the switch's uptime in microseconds (μs). *standard_metadata.ingress_global_timestamp* is initialized as 0 when the switch is powered on.

total processing time, and iii) find where the accumulated delay should be stored inside the two P4 registers. We used two P4 registers to store the 48-bit accumulated delay and two registers for the 48-bit current switch's time, in a total of four P4 registers.

Algorithm 4: Start Time and Accumulated Delay Storage.

```
1  Function StoreDelay(pkt.id, pkt.delay):
2      if LenStoredData(pkt.id) == 0 then
3          currentTime = standard_metadata.ingress_global_timestamp
4          startTimeLowRegister[pkt.id] = currentTime[22:0]
5          startTimeHighRegister[pkt.id] = currentTime[47:23]
6      end
7                                  ▷ concatenate high and low bits
8      accDelay = accDelayHighRegister[pkt.id] ++ accDelayLowRegister[pkt.id]
9      accDelay += pkt.delay
10     accDelayLowRegister[pkt.id] = accDelay[22:0]
11     accDelayHighRegister[pkt.id] = accDelay[47:23]
12 end
```

Each 48-bit delay information (accumulated delay or current switch time) could be stored within a P4 register, but such an approach would require matrix linearization, as these delay information are associated with an IoTP service ID. Therefore, if such an approach were to be used, we would have to implement a hash function to decide which register position the delay should be stored, which harms packet forwarding performance and incurs additional overhead.

4.3.2 Data Storage and Aggregation

The Algorithm 1 describes the data and delay storage, while Algorithm 2 describes the data aggregation strategy and reception delay calculation. The data storage and aggregation of those algorithms require loop structures to be implemented, but the P4 specification lacks this feature. Therefore, we specified the maximum amount of data to aggregate and store in the IoTP switches (M) as 50 data blocks[4]. As a result, the number of loop executions became finite and known at compile-time, thus allowing us to

[4]The 50 data block limit was set due to P4 emulation environment instability caused by BMv2 memory constraints. Further details about this issue are described in Section 5.3.

use the loop unroll technique. Such a technique not only provides loop-like features to P4 programs but also optimizes the pipeline execution speed. However, such a technique implies duplicated P4 code.

To store and retrieve the IoTP data inside the P4 switches, it was necessary to associate the data with their respective service IDs. For that, we would need to use a matrix data structure. However, the P4 language has only vector data structures (i.e., P4 registers). Thus, we had to use the matrix linearization technique to store the data from each IoTP packet inside the P4 registers, according to the service ID of each packet. For this, we used the hash function described in Equation 4.1.

$$H(id) = lenDataStored(id) + M * id \tag{4.1}$$

where $lenDataStored(id)$ is a function that returns the amount of data stored on the switch associated with the service id, M is the maximum number of storable data blocks within the switch, and id is the service ID of the IoTP packet.

4.3.3 Delay Calculation

In Subsection 4.3.1 we described the storage of accumulated delay and switch start time, while in Subsection 4.3.2 we presented implementation details concerning data storage and aggregation. In this subsection, we focus on the delay calculation, the final step of IoTP's match-action pipeline. In this P4 control block, to calculate the switch's aggregation elapsed time, the current switch's time is subtracted from the $startTime$, which represents the moment the first IoTP data block is received by the switch. Then, we sum the elapsed time with the accumulated aggregation delay, as depicted in Algorithm 5.

4.4 CHAPTER'S SUMMARY

In this chapter, IoTP's implementation details were presented and important aspects related to data plane programming were discussed. IoTP was implemented through the Programming Protocol-Independent Packet Processors (P4) data plane programming

Algorithm 5: Delay Calculation.

```
1 Function DelayCalculation(pkt.id):
2     currentTime = standard_metadata.ingress_global_timestamp
3     startTime[22:0] = startTimeLowRegister[pkt.id]
4     startTime[47:23] = startTimeHighRegister[pkt.id]
5     elapsedTime = currentTime - startTime
6     accDelay[22:0] = accDelayLowRegister[pkt.id]
7     accDelay[47:23] = accDelayHighRegister[pkt.id]
8     pkt.delay = elapsedTime + accDelay
9 end
```

language. The P4 programming language allows the design of custom packet parsers, as well as custom match-action tables, which allow complex protocols, such as IoTP, to be deployed easily within the data plane of hardware switches. P4 also allows IoTP to be independent of data aggregation algorithms, as it can be implemented as a series of match-action tables that process the IoTP's packet payload. In the next chapter, the details of the conduction of the experiments will be presented and the results obtained will be discussed, comparing both the IoTP protocol with the UDP/IP and the IoTP's modified Accretion data aggregation algorithm with the End-to-end's.

Chapter 5

IOTP EVALUATION

This chapter describes the performance evaluation of IoTP. We carried out experiments using the Mininet environment and compared the results with the End-to-End (KIM et al., 2006) aggregation algorithm running over an L2 switch using UDP/IP protocols. The End-to-End aggregation algorithm stores the captured IoT data inside each IoT device. Afterwards, it aggregates the data and, when a certain amount of data has been aggregated, the data is sent out to the IoTP gateway. The main difference between the End-to-End algorithm and the one implemented through IoTP is the amount of data available for aggregation, the protocol stack used (i.e., IoTP vs. UDP/IP), and where the aggregation takes place (either within the network or within the IoT devices).

5.1 EMULATION ENVIRONMENT

The emulation environment used to evaluate the solution proposed in this study is depicted in Figure 5.1. To accomplish it, we ran experiments within an Ubuntu 16.04 LTS virtual machine with kernel 4.4.0-141-generic x86_64 SMP in VirtualBox 5.2.26 PUEL with two Intel Core i7 4500U virtual CPUs running at 2 GHz and with 4 GB of 1666 MHz DDR3 RAM. To install the virtual machine in VirtualBox, we cloned the P4 Tutorial (P4 LANGUAGE CONSORTIUM, 2019a) repository and we ran the "vagrant" tool

on the Debian 9 host operating system, according to the instructions in that repository. For more details about the scripts used to configure the virtual machine, see Appendix A.

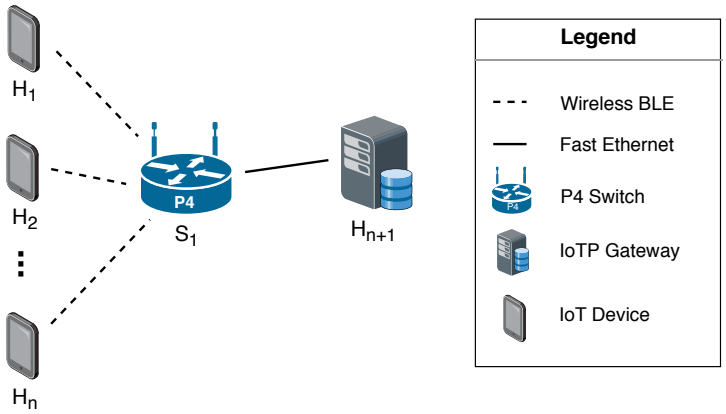


Figure 5.1: IoTP Evaluation Scenario (MADUREIRA; ARAúJO; SAMPAIO, 2020)

The emulation environment used for the experiments is based on Mininet and BMv2 software. We connected a pool of IoT devices and an IoTP gateway to the same switch (i.e., S_1), as Figure 5.1 illustrates. We connected the IoT devices to the switch via Bluetooth Low Energy (BLE) wireless link technology, whereas the IoTP gateway was connected via IEEE 802.3u (Fast Ethernet link). To emulate BLE links, we configured them to reach 1 Mbps maximum throughput and 6 ms delay (i.e., 12 ms RTT). We emulated the 802.3u link by configuring it to reach a maximum throughput of 100 Mbps and 0 ms delay. All the remaining emulation parameters not listed in the Table 5.1 were adopted from the Mininet's defaults. It is worth mentioning that, given that the Mininet emulation environment is highly dependent on the resources available in the virtual machine, the maximum throughput defined for the emulated links is not necessarily achieved by the emulation environment.

To provide the emulated IoT devices with real data, we built a trace using the Intel Berkeley Research Lab data set (MADDEN et al., 2004). This trace contains data from 54 IoT devices that measure room temperature, humidity, light and battery voltages for each device. Each type of data (temperature, humidity, light and voltage) was associated with the IoTP Service IDs 0, 1, 2 and 3, respectively. Then, each Mininet host emulated an IoT device of the trace and that host read the data and send it out to the network,

Table 5.1: Emulation Parameters.

General	Duration of Each Run	10 sec
	Number of Runs	20
Link Technology	IoT Devices	BLE
	IoTP Gateway	802.3u
Link Capacity	IoT Devices	1 Mbps
	IoTP Gateway	100 Mbps
Propagation Delay	IoT Devices	6 ms
	IoTP Gateway	0 ms

according to the timestamps contained in the trace.

5.1.1 Evaluation Methodology

We organized the performance evaluation of the IoTP in three steps. First, we carried out a conformity test in order to establish a baseline that allows us to distinguish the emulation environment performance from the evaluated aggregation mechanisms and protocol stacks. Then, we applied the full factorial design with k factors with two levels each one (2^k design) (JAIN, 1990). Finally, we considered the most effective factors for our performance analysis. In Table 5.2 we describe the chosen factors and their levels. The factors were chosen according to the state-of-the-art literature recommendations. Only the minimum and maximum levels were taken into account for the factorial design phase. For statistical purposes, we replicated the experiment 20 times, extracted the average, the standard deviation and calculated the 95% confidence level. To perform the aforementioned statistical analysis, the following libraries and frameworks were used: Pandas (MCKINNEY, 2010), Seaborn (WASKOM et al., 2017), Matplotlib (HUNTER, 2007) and Jupyter Notebook (KLUYVER et al., 2016).

Table 5.2: Full Factorial Design.

	Factors	**Min - Max**	**Unit**
A	Data Aggregation	10 - 50	data blocks
B	Number of IoT Devices	10 - 50	devices
C	IoTP Sync Flag Send Interval	0 - 0.1	seconds

In Figure 5.2, the Pareto distribution shows the factor's effects over the Average Delay metric. We noticed that all the factors (A, B and C) have a big impact over the Average Delay and that they also interact with each. Therefore, the performance analysis phase considered the combined use of them to evaluate every possible combination of factors and levels. We also added levels 20, 30, and 40 to the B factor to analyze the impact of aggregation strategies according to the number of IoT devices on the network. Hence, from the factors and levels mentioned, the following performance metrics were used to evaluate the proposed solution:

- **Total sent data rate (Kbps)**: Total data contained in the packets sent to the gateway, including the headers and the payload of each packet, divided by the duration of each emulation run;
- **Total transmitted payload rate (Kbps)**: Total payload contained in each packet sent to the gateway, excluding the protocol headers, divided by the duration of each emulation run;
- **Network efficiency (%)**: Ratio between the total payload and the total data sent, which serves as an indicator of efficiency in data transmission on the network, as described by Zechinelli-Martini, Bucciol e Vargas-Solar (2011) ;
- **Total rate of packets sent by the switch (pps)**: Total number of packets sent by the switch to the gateway divided by the duration of each emulation run;
- **Average delay (ms)**: Time interval between the capture of the data by the IoT sensor and its reception by the gateway.
- **Average IoT Device Battery (V)**: The average of the remaining batteries of the IoT devices connected to the network. To evaluate this metric in an emulated environment, we modeled the IoT device battery consumption as a function of the number of transmitted packets. To create such model, we performed a linear regression using the battery data and the number of transmitted packets of each IoT device contained within the Intel Berkeley Research Lab data set.

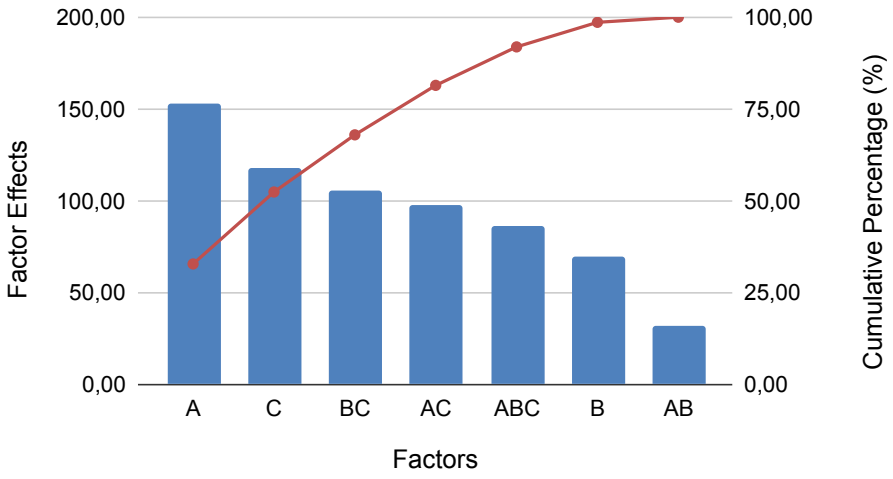


Figure 5.2: Full Factorial Design over the Average Delay metric (MADUREIRA; ARAúJO; SAMPAIO, 2020)

5.2 CONFORMITY TESTS

To evaluate the performance of the emulation environment, we carried out tests using the IoTP protocol and the UDP/IP stack without any data aggregation strategy. Thus, we got baselines to distinguish the impact of the performance of the emulation environment from the impact of the evaluated aggregation mechanisms and protocol stacks.

Comparing the strategies without data aggregation, we realize that when the number of IoT devices increases, IoTP sends fewer packets to the gateway than the UDP/IP strategy, which indicates a processing bottleneck in the IoTP's switch. Such bottleneck in the IoTP's switch is more noticeable when it's connected to 40 or more IoT devices, as depicted in Figure 5.3, and it can be explained by the total number of match-action tables used to implement the aggregation strategy within the IoTP switch. According to (DANG et al., 2017) and (ZHANG et al., 2018), a P4 switch slows down as the number of match-action tables grows. So, to properly execute the IoTP switch's aggregation strategy, the switch checks it's match-action tables and it's internal P4 registers whenever a packet arrives on its network interfaces. Later on, the switch decides whether the aggregation strategy is executed and how it should be done.

The processing bottleneck in the IoTP switch is due to the use of a more significant

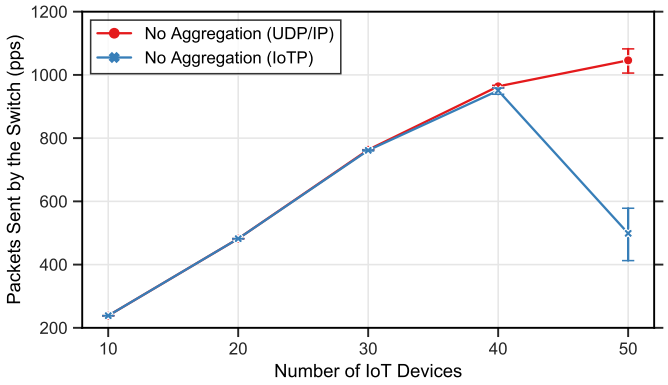


Figure 5.3: Packets sent by the switch in no data aggregation scenario (MADUREIRA; ARAúJO; SAMPAIO, 2020)

number of match-action tables and P4 registers compared to a conventional Ethernet switch. However, the IoTP protocol still presents higher network efficiency than UDP/IP for the no data aggregation scenario, as depicted in Figure 5.4. It indicates that IoTP is able to send a larger payload volume for a given amount of total data sent through the network.

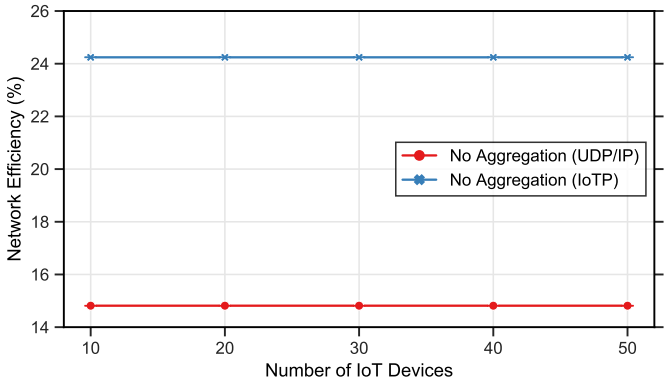
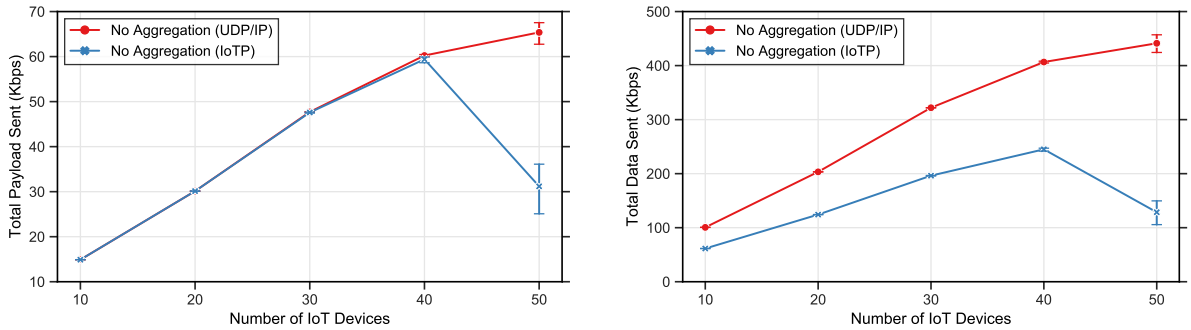


Figure 5.4: Network efficiency in no data aggregation scenario (MADUREIRA; ARAúJO; SAMPAIO, 2020)

In respect to the total payload metric, due to the processing bottleneck, the total payload that IoTP can transmit is also reduced when compared to UDP/IP stack in the no data aggregation scenario, as depicted in Figure 5.5(a). The same behavior is presented in the total data sent, as illustrated by Figure 5.5(b). However, even with such constraint, IoTP showed a greater network efficiency, as depicted in Figure 5.4. This indicates that the IoTP can send a larger volume of payload for a given amount of data

sent.



(a) Total payload sent. (b) Total data sent.

Figure 5.5: Total data in scenario without data aggregation (MADUREIRA; ARAúJO; SAMPAIO, 2020)

Regarding the energy autonomy of IoT devices, UDP/IP has a slight advantage in relation to energy autonomy in relation to IoTP, as depicted in Figure 5.6. However, such slight advantage is not statistically significant.

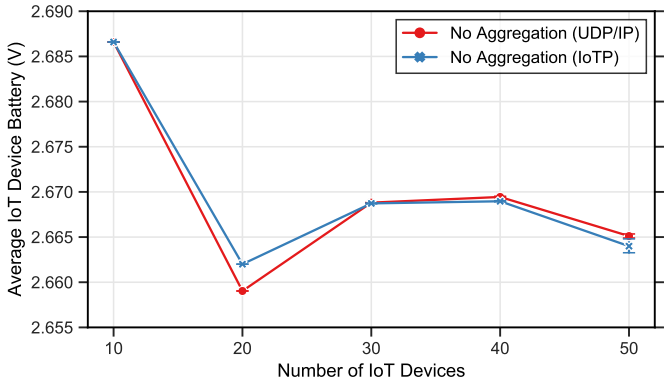


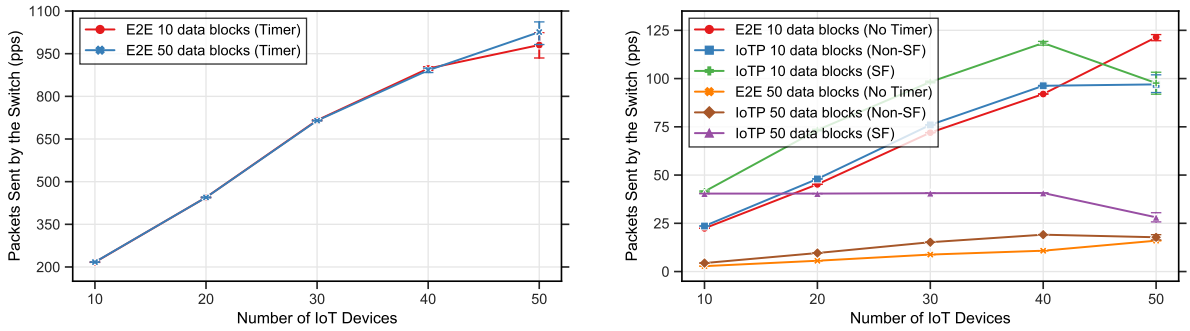
Figure 5.6: Remaining battery of IoT Devices in scenario without data aggregation (MADUREIRA; ARAúJO; SAMPAIO, 2020)

5.2.1 Comparative Analysis

For the sake of simplification, this section uses the term IoTP to refer to the aggregation algorithm implemented in the IoTP switch.

In our experiments, we evaluated the effects of two IoTP strategies that interfere with data aggregation: (i) periodically sending IoTP packets with Sync Flag (SF) enabled (i.e.,

SF IoTP strategy), and (ii) not sending such SF packets (i.e., Non-SF IoTP strategy). When comparing the SF IoTP with the Non-SF IoTP, we noted that there is an increase in the number of packets received by the IoTP gateway when SF IoTP packets are sent, as depicted in Figure 5.7. The SF IoTP strategy works by sending SF IoTP packets periodically to inform the switch that we need to obtain data from IoT devices right away to satisfy a particular deadline. Therefore, the IoTP switch attempts to meet this demand by sending a larger number of packets with a small payload. That is, in SF IoTP, we are prioritizing the deadline over the efficiency of the aggregation strategy implemented within the IoTP switch. In Non-SF IoTP, we prioritize the efficiency of the aggregation strategy over the deadline.

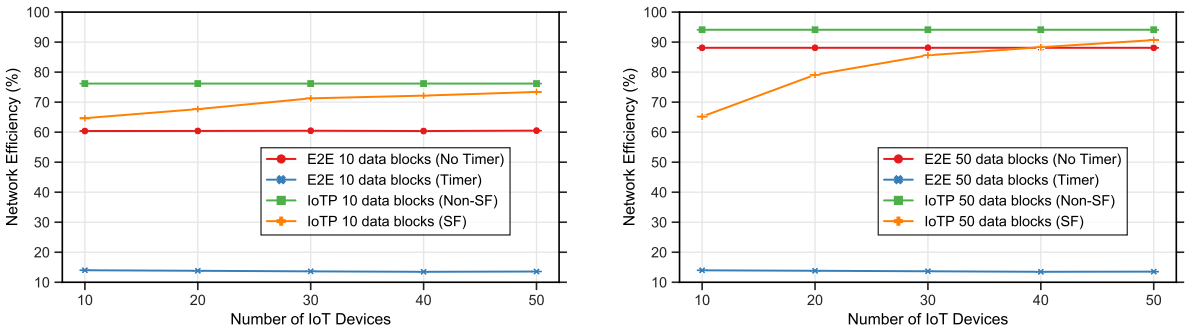


(a) End-to-End Strategy with Timer.

(b) Other aggregation strategies.

Figure 5.7: Packets sent by the switch in data aggregation scenario (MADUREIRA; ARAúJO; SAMPAIO, 2020)

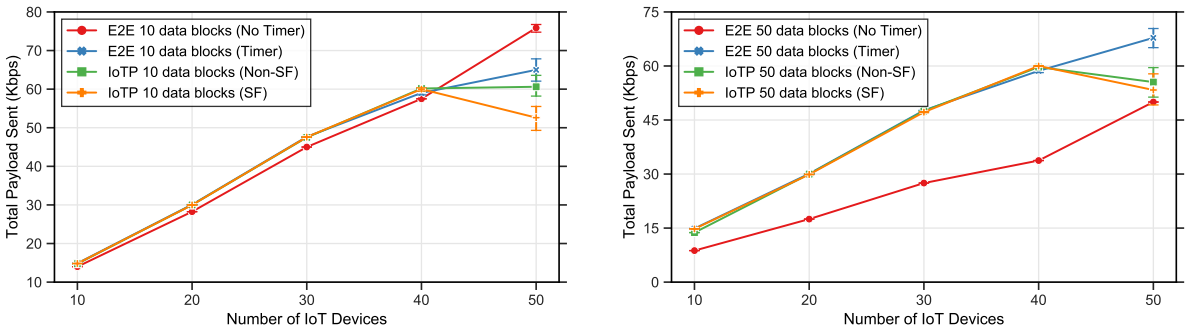
Statistically, End-to-End (E2E) and IoTP presented a similar reduction in the total number of packets sent over the network, with End-to-End presenting a slight advantage over IoTP. As a result of the reduction in the number of packets, the network efficiency is expected to increase (AKYUREK; ROSING, 2018). It can be noted that, in fact, the network efficiency was improved by using IoTP, since the protocol was able to obtain better network efficiency than End-to-End. We also remark that the IoTP's network efficiency decreases when it is periodically sending Sync Flag packets (i.e., SF IoTP), as depicted in Figure 5.8. Due to the use of Timers in End-to-End and Sync Flags in SF IoTP, the average aggregation delay gets a higher priority than data aggregation efficiency.



(a) Up to 10 aggregated data blocks. (b) Up to 50 aggregated data blocks.

Figure 5.8: Network efficiency in the data aggregation scenario (MADUREIRA; ARAúJO; SAMPAIO, 2020)

Concerning the total payload sent, we found a significant difference between the End-to-End strategy with 50 blocks of data when compared to the others, as depicted in Figure 5.9. It stems from the lack of enough time for IoT devices to accumulate 50 data blocks to send a packet with the data aggregate. Thus, the data aggregate that does not reach 50 data blocks is not forwarded by the IoT devices, resulting in a payload loss. This effect was not found in IoTP, since it operates with the data of the IoT devices as a whole, by performing aggregation within the network. Thus, there are more opportunities for IoTP to aggregate data and reach the configured amount of data blocks.



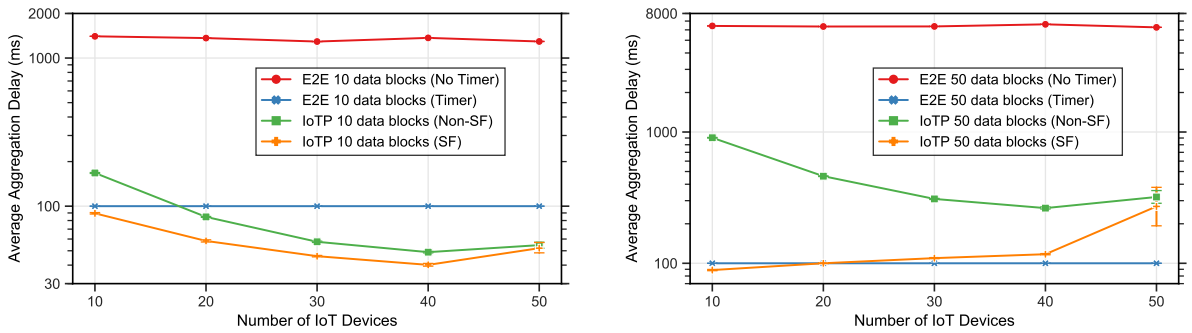
(a) Up to 10 aggregated data blocks. (b) Up to 50 aggregated data blocks.

Figure 5.9: Total payload sent in scenario with data aggregation (MADUREIRA; ARAúJO; SAMPAIO, 2020)

It is worth mentioning that the aforementioned payload loss problem can be mitigated through the use of timers or Sync Flags. They guarantee that the data aggregate will be sent within a certain predefined time interval. However, the efficiency of data aggregation

can be impaired when using such solutions, as depicted in Figure 5.8. Still regarding to the total payload sent, IoTP can send a similar amount of payload to other aggregation strategies, as depicted in Figure 5.9. However, IoTP presents higher network efficiency compared to End-to-End. It indicates that IoTP network link usage is lower than End-to-End's. Therefore, IoTP is more efficient than End-to-End, as depicted in Figure 5.8.

In respect to the average delay, IoTP presented lower values when aggregating up to 10 data blocks in comparison with the End-to-End strategy, as depicted in Figure 5.10. However, when IoTP operates with 50 data blocks, the timer-version of End-to-End could achieve the lowest average delay. We believe that this difference results from the fact that, unlike End-to-End, IoTP does not operate with timers. IoTP is a reactive protocol that can control the average delay by receiving Sync Flag packets (i.e., SF IoTP). Thus, control of the average IoTP delays depends on the timing of sending Sync Flag packets and network characteristics such as propagation delay and link transmission capacity.



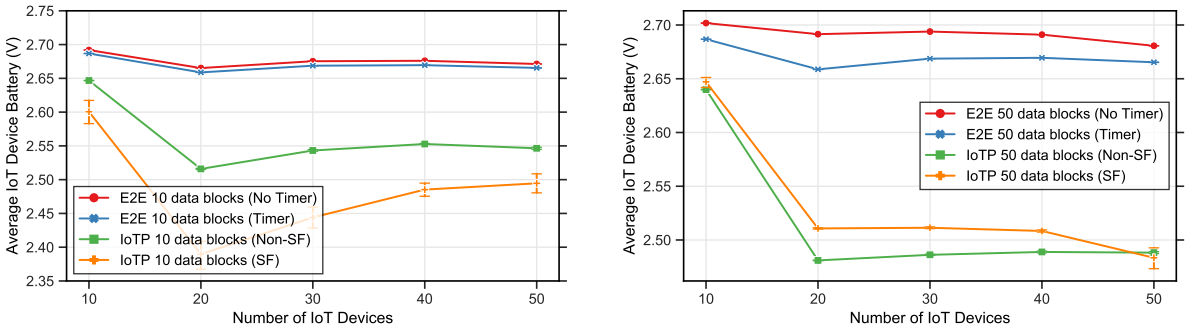
(a) Up to 10 aggregated data blocks.

(b) Up to 50 aggregated data blocks.

Figure 5.10: Average Delay in Data Aggregation Scenario (MADUREIRA; ARAúJO; SAMPAIO, 2020)

Regarding the energy autonomy of IoT devices, End-to-End achieved better energy autonomy than the algorithm implemented within IoTP switches, as depicted in Figure 5.11. We also noticed that the End-to-End without timer provides greater energy autonomy to IoT devices than the End-to-End with a timer (Figure 5.11(b)), due to the lower amount of packets sent by IoT devices (Figure 5.7(b)). When using the End-to-End algorithm, IoT devices aggregate data before sending it to the network. Thus, fewer packets are sent to the network, which improves the energy autonomy of IoT devices.

In experiments carried out with IoTP, the implemented aggregation algorithm performs in-network data aggregation and, therefore, is not able to improve the energy autonomy of IoT devices. However, IoTP is a communication protocol that provides support for several aggregation strategies, which allows algorithms such as End-to-End and other algorithms based on in-network data aggregation to be associated. In this way, the IoTP protocol accomplishes the same energetic autonomy as the End-to-End.



(a) Up to 10 aggregated data blocks.

(b) Up to 50 aggregated data blocks.

Figure 5.11: Remaining IoT Device's Battery in Data Aggregation Scenario (MADUREIRA; ARAúJO; SAMPAIO, 2020)

5.3 LIMITATIONS OF THIS STUDY

The Mininet/BMv2 network emulation environment is very flexible and offers many facilities to the P4 developer. However, this environment has performance limitations that affect P4 programs. When deploying IoTP implementations using the emulation environment, we verified that there is a maximum memory limit that the environment can allocate and properly use. As tests were performed using IoTP, whenever the 200 KB limit of the P4 registers was reached, Mininet/BMv2 suffered a crash and stopped working. Thus, the Mininet emulation environment limited the amount of data that the IoTP could store in a single packet up to 50 data blocks.

5.4 CHAPTER'S SUMMARY

This chapter covered the experimental evaluation of IoTP in detail, from the emulation environment and full factorial design, through the evaluation study of the IoTP and UDP/IP protocol stacks. In addition, it showed the data aggregation performance evaluation employed, as well as a deep discussion of the methodology used, with its relevant performance metrics, and, finally, it showed a detailed analysis of the results obtained in the proposed IoTP solution, comparing it with the End-to-end's.

Chapter 6

CONCLUSION

In Internet of Things (IoT) scenarios, the majority of device power consumption occurs in transmitting packets. Packet aggregation on such networks is traditionally performed within the IoT devices in order to reduce the number of communications with base stations. Network packet aggregation is a technique that combines multiple packets, possibly from different sources, with heterogeneous requirements and characteristics. This combination allows a single packet to be transmitted to the next network node. By reducing the number of packets on the network, aggregation techniques can also increase network efficiency, and eliminate recurring packet headers. In addition, packet aggregation can reduce end-to-end latency and jitter.

IoTP is a low overhead protocol for IoT sensor networks that can improve network efficiency, control the average delay induced by data aggregation, and reduce the overhead that repeated protocol headers cause on the network by providing network information to data aggregation algorithms. IoTP is also able to reduce the number of packets that circulate over the network, which should result in less computational resource consumption inside network switches, such as CPU and memory. Consequently, an improvement in network performance is expected. Such improvements are a consequence of the combination of the IoTP protocol with a data aggregation algorithm that runs alongside it within

the switch. Aggregation within the IoTP switch is possible through the IoTP's service ID field, that marks the packet payload as belonging to an specific IoT application, such as fire detection in a geographical region. However, as expected in any data aggregation solution, as IoTP multi-layer architecture allows data aggregation within network switches, this induces a delay in the IoT data transmission. Nonetheless, IoTP was designed to have more precise control over data aggregation delay using Sync Flag packet. When these packets reach an IoTP switch, it sends all accumulated data to the IoTP gateway immediately. Thus, the IoTP gateway can send these packets periodically, according to the IoT application requirements, promoting a trade-off between the efficiency of the network and its average delay (responsiveness).

IoTP was design to be embedded within the data plane of hardware-based switches alongside with a suitable data aggregation algorithm, considering the node's resources constraints. In this work, IoTP was implemented using Programming Protocol-Independent Packet Processors (P4) and a modified Accretion data aggregation algorithm was deployed alongside the protocol, inside the P4 switches. For the IoTP implementation we considered the constraints associated with the P4 language and the P4 Very Simple Switch (VSS) switch architecture. Such restrictions imposed that our IoTP implementation did not use loops, had only one parser, ingress, egress and deparser control blocks. We also had to store IoT data within the switch's registers using the matrix linearization technique.

Our results show that IoTP has a 78% better network efficiency and is 5 times faster in terms of average delay when compared to the End-to-End data aggregation strategy. Considering the scenario without data aggregation, the IoTP protocol also achieved better network efficiency than UDP/IP protocol stack.

6.1 DISCUSSION

To develop the embedded modified Accretion data aggregation algorithm executed within an IoTP switch, loop unroll and matrix linearization techniques were employed due to P4 implementation constraints, such as the lack of loops and matrix data structures.

However, the IoTP protocol in itself does not require such features, and it is independent of the switch's embedded aggregation algorithm.

The P4 emulation environment also posed some limitations to IoTP's implementation and performance evaluation. Nonetheless, IoTP's evaluation results show that, even in face of such emulation restraints, IoTP achieved its goals, improving the network and data aggregation efficiency.

The following scientific publications were a direct result of this research's efforts:

1. XXXVIII SBRC 2020 – the 2020 edition of the symposium took place from December 7 to 10, 2020 in online format. Article accepted in the main SBRC track, entitled – Um Protocolo IoT para Redução de Tráfego em Redes de Plano de Dados Programáveis. Authors: André L. R. Madureira, Francisco R. C. Araújo, Leobino N. Sampaio.

2. Computer Networks – article published in January 2020, entitled – On supporting IoT data aggregation through programmable data planes. Authors: André L. R. Madureira, Francisco R. C. Araújo, Leobino N. Sampaio.

6.1.1 Follow-up Researches

In parallel to the IoTP's research, another study named NDN-Fabric was conducted in the Named Data Networking (NDN) field, aiming to facilitate NDN's deployment in current commodity-hardware switches based on programmable data plane. The NDN (ZHANG et al., 2014) follows the content-centric model (JACOBSON et al., 2009) and is based on stateful forwarding (GUNDOGRAN et al., 2018). These features bring scalability and robustness of another magnitude. Besides the Forwarding Information Base (FIB), the NDN architecture adds to the router the Pending Interest Table (PIT) and Content Store (CS) data structures (ROHMAH; SUDIHARTO; HERUTOMO, 2017). Together, they favor the efficient forwarding of packets to every interface that requested them in the past, sometimes following a multicast data delivery model. Such a feature makes this architecture ideal for dynamic networking scenarios and significantly benefits networking

applications that focus on the distribution of content. Although the NDN architecture brings significant contributions to network scalability (ZHANG et al., 2019), there lacks a clear understanding of how it will be managed and operated in networking settings characterized by centralized control. The knowledge about network links and location of data repositories within administrative domains may contribute to the design of more efficient and application-tailored forwarding solutions. Operators of such NDN-based infrastructures will likely look for information about router's states to support traffic engineering for performance and optimization. Differently from IP, NDN routers must carry out both read and write operations on PIT tables to keep track of Interest states, not to mention the CS used for caching.

In the opposite direction, the Software-Defined Networks (SDN) (KREUTZ et al., 2015) paradigm is a centralized model that has been discussed in many research papers throughout the last years as an ideal solution for the network operation. In particular, when applied to specific network domains, it introduces a productive way of managing and operating the network from a centralized path-based model (OKAY; OZDEMIR, 2018; BAKTIR; OZGOVDE; ERSOY, 2017). Through the controller's central view, the SDN approach tries to address the lack of intelligence in the IP stateless forwarding plane, by supporting the networking operation with the ability to set up router's forwarding rules on the fly. While the SDN has been proven to be an ideal solution for network operation, it lacks scalability due to the centralized control plane, path-based communication, and stateless forwarding.

The tradeoff between these two networking approaches can be balanced by a solution that explores the best of them. The additional overhead resulted from extra content-centric router's data structures pays off well for uncontrolled networking scenarios (e.g., Internet of Things (IoT), Vehicular networks) that require more scalable, intelligent, and adaptable forwarding strategies. Conversely, stable and controlled environments, e.g., data centers, can benefit from the performance and control of path-based solutions (ARAúJO; SAMPAIO; ZIVIANI, 2017). Hence, the main motivation with our NDN-Fabric proposal is to design an architecture that best assembles the advantages of path-

based and content-centric models through the SDN and NDN architectures.

Inspired by the fabric model (CASADO et al., 2012), NDN-Fab suggests the coupled use of content-centric and path-based models, by applying NDN routers at the network edge and SDN-based operation in the network's backhaul. For better efficiency on packet delivery and network control, it relies on segment routing to seamless and efficiently forward data within the fabric through pre-defined and optimal network paths maintained by an SDN-based fabric network controller. Due to these features, this work is threefold, since NDN-Fab: (i) enables different packet forwarding strategies to take place at the network's edge and core, (ii) improves the efficiency of packet forwarding in the network's core through segment-routing, and (iii) favors the designing of traffic engineering techniques in the network's core, such as multi-path packet sending, load balancing, packet steering, and redundant routes.

To accomplish its goals, NDN-Fab architecture relies on P4-based data plane programmability(BOSSHART et al., 2014) and the centralized network controller. It was implemented inside the Mininet emulation environment (LANTZ; HELLER; MCKEOWN, 2010), in which we conducted the experiments that served as a proof of concept. Emulation results demonstrated that NDN-Fab could handle efficiently NDN Data packets delivery and overcome legacy forwarding protocols, i.e., IPv4. Moreover, it uses the traditional IP-based fabric model (CASADO et al., 2012) in which the edge switches that interconnect the edge and core networks are named Instruction Reconstructors (IR). Those switches are responsible for informing the P4 network controller about incoming packets from the edges of the network. They are also responsible for converting the packet's forwarding strategy between the core and edge networks. To this end, the reconstructor switch intercepts incoming Interest and Data packets and requests to the network controller forwarding instructions used for segment-routing.

The following scientific publications were a direct result of this research's efforts:

1. XXXVIII SBRC 2020 – the 2020 edition of the symposium took place from December 7 to 10, 2020 in online format. Article accepted in the main SBRC track, entitled – NDN-ADAP: Uma Arquitetura para Encaminhamento Eficiente de Pa-

cotes em Redes de Dados Nomeados. Authors: André L. R. Madureira, Francisco R. C. Araújo, Lucas N. B. Prates, Leobino N. Sampaio.

2. IEEE Transactions on Network and Service Management – article accepted for publication in December 2020, entitled – NDN Fabric: Where the Software-Defined Networking Meets the Content-Centric Model. Authors: André L. R. Madureira, Francisco R. C. Araújo, Guilherme B. Araújo, Leobino N. Sampaio.

6.2 FUTURE WORKS

As a suggestion for future work with IoTP, we intend to pursue the following research topics:

1. **Integration with other data aggregation algorithms**: In this dissertation work we implemented a modified version of the Accretion data aggregation algorithm (KIM et al., 2006). However, IoTP can be integrated with a wide plethora of algorithms, achieving different results depending on where in the network they are deployed. Therefore, two major points need further research: (i) where to deploy the data aggregation algorithms, and (ii) which data aggregation algorithms are suitable for which network node, considering its resource constraints and the hierarchical multi-layer data aggregation network topology.

2. **Extend the protocol to provide security**: There is an increasing concern regarding IoT security and privacy in the literature (BERA; MISRA; VASILAKOS, 2017). To a certain extent, as IoTP aggregates data from multiple IoT devices into a packet within the network, IoTP provides some privacy regarding IoT device identification. However, the protocol was not designed with this goal, and it does not provide authentication or other security-related features. Thus, we intend to assess ways to incorporate such relevant features into the protocol in future researches.

Another investigation that we seek to evaluate during the Ph.D. research is how we can harness the best performance out of the NDN-Fabric architecture, mainly through the following lines of research:

1. **Network controller performance**: NDN-Fabric performance is tightly associated with its controller and, for this reason, performance optimizations will be required. There are many different approaches to reach higher performance within the controller, such as routing information caching, deployment of multiple local controllers throughout the network, use of dynamic packet steering techniques, among others. Thus, further investigation is required;

2. **Enhanced data delivery with NDN repositories (NDN repo)**: NDN-Fabric data delivery can be optimized through the deployment of NDN repos at specific points of the network. However, the optimum amount of repos and their points of deployment throughout the network are still debatable within the NDN literature, requiring additional research;

3. **Improved data delivery reliability with native NDN repo synchronization**: Multiple NDN repos can be deployed within NDN networks, but they serve little to no purpose if they represent a single point of failure. That is, if each of them stores a specific set of data content when one of them fails, the network data delivery is compromised. Thus, we intend to research how NDN-Fabric can provide native NDN repo synchronization within the network, without relying on additional protocols, thereby improving network data delivery reliability and redundancy;

4. **Improved hash function**: NDN-Fabric's hash function has the important role of mapping a content name with a fixed-length label to improve packet forwarding within the network. However, such a process requires a highly efficient hash function, that does not significantly comprise the forwarding performance. This aspect of NDN-Fabric's is also within our research goals, as there is a wide variety of functions with different benefits and drawbacks that NDN-Fabric could use for the hashing purpose.

BIBLIOGRAPHY

AKYUREK, A. S.; ROSING, T. S. Optimal in-network packet aggregation policy for maximum information freshness. In: *2016 European Conference on Networks and Communications (EuCNC)*. [S.l.: s.n.], 2016. p. 89–93.

AKYUREK, A. S.; ROSING, T. S. Optimal packet aggregation scheduling in wireless networks. *IEEE Transactions on Mobile Computing*, v. 17, n. 12, p. 2835–2852, Dec 2018. ISSN 1536-1233.

ALGHAMDI, A.; ALSHAMRANI, M.; ALQAHTANI, A.; GHAMDI, S. S. A. A.; HARRATHI, R. Secure data aggregation scheme in wireless sensor networks for iot. In: *2016 International Symposium on Networks, Computers and Communications (ISNCC)*. [S.l.: s.n.], 2016. p. 1–5.

ALKHAMISI, A.; NAZMUDEEN, M. S. H.; BUHARI, S. M. A cross-layer framework for sensor data aggregation for iot applications in smart cities. In: *2016 IEEE International Smart Cities Conference (ISC2)*. [S.l.: s.n.], 2016. p. 1–6.

ARAúJO, A. C. d. S.; SAMPAIO, L. N.; ZIVIANI, A. Beep: Balancing energy, redundancy, and performance in fat-tree data center networks. *IEEE Internet Computing*, v. 21, n. 4, p. 44–53, 2017.

ARBETTU, R. K.; KHONDOKER, R.; BAYAROU, K.; WEBER, F. Security analysis of opendaylight, onos, rosemary and `Ryu` sdn controllers. In: *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*. [S.l.: s.n.], 2016. p. 37–44.

ARSHAD, S.; AZAM, M. A.; REHMANI, M. H.; LOO, J. Recent advances in information-centric networking based internet of things (icn-iot). *IEEE Internet of Things Journal*, p. 1–1, 2019. ISSN 2327-4662.

BAKTIR, A. C.; OZGOVDE, A.; ERSOY, C. How can edge computing benefit from software-defined networking: A survey, use cases, and future directions. *IEEE Communications Surveys Tutorials*, v. 19, n. 4, p. 2359–2391, Fourthquarter 2017. ISSN 1553-877X.

BAKTIR, A. C.; OZGOVDE, A.; ERSOY, C. Implementing service-centric model with p4: A fully-programmable approach. In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. [S.l.: s.n.], 2018. p. 1–6. ISSN 2374-9709.

BERA, S.; MISRA, S.; VASILAKOS, A. V. Software-defined networking for internet of things: A survey. *IEEE Internet of Things Journal*, v. 4, n. 6, p. 1994–2008, Dec 2017. ISSN 2327-4662.

BERNARDOS, C. J.; OLIVA, A. de la; SERRANO, P.; BANCHS, A.; CONTRERAS, L. M.; JIN, H.; ZUNIGA, J. C. An architecture for software defined wireless networking. *IEEE Wireless Communications*, v. 21, n. 3, p. 52–61, June 2014. ISSN 1536-1284.

BOSSHART, P.; DALY, D.; GIBB, G.; IZZARD, M.; MCKEOWN, N.; REXFORD, J.; SCHLESINGER, C.; TALAYCO, D.; VAHDAT, A.; VARGHESE, G.; WALKER, D. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 87–95, jul. 2014. ISSN 0146-4833. Disponível em: ⟨https://doi.org/10.1145/2656877.2656890⟩.

CASADO, M.; KOPONEN, T.; SHENKER, S.; TOOTOONCHIAN, A. Fabric: A Retrospective on Evolving SDN. In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. New York, NY, USA: ACM, 2012. (HotSDN '12), p. 85–90. ISBN 978-1-4503-1477-0.

DANG, H. T.; WANG, H.; JEPSEN, T.; BREBNER, G.; KIM, C.; REXFORD, J.; SOULÉ, R.; WEATHERSPOON, H. Whippersnapper: A p4 language benchmark suite. In: *Proceedings of the Symposium on SDN Research*. New York, NY, USA: ACM, 2017. (SOSR '17), p. 95–101. ISBN 978-1-4503-4947-5. Disponível em: ⟨http://doi.acm.org/10.1145/3050220.3050231⟩.

DARGAHI, T.; CAPONI, A.; AMBROSIN, M.; BIANCHI, G.; CONTI, M. A survey on the security of stateful sdn data planes. *IEEE Communications Surveys Tutorials*, v. 19, n. 3, p. 1701–1725, thirdquarter 2017. ISSN 1553-877X.

GUNDOGRAN, C.; KIETZMANN, P.; LENDERS, M.; PETERSEN, H.; SCHMIDT, T. C.; WäHLISCH, M. Ndn, coap, and mqtt: A comparative measurement study in the iot. In: *Proceedings of the 5th ACM Conference on Information-Centric Networking*. New York, NY, USA: Association for Computing Machinery, 2018. (ICN '18), p. 159–171. ISBN 9781450359597. Disponível em: ⟨https://doi.org/10.1145/3267955.3267967⟩.

GUPTA, V.; KAUR, K.; KAUR, S. Network programmability using software defined networking. In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. [S.l.: s.n.], 2016. p. 1170–1173.

HEINZELMAN, W. B.; CHANDRAKASAN, A. P.; BALAKRISHNAN, H. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, v. 1, n. 4, p. 660–670, Oct 2002. ISSN 1536-1276.

HU, F.; CAO, X.; MAY, C. Optimized scheduling for data aggregation in wireless sensor networks. In: *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*. [S.l.: s.n.], 2005. v. 2, p. 557–561 Vol. 2.

HU, F.; HAO, Q.; BAO, K. A survey on software-defined network and openflow: From concept to implementation. *IEEE Communications Surveys Tutorials*, v. 16, n. 4, p. 2181–2206, Fourthquarter 2014. ISSN 1553-877X.

HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science Engineering*, v. 9, n. 3, p. 90–95, May 2007. ISSN 1521-9615.

HYUN, J.; HONG, J. W. Knowledge-defined networking using in-band network telemetry. In: *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. [S.l.: s.n.], 2017. p. 54–57.

JACOBSON, V.; SMETTERS, D. K.; THORNTON, J. D.; PLASS, M. F.; BRIGGS, N. H.; BRAYNARD, R. L. Networking named content. In: ACM. *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. [S.l.], 2009. p. 1–12.

JAIN, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, 1990. ISBN 9788126519057. Disponível em: ⟨https://books.google.com.br/books?id=eOR0kJjgMqkC⟩.

JAIN, R.; PAUL, S. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine*, v. 51, n. 11, p. 24–31, November 2013. ISSN 0163-6804.

KARIM, L.; AL-KAHTANI, M. S. Sensor data aggregation in a multi-layer big data framework. In: *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. [S.l.: s.n.], 2016. p. 1–7.

KARLSSON, J.; KASSLER, A.; BRUNSTROM, A. Impact of packet aggregation on tcp performance in wireless mesh networks. In: *2009 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks Workshops*. [S.l.: s.n.], 2009. p. 1–7.

KATTA, N.; ALIPOURFARD, O.; REXFORD, J.; WALKER, D. Cacheflow: Dependency-aware rule-caching for software-defined networks. In: *Proceedings of the Symposium on SDN Research*. New York, NY, USA: ACM, 2016. (SOSR '16), p. 6:1–6:12. ISBN 978-1-4503-4211-7. Disponível em: ⟨http://doi.acm.org/10.1145/2890955.2890969⟩.

KIM, H.; FEAMSTER, N. Improving network management with software defined networking. *IEEE Communications Magazine*, v. 51, n. 2, p. 114–119, February 2013. ISSN 0163-6804.

KIM, K.; GANGULY, S.; IZMAILOV, R.; HONG, S. On packet aggregation mechanisms for improving voip quality in mesh networks. In: *2006 IEEE 63rd Vehicular Technology Conference*. [S.l.: s.n.], 2006. v. 2, p. 891–895. ISSN 1550-2252.

KIM, K.; MIN, S.; HAN, Y. A programmable data plane to support in-network data processing in software-defined iot. In: *2017 International Conference on Information and Communication Technology Convergence (ICTC)*. [S.l.: s.n.], 2017. p. 855–860.

KLUYVER, T.; RAGAN-KELLEY, B.; PÉREZ, F.; GRANGER, B.; BUSSONNIER, M.; FREDERIC, J.; KELLEY, K.; HAMRICK, J.; GROUT, J.; CORLAY, S.; IVANOV, P.; AVILA, D.; ABDALLA, S.; WILLING, C. Jupyter notebooks – a publishing format for reproducible computational workflows. In: LOIZIDES, F.; SCHMIDT, B. (Ed.). *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. [S.l.], 2016. p. 87–90.

KREUTZ, D.; RAMOS, F. M. V.; VERíSSIMO, P. E.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, v. 103, n. 1, p. 14–76, Jan 2015. ISSN 0018-9219.

LAISSAOUI, C.; IDBOUFKER, N.; ELASSALI, R.; BAAMRANI, K. E. A measurement of the response times of various openflow/sdn controllers with cbench. In: *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*. [S.l.: s.n.], 2015. p. 1–2.

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: ACM, 2010. (Hotnets-IX), p. 19:1–19:6. ISBN 978-1-4503-0409-2. Disponível em: ⟨http://doi.acm.org/10.1145/1868447.1868466⟩.

MADDEN, S.; BODIK, P.; HONG, W.; GUESTRIN, C.; PASKIN, M.; THIBAUX, R. *Intel Lab Data*. Intel Berkeley Research lab, 2004. Disponível em: ⟨http://db.csail.mit.edu/labdata/labdata.html⟩. Acesso em: 18 de maio de 2019.

MADUREIRA, A. L. R.; ARAúJO, F. R. C.; SAMPAIO, L. N. On supporting iot data aggregation through programmable data planes. *Computer Networks*, v. 177, p. 107330, 2020. ISSN 1389-1286. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S1389128620301195⟩.

MCKINNEY, W. Data structures for statistical computing in python. In: WALT, S. van der; MILLMAN, J. (Ed.). *Proceedings of the 9th Python in Science Conference*. [S.l.: s.n.], 2010. p. 51 – 56.

NUNES, B. A. A.; MENDONCA, M.; NGUYEN, X. N.; OBRACZKA, K.; TURLETTI, T. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys Tutorials*, v. 16, n. 3, p. 1617–1634, Third 2014. ISSN 1553-877X.

OKAY, F. Y.; OZDEMIR, S. Routing in fog-enabled iot platforms: A survey and an sdn-based solution. *IEEE Internet of Things Journal*, v. 5, n. 6, p. 4871–4889, Dec 2018. ISSN 2327-4662.

P4 LANGUAGE CONSORTIUM. *P4 Tutorial*. GitHub, 2019. GitHub Repository. Disponível em: ⟨https://github.com/p4lang/tutorials⟩. Acesso em: 11 de abril de 2019.

P4 LANGUAGE CONSORTIUM. *$P4_{16}$ Language Specification v1.2.0*. 2019. Disponível em: ⟨https://p4.org/p4-spec/docs/P4-16-v1.2.0.pdf⟩. Acesso em: 17 de fevereiro de 2020.

PAKZAD, F.; PORTMANN, M.; HAYWARD, J. Link capacity estimation in wireless software defined networks. In: *2015 International Telecommunication Networks and Applications Conference (ITNAC)*. [S.l.: s.n.], 2015. p. 208–213.

RAHMAN, H.; AHMED, N.; HUSSAIN, I. Comparison of data aggregation techniques in internet of things (iot). In: *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. [S.l.: s.n.], 2016. p. 1296–1300.

RASTOGI, A.; BAIS, A. Comparative analysis of software defined networking (sdn) controllers 2014; in terms of traffic handling capabilities. In: *2016 19th International Multi-Topic Conference (INMIC)*. [S.l.: s.n.], 2016. p. 1–6.

ROHMAH, Y. N.; SUDIHARTO, D. W.; HERUTOMO, A. The performance comparison of forwarding mechanism between ipv4 and named data networking (ndn). case study: A node compromised by the prefix hijack. In: *2017 3rd International Conference on Science in Information Technology (ICSITech)*. [S.l.: s.n.], 2017. p. 302–306.

SAPIO, A.; ABDELAZIZ, I.; ALDILAIJAN, A.; CANINI, M.; KALNIS, P. In-network computation is a dumb idea whose time has come. In: *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. New York, NY, USA: ACM, 2017. (HotNets-XVI), p. 150–156. ISBN 9781450355698.

SEZER, S.; SCOTT-HAYWARD, S.; CHOUHAN, P. K.; FRASER, B.; LAKE, D.; FINNEGAN, J.; VILJOEN, N.; MILLER, M.; RAO, N. Are we ready for sdn? implementation challenges for software-defined networks. *IEEE Communications Magazine*, v. 51, n. 7, p. 36–43, July 2013. ISSN 0163-6804.

SHEN, Y.; ZHANG, T.; WANG, Y.; WANG, H.; JIANG, X. Microthings: A generic iot architecture for flexible data aggregation and scalable service cooperation. *IEEE Communications Magazine*, v. 55, n. 9, p. 86–93, Sep. 2017. ISSN 0163-6804.

SRUTHI, S. S.; GEETHAKUMARI, G. An efficient secure data aggregation technique for internet of things network: An integrated approach using db-mac and multi-path topology. In: *2016 IEEE 6th International Conference on Advanced Computing (IACC)*. [S.l.: s.n.], 2016. p. 599–603.

STANCU, A. L.; HALUNGA, S.; VULPE, A.; SUCIU, G.; FRATU, O.; POPOVICI, E. C. A comparison between several software defined networking controllers. In: *2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS)*. [S.l.: s.n.], 2015. p. 223–226.

SVIRIDOV, G.; BONOLA, M.; TULUMELLO, A.; GIACCONE, P.; BIANCO, A.; BIANCHI, G. Lodge: Local decisions on global states in programmable data planes. In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. [S.l.: s.n.], 2018. p. 257–261.

TANTAYAKUL, K.; DHAOU, R.; PAILLASSA, B.; PANICHPATTANAKUL, W. Experimental analysis in sdn open source environment. In: *2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. [S.l.: s.n.], 2017. p. 334–337.

Tortonesi, M.; Michaelis, J.; Morelli, A.; Suri, N.; Baker, M. A. Spf: An sdn-based middleware solution to mitigate the iot information explosion. In: *2016 IEEE Symposium on Computers and Communication (ISCC)*. [S.l.: s.n.], 2016. p. 435–442.

WASKOM, M.; BOTVINNIK, O.; O'KANE, D.; HOBSON, P.; LUKAUSKAS, S.; GEMPERLINE, D. C.; AUGSPURGER, T.; HALCHENKO, Y.; COLE, J. B.; WARMENHOVEN, J.; RUITER, J. de; PYE, C.; HOYER, S.; VANDERPLAS, J.; VILLALBA, S.; KUNTER, G.; QUINTERO, E.; BACHANT, P.; MARTIN, M.; MEYER, K.; MILES, A.; RAM, Y.; YARKONI, T.; WILLIAMS, M. L.; EVANS, C.; FITZGERALD, C.; BRIAN; FONNESBECK, C.; LEE, A.; QALIEH, A. *mwaskom/seaborn: v0.8.1 (September 2017)*. 2017. Disponível em: ⟨https://doi.org/10.5281/zenodo.883859⟩. Acesso em: 27 de maio de 2019.

WISSINGH, B.; D'ACUNTO, L.; TRICHIAS, K. In-network data aggregation in icn: Demo paper. In: *2017 8th International Conference on the Network of the Future (NOF)*. [S.l.: s.n.], 2017. p. 129–131.

XIA, W.; WEN, Y.; FOH, C. H.; NIYATO, D.; XIE, H. A survey on software-defined networking. *IEEE Communications Surveys Tutorials*, v. 17, n. 1, p. 27–51, Firstquarter 2015. ISSN 1553-877X.

XU, J.; ZENG, S.; QU, F. A new in-network data aggregation technology of wireless sensor networks. In: *2006 Semantics, Knowledge and Grid, Second International Conference on*. [S.l.: s.n.], 2006. p. 104–104.

YEGANEH, S. H.; TOOTOONCHIAN, A.; GANJALI, Y. On scalability of software-defined networking. *IEEE Communications Magazine*, v. 51, n. 2, p. 136–141, February 2013. ISSN 0163-6804.

YOKOTANI, T.; SHIMUZU, A.; SASAKI, Y.; MUKAI, H. Proposals for packet processing and performance evaluation of iot devices. In: *2017 Japan-Africa Conference on Electronics, Communications and Computers (JAC-ECC)*. [S.l.: s.n.], 2017. p. 5–8.

ZECHINELLI-MARTINI, J. L.; BUCCIOL, P.; VARGAS-SOLAR, G. Energy aware data aggregation in wireless sensor networks. In: *2011 2nd International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE)*. [S.l.: s.n.], 2011. p. 1–5.

ZHANG, C.; BI, J.; ZHOU, Y.; WU, J.; LIU, B.; LI, Z.; DOGAR, A. B.; WANG, Y. P4db: On-the-fly debugging of the programmable data plane. In: *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. [S.l.: s.n.], 2017. p. 1–10.

ZHANG, C.; BI, J.; ZHOU, Y.; ZHANG, K.; MA, Z. B-cache: A behavior-level caching framework for the programmable data plane. In: *2018 IEEE Symposium on Computers and Communications (ISCC)*. [S.l.: s.n.], 2018. p. 00084–00090. ISSN 1530-1346.

ZHANG, L.; AFANASYEV, A.; BURKE, J.; JACOBSON, V.; CLAFFY, K.; CROWLEY, P.; PAPADOPOULOS, C.; WANG, L.; ZHANG, B. Named Data Networking. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 44, n. 3, p. 66–73, Jul. 2014. ISSN 0146-4833.

ZHANG, Y.; XIA, Z.; AFANASYEV, A.; ZHANG, L. A note on routing scalability in named data networking. In: *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*. [S.l.: s.n.], 2019. p. 1–6.

Appendix A

SCRIPTS

A.1 P4 ENVIRONMENT CONFIGURATION

```
#!/bin/bash

# Print commands and exit on errors
set -xe

DEBIAN_FRONTEND=noninteractive sudo add-apt-repository -y ppa
    ↪ :webupd8team/sublime-text-3
DEBIAN_FRONTEND=noninteractive sudo add-apt-repository -y ppa
    ↪ :webupd8team/atom

apt-get update

KERNEL=$(uname -r)
DEBIAN_FRONTEND=noninteractive apt-get -y -o Dpkg::Options::=
    ↪ "--force-confdef" -o Dpkg::Options::="--force-confold"
    ↪ upgrade
apt-get install -y --no-install-recommends \
  atom \
  autoconf \
  automake \
  apt-transport-https \
  bison \
```

```
19  build-essential \
20  ca-certificates \
21  cmake \
22  cpp \
23  curl \
24  emacs24 \
25  flex \
26  git \
27  hping3 \
28  iperf \
29  iperf3 \
30  nmap \
31  libboost-dev \
32  libboost-filesystem-dev \
33  libboost-iostreams1.58-dev \
34  libboost-program-options-dev \
35  libboost-system-dev \
36  libboost-test-dev \
37  libboost-thread-dev \
38  libc6-dev \
39  libevent-dev \
40  libffi-dev \
41  libfl-dev \
42  libgc-dev \
43  libgc1c2 \
44  libgflags-dev \
45  libgmp-dev \
46  libgmp10 \
47  libgmpxx4ldbl \
48  libjudy-dev \
49  libpcap-dev \
50  libreadline6 \
51  libreadline6-dev \
52  libssl-dev \
53  libtool \
54  linux-headers-$KERNEL\
55  lubuntu-desktop \
56  make \
57  mktemp \
58  pkg-config \
59  python \
60  python-dev \
61  python-ipaddr \
62  python-pip \
```

```
63   python-scapy \
64   python-setuptools \
65   sublime-text-installer \
66   tcpdump \
67   tcpreplay \
68   fonts-noto \
69   fonts-roboto \
70   unzip \
71   vim \
72   wget \
73   xcscope-el \
74   xterm
75
76 apt-get install -y --no-install-recommends ttf-mscorefonts-
     ↪ installer
77 rm -rf /root/.cache/matplotlib/*
78 find /home/ -mindepth 1 -maxdepth 1 -name '*' -exec rm -rf
     ↪ {}/.cache/matplotlib/* \;
```

Listing A.1 : Dependency Installations

```
1  #!/bin/bash
2
3  # Print script commands.
4  set -x
5  # Exit on errors.
6  set -e
7
8  BMV2_COMMIT="7e25eeb19d01eee1a8e982dc7ee90ee438c10a05"
9  PI_COMMIT="219b3d67299ec09b49f433d7341049256ab5f512"
10 P4C_COMMIT="48a57a6ae4f96961b74bd13f6bdeac5add7bb815"
11 PROTOBUF_COMMIT="v3.2.0"
12 GRPC_COMMIT="v1.3.2"
13
14 NUM_CORES=`grep -c ^processor /proc/cpuinfo`
15
16 # Mininet
17 git clone git://github.com/mininet/mininet mininet
18 cd mininet
19 sudo ./util/install.sh -nwv
20 cd ..
21
22 # Protobuf
23 git clone https://github.com/google/protobuf.git
24 cd protobuf
```

```
25 git checkout ${PROTOBUF_COMMIT}
26 export CFLAGS="-Os"
27 export CXXFLAGS="-Os"
28 export LDFLAGS="-Wl,-s"
29 ./autogen.sh
30 ./configure --prefix=/usr
31 make -j${NUM_CORES}
32 sudo make install
33 sudo ldconfig
34 unset CFLAGS CXXFLAGS LDFLAGS
35 # force install python module
36 cd python
37 sudo python setup.py install
38 cd ../..
39
40 # gRPC
41 git clone https://github.com/grpc/grpc.git
42 cd grpc
43 git checkout ${GRPC_COMMIT}
44 git submodule update --init --recursive
45 export LDFLAGS="-Wl,-s"
46 make -j${NUM_CORES}
47 sudo make install
48 sudo ldconfig
49 unset LDFLAGS
50 cd ..
51 # Install gRPC Python Package
52 sudo pip install grpcio
53
54 # BMv2 deps (needed by PI)
55 git clone https://github.com/p4lang/behavioral-model.git
56 cd behavioral-model
57 git checkout ${BMV2_COMMIT}
58 # From bmv2's install_deps.sh, we can skip apt-get install.
59 # Nanomsg is required by p4runtime, p4runtime is needed by
      ↪ BMv2...
60 tmpdir=`mktemp -d -p .`
61 cd ${tmpdir}
62 bash ../travis/install-thrift.sh
63 bash ../travis/install-nanomsg.sh
64 sudo ldconfig
65 bash ../travis/install-nnpy.sh
66 cd ..
67 sudo rm -rf $tmpdir
```

```
68 cd ..
69
70 # PI/P4Runtime
71 git clone https://github.com/p4lang/PI.git
72 cd PI
73 git checkout ${PI_COMMIT}
74 git submodule update --init --recursive
75 ./autogen.sh
76 ./configure --with-proto
77 make -j${NUM_CORES}
78 sudo make install
79 sudo ldconfig
80 cd ..
81
82 # Bmv2
83 cd behavioral-model
84 ./autogen.sh
85 ./configure --enable-debugger --with-pi
86 make -j${NUM_CORES}
87 sudo make install
88 sudo ldconfig
89 # Simple_switch_grpc target
90 cd targets/simple_switch_grpc
91 ./autogen.sh
92 ./configure --with-thrift
93 make -j${NUM_CORES}
94 sudo make install
95 sudo ldconfig
96 cd ..
97 cd ..
98 cd ..
99
100 # P4C
101 git clone https://github.com/p4lang/p4c
102 cd p4c
103 git checkout ${P4C_COMMIT}
104 git submodule update --init --recursive
105 mkdir -p build
106 cd build
107 cmake ..
108 make -j${NUM_CORES}
109 make -j${NUM_CORES} check
110 sudo make install
111 sudo ldconfig
```

```
112 cd ..
113 cd ..
114
115 #cleanup
116 sudo rm -r behavioral-model/ grpc/ mininet/ p4c/ PI/ protobuf
    ↪ /
```

Listing A.2 : Installation of the P4 Environment

```
1 #!/bin/bash
2 sudo pip install -U pip
3 sudo pip install -U six wheel
4 sudo pip install -U pandas seaborn matplotlib numpy
    ↪ statsmodels jupyter
```

Listing A.3 : Installing Python Dependencies