Dissertação de Mestrado

Inatel
Instituto Nacional de Telecomunicações

NovaGenesis Control Agent for Future Internet eXchange Point

THIAGO BUENO DA SILVA

DEZEMBRO / 2021

NOVAGENESIS CONTROL AGENT FOR FUTURE INTERNET EXCHANGE POINT

THIAGO BUENO DA SILVA

Dissertação apresentada ao Instituto Nacional de Telecomunicações, como parte dos requisitos para obtenção do Título de Mestre em Telecomunicações.

ORIENTADOR: Prof. Dr. Antônio Marcos Alberti.

Santa Rita do Sapucaí
2021

Silva, Thiago Bueno da Silva
S586n
NovaGenesis Control Agent for Future Internet eXchange Point / Thiago Bueno da Silva. – Santa Rita do Sapucaí, 2021.
167 p.

Orientador: Prof. Dr. Antônio Marcos Alberti.
Dissertação de Mestrado em Telecomunicações – Instituto Nacional de Telecomunicações – INATEL.
Inclui bibliografia e anexo.

1. NovaGenesis 2. Future Internet eXchange Point 3. P4 4. Future Internet Architecture. 5. Software-Defined Network. 6. Mestrado em Telecomunicações. I. Alberti, Antônio Marcos. II. Instituto Nacional de Telecomunicações – INATEL. III. Título.

CDU 621.39

Ficha Catalográfica elaborada pelo Bibliotecário da Instituição
CRB6-2718 – Paulo Eduardo de Almeida

FOLHA DE APROVAÇÃO

Dissertação defendida e aprovada em _____/_____/_____, pela comissão julgadora:

Prof. Dr. Antônio Marcos Alberti
INATEL

Prof. Dr. Cristiano Bonato Both
UNISINOS

Prof. Dr. Guilherme Augusto Barucke Marcondes
INATEL

Prof. Dr. Antônio Marcos Alberti
INATEL

Prof. Dr. José Marcos Câmara Brito
Coordenador do Curso de Mestrado

You’re alive, Bod. That means you have infinite potential. You can do anything, make anything, dream anything.

Neil Gaiman

Aos meus pais,
meus primeiros grandes professores.

Agradecimentos

É curioso olhar toda esta jornada e ter este momento para agradecer. Muitas coisas mudaram, mas poucas foram de fato perdidas pelo caminho. Muitas vezes a tela vazia, do que viria a ser um código ou mesmo este documento, acabou me vencendo. No entanto, sou afortunado por contar com pessoas que souberam reconhecer que nem todos os que vagam estão perdidos. Assim, agradeço a Deus por ter pavimentado esse mundo caótico com pessoas que iluminaram o meu caminho quando tudo parecia perdido.

Dentre essas pessoas, reconheço a importância dos meus pais, Antônio e Cláudia, e minha irmã, Thamires, por todo o suporte, motivação e carinho. Agradeço também a todos os familiares que se dispuseram a me incentivar diretamente e por meio de orações. Aos meus amigos, saibam que possuem parte da culpa pelas suas palavras sábias (e nem sempre delicadas).

Agradeço ao professor e orientador Antônio Marcos Alberti por toda a sua disposição em me guiar. Apesar de toda sua agenda, conseguiu conceder momentos que expandiram a minha visão limitada do que este trabalho, novas tecnologias e o futuro poderiam ser. Agradeço a todos os integrantes do ICT-Lab e colegas de mestrado que ousaram dividir este fardo de “apenas estudar” e ajudar nas mais diversas demandas, em especial ao José Rodrigo, Victor Hugo, Rodrigo Hilário, Everton Morais, Luiz Felipe, Douglas Amante, Élcio Carlos, Fábio Carli, Epper Bonomo, Tibério Tavares, Karine da Costa, Diego Pivoto e Carlos Eugênio.

Agradeço também ao CRR por possibilitar a minha participação em outros projetos de alto nível que complementaram a minha trajetória. Ao grupo FIXP por todas as discussões que contribuíram para este trabalho. Ao INATEL por fornecer todos os recursos, pessoais e tecnológicos, com extremo zelo aos seus alunos, possuindo ambientes altamente qualificados de nível internacional. Agradeço também à CAPES, FAPESP e CNPq.

Por trás de acertos e erros (não foram poucos), fui evoluindo graças a tais interações nestes caminhos da vida. Gostaria de agradecer nominalmente a todos, mas este texto não comporta tal lista. Talvez por sermos pequenas manifestações do cosmos e dividirmos um mesmo céu, transformamos sonhos em realidade, guiados pela nossa imaginação, loucura e ousadia.

Sigamos em frente e adiante!

Thiago Bueno da Silva

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

5G	Fifth Generation
Ack	Acknowledgement
ACM	Association for Computing Machinery
AL	Abstraction Layer
API	Application Programming Interface
ARPANET	Advanced Research Projects Agency Network
AS	Autonomous System
BGP	Border Gateway Protocol
BID	Block Identifier
BMv2	Behavioral Model version 2
CL	Control Layer
ContentApp	Content Repository and Distribution Application
CP	Control Plane
CPU	Control Process Unit
CRS	Client Raw Socket
CS	Content Storage
DAF	Distributed Application Facility
DAG	Directed Acyclic Graph
DAP	Distributed Aplication Process
DHID	Destination Host Identifier
DIF	Distributed IPC Facility
DNS	Domain Name System
DP	Data Plane
DSL	Digital Subscriber Line
EBGP	External Border Gateway Protocol
ENDN	Enhanced NDN
EP	Egress Port
FCPH	FIXP Controller Packet Handler
FCSID	FIXP Client Raw Socket Identifier
FI	Future Internet
FIA	Future Internet Architecture
FIB	Forwarding Interest Base

FIBRE	Future Internet Brazilian Environment for Experimentation
FIC	Future Internet Controller
FIFu	Future Internet Fusion
FIXP	Future Internet Exchange Point
FIXPSW	Future Internet Exchange Point Switch
FRHS	FIXP Rule Handler Service
FSPH	FIXP Switch Packet Handler
GB	Gigabyte
GIRS	Generic Indirection Resolution System
gNMI	GRPC Network Management Interface
gNOI	GRPC Network Operations Interface
GNRS	Global Name Resolution Service
GSTAR	Generalized STorage-Aware Routing
GUID	Globally Unique IDentifier
GW	Gateway
HID	Host Identifier
HT	Hash Table
HTS	Hash Table Service
HTTP	Hypertext Transfer Protocol
HW	Hardware
IBGP	Internal Border Gateway Protocol
ICN	Information Centric Network
ID	Identifier
ID/Loc	Identifier/Locator
IDD	Inter-DIF Directory
IEEE	Institute of Electrical and Electronics Engineers
IngPort	Ingress Port
IoT	Internet of Everything
IoT	Internet of Things
IP	Internet Protocol
IPC	Inter-Process Communication
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IRATI	Investigating RINA as an Alternative to TCP/IP
IS-IS	Intermediate System to Intermediate System
ISP	Internet Service Provider
IXP	Internet Exchange Point
JSON	JavaScript Object Notation
LAN	Local Area Network
LoWPAN	Low-Power Wide-Area Network
MAC	Media Access Control
MAN	Metropolitan Area Network
MF	MobilityFirst

MsgID	Message Identifier
MTU	Maximum Transmission Unit
NA	Network Address
NB	Name-Binding
NDN	Named Data Network
NFV	Network Function Virtualization
NG	NovaGenesis
NGCA	NovaGenesis Control Agent
NLN	Natural Language Naming
NOS	Network Operating System
NRNCS	Name Resolution and Networking Cache System
NSF	National Science Foundation
ONF	Open Networking Foundation
ORBIT	Open-Access Research Testbed for Next-Generation Wireless Networks
OS	Operational System
OSI	Open Systems Interconnection Model
OSID	Operational System Identifier
OS	Open Shortest Path First
P4	Programming Protocol-Independent Packet Processors
PG	Proxy/Gateway
PGCS	Proxy/Gateway Controller Service
PID	Process Identifier
PISA	Protocol Independent Switch Architecture
PIT	Pending Interest Table
PL	Physical Layer
PoP	Point of Presence
PSIRP	Publish Subscribe Internet Routing Paradigm
PSS	Publish/Subscribe Service
pub-sub	Publish/subscribe
PURSUIT	Publish Subscribe Internet Technology
QoS	Quality of Service
RAM	Random Access Memory
RINA	Recursive InterNetwork Architecture
RT	Routing Table
RTT	Round Trip Time
SCN	Service-Centric Network
SDN	Software Defined Network
SeqID	Sequence Identifier
SHID	Source Host Identifier
SHM	Shared Memory
SLA	Service Level Agreement
SOA	Service-Oriented Architecture
SON	Self-Organizing Network

SRS	Server Raw Socket
SVN	Self-Verifying Name
SW	Switch
SWID	Switch Identifier
TCP/IP	Transmission Control Protocol/Internet Protocol
URI	Uniform Resource Identifier
VM	Virtual Machine
LAN	Wide Area Network

Resumo

Silva, T.B. da. NovaGenesis Control Agent for Future Internet Exchange Point [dissertação de mestrado]. Santa Rita do Sapucaí: Instituto Nacional de Telecomunicações; 2021.

Ao longo de sua existência, a Internet se tornou um serviço de utilidade pública e modificou a nossa forma de lazer, modelos de trabalho e as econômicas. No entanto, a arquitetura e a infraestrutura atuais de Internet precisam de melhorias para fomentarem aplicações futuras. Algumas limitações se apresentam na forma de suporte à identificação, mobilidade e segurança de entidades na rede. Estas e outras limitações impulsionam o desenvolvimento de Arquiteturas de Internet do Futuro, as quais podem propor projetos revolucionários para repensar a Internet do zero, e.g. NovaGenesis. Ao se considerar a complexidade e heterogeneidade atuais das aplicações, uma demanda pode surgir na forma de uma rede multi arquitetura, onde projetos disjuntos coexistem em uma mesma infraestrutura para atender requisitos de operação variados e Pontos de Troca de Tráfego serão cruciais para encaminhar pacotes de natureza distintas. O presente trabalho apresenta o Agente de Controle NovaGenesis para o *Future Internet eXchange Point* baseado na arquitetura P4. Esta proposta de dissertação avança o estado da arte ao desenvolver um controlador nativo e eficiente para a arquitetura NovaGenesis que gerencia um Plano de Dados dinamicamente. Conclui-se que o protótipo valida a solução *bottom-up* adotada em topologias variadas sem afetar significativamente o cenário de troca de conteúdos.

***Palavras-Chave*:** NovaGenesis, Future Internet eXchange Point, Next-Generation Software Defined Networks, Future Internet Architectures, P4.

Abstract

Silva, T.B. da. NovaGenesis Control Agent for Future Internet Exchange Point [MSc dissertation]. Santa Rita do Sapucaí: Instituto Nacional de Telecomunicações; 2021.

The Internet has become a public utility service and changed our leisure, work, and economy models. However, the current architecture and infrastructure of the Internet need improvement to support future applications. Some limitations concern the support for the identification, mobility, and security of entities in the network. These and other drawbacks have driven the development of Future Internet Architectures, where some revolutionary projects rethink the Internet from scratch, e.g. NovaGenesis. When considering the current complexity and heterogeneity of technologies, a future requirement can arise in the form of a multi-architectural network, where disjoint architectures coexist in the same infrastructure to fulfill varied operational requirements. Therefore, Internet Exchange Points will be crucial to forward packets of different nature. This work presents the NovaGenesis Control Agent for Future Internet eXchange Point based on the P4 architecture. This dissertation proposal advances the state-of-the-art by developing a native and efficient controller for the NovaGenesis architecture that dynamically manages a Data Plane. The proposed prototype validates the *bottom-up* solution adopted in varied topologies without significantly affecting the content exchange evaluation scenario.

***Keywords*:** NovaGenesis, Future Internet eXchange Point, Next-Generation Software Defined Networks, Future Internet Architectures, P4.

Chapter 1

Introduction

THIS chapter introduces concepts related to the current Internet and its evolvability. At first, it defines the Internet, its structure, and routing. Therefore, it highlights how heterogeneous devices with the most diverse access technologies benefits from this architecture.

In a second moment, it exposes some challenges of the current Internet model and how to address them in an evolutionary and revolutionary way. In addition, we discuss the future technology, how society can benefit from it, and some approaches to converge some technology trends. Paradoxically, contemporary society has the challenge to evolve the external technological environment and the Internet infrastructure at the same time.

Given this panorama, we can specify the objectives and the main general collaborations of the work presented in this dissertation. As proof of its relevance, it also lists the developed publications during the author's Masters' period. Finally, it displays the overall organization of this thesis.

1.1 Current Internet

This sections presents the current Internet model. At first, it defines what is the Internet. After this, it presents how the Internet can support so many heterogeneous devices, technologies, and systems globally. Following, it highlights the network organization and its routing scheme briefly.

1.1.1 After all, what is the Internet?

As ubiquitous as the Internet is today, it is essential to define this infrastructure beforehand. This system is much more than a simple network of interconnected computers, a system like the World Wide Web or an Internet provider that guarantees your connection to the global network. Hence, the Internet is a much more complex artifact that encompasses all the previous examples, in which heterogeneous devices, access methods, and systems

share consonant protocols to provide general services [1]. The Internet can encompass the most diverse devices such as personal computers, smartphones, or Internet of Things (IoT) sensors, wherein distributed networks present the most varied types and sizes of gadgets and topologies. These are then connected globally through wired or wireless systems, supported by Internet providers, mobile telephony, or satellites.

This whole access democratization effort started around 1970 through the Advanced Research Projects Agency Network (ARPANET) [1, 2]. During this project, the intent was to develop a way to connect research and scattered military centers to exchange data over long distances robustly and reliably. This project goal was to avoid possible attacks that would affect the entire communication network, compromising its operation by creating disconnected islands [1]. Over time, this infrastructure has expanded its reach to create Local Area Networks (LANs), Metropolitan Area Networks (MANs), and Wide Area Networks (LANs) with new technologies to achieve the goal of increasingly connecting more devices [1].

The Internet connection scope has increased only through evolutionary proposals, which shifted from a restricted communication network to a global infrastructure. As some outstanding examples, we can mention the Domain Name System (DNS) that allowed the translation of domains into IP addresses to locate resources conveniently. Furthermore, the official standardization of the TCP/IP protocol guided how systems should communicate coherently. Along the same lines, the evolution of analog telephony technology to Digital Subscriber Line (DSL) has optimized the telephony infrastructure to transmit digital broadband data, simultaneously with voice over the same telephone line [1, 3].

Through these and other evolutions, the Internet has become an ecosystem focused on the client/server model, where the network infrastructure is only responsible for forwarding data. In this way, applications are responsible for exercising intelligence and giving purpose to the data they transmit. This entire ecosystem enables the exchange of data from social media content to services provided by individuals, companies, and governments to a large mass in the most varied way possible.

1.1.2 One Internet to Rule Them All

As you probably can see in your life, the Internet establishes the same foundation to connect any entity in its network infrastructure. This means that the communication of any device happens based on a standardized reference model. In this way, you do not need to worry with how your gadget connects to the Internet. The Internet standards define a layered model to create a protocol stack wherein each layer has a set of well-defined (or something like a common consensus) functions to perform regardless of

the device technology or its Internet access. In sum, these layers support any application and adapt its service to be compatible with the Internet.

Over time, we had two major reference models for the Internet. At first, the Open Systems Interconnection Model (OSI) model presents 7 layers that encompass from the physical data transmission until the highest level where applications interact with human beings [1, 3]. Nonetheless, this is not the current Internet foundation. Concerning the most expressive Internet model nowadays, the Transmission Control Protocol/Internet Protocol (TCP/IP) is the adopted model, which normalizes the execution of globally distributed services across the many diverse systems through the network [1, 3]. Figure 1.1 illustrates the TCP/IP considering two hosts and two network elements, regardless of the Link Layer technology.

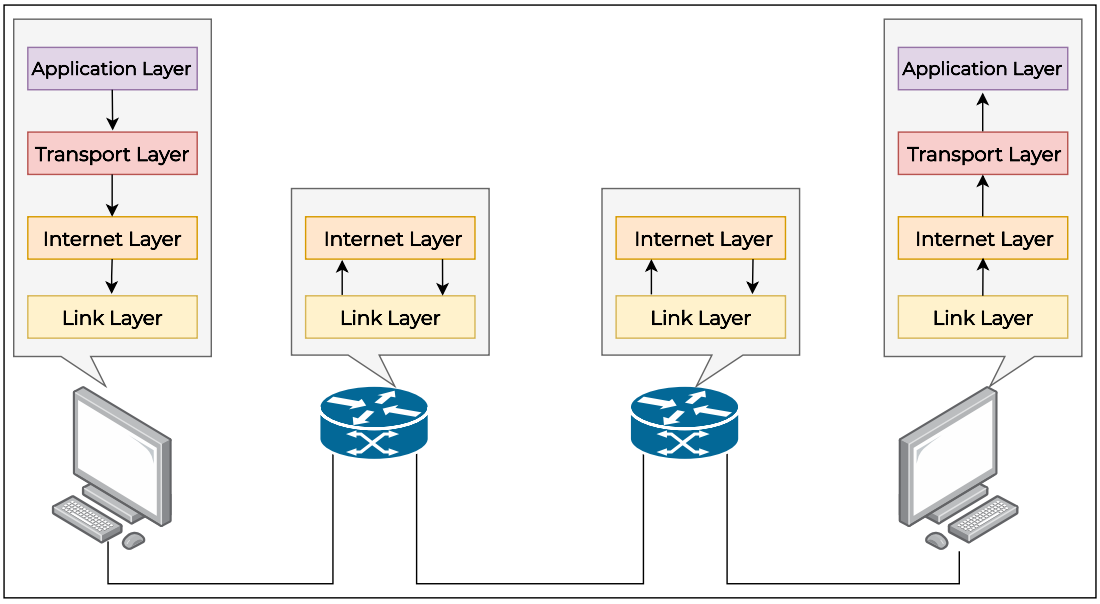

Figure 1.1: *TCP/IP Network Layer Model.*

Discussing this model is out of the scope of this document (and it is well-established in the literature). Nonetheless, notice that these layers fosters the data exchange through heterogeneous technologies. Usually, network elements present only two layers that provide the physical and logical network interconnection, forwarding and/or adapting the received data to another protocol, such as network gateways. Even though the network might be robust to adapt to a faulty link, it lacks the efficiency to ascertain the data at the network level, diminishing possible redundant data replication, and improving the security of the forwarded data.

1.1.3 Network Domains and Routing

Through the TCP/IP model and the DNS, the Internet ensures the localization of any connected entity regardless of its physical localization

through Internet Protocol (IP) addresses [1, 3]. An Internet client requires an Internet Service Provider (ISP) to access the Internet Infrastructure. Each ISP provides the best and most profitable manner to connect its clients through Point of Presences (PoPs), exploiting any access media possible, such as fiber, mobile, or satellite networks. Depending on the ISP scale, its network links or backbones can interconnect local, national, or international clients through several Autonomous Systems (ASs). This fact yields in MANs and LANs that may exploit different Link Layer technologies. In its turn, an isolated and individual ISP hardly can present a global reach. Therefore, the interconnection between ISPs happens through Internet Exchange Point (IXP) peering that vertically connects ISPs. Focusing on this vertical hierarchy, a Tier 1 ISP is an ISP that can reach a major set of ISP networks. Figure 1.2 tries to simplify this presentation.

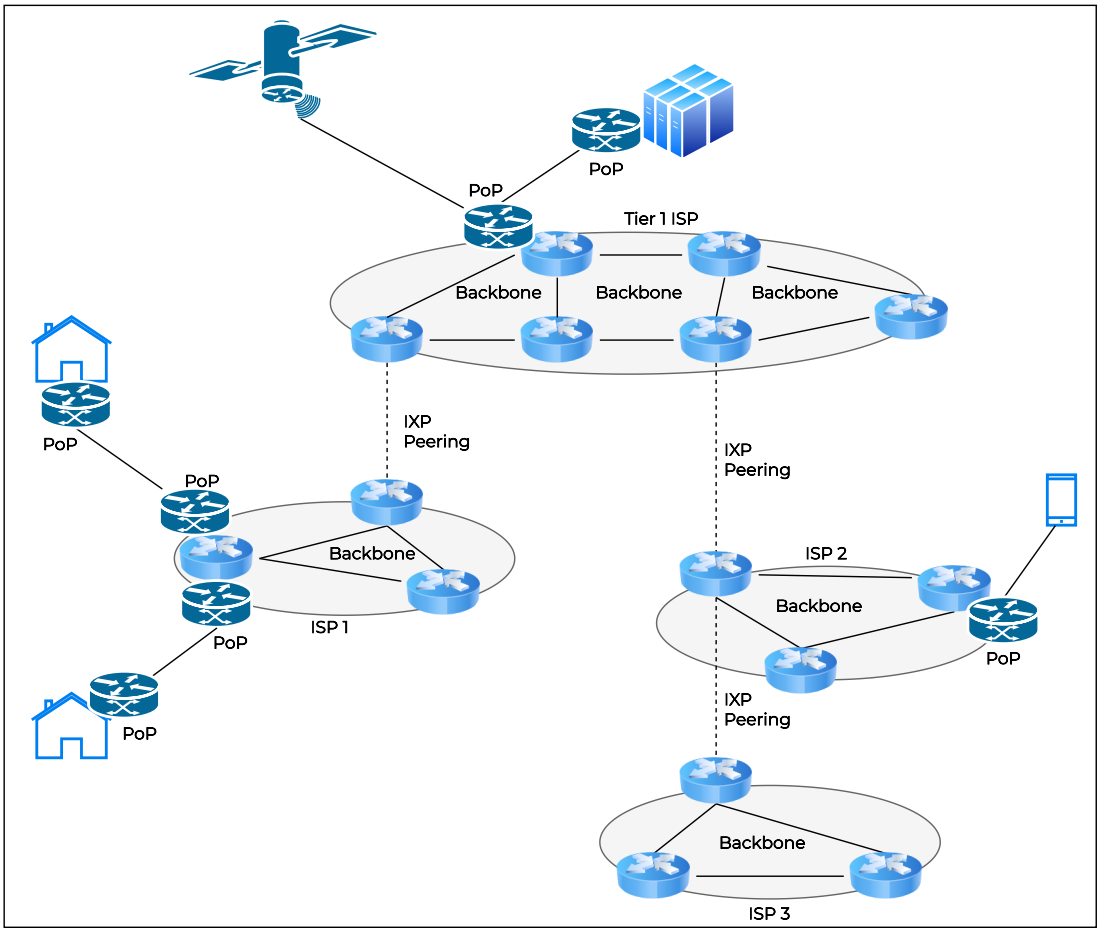

Figure 1.2: *Internet network infrastructure. Adapted from [1].*

Considering intra-domain communication, the routing scheme is straightforward once the data exchange happens within the same network domain and complies with the ISP policies. Nonetheless, this is the case that seldom happens [3]. In most cases, we are trying to reach content that is in a different domain. For example, someone could try to access the NASA

website from Brazil. In this way, the whole infrastructure has to adapt for inter-domain communication, exploiting the IXP peering to fulfill a request.

Therefore, inter-domain communication is even more complex because it involves divergent policies. ISPs hinder data that may compromise their business and countries differ on how to ensure data security. For example, Brazilian General Personal Data Protection Law is not the same as the North American. In summary, the inter-domain communication is convoluted and its discussion is out of the scope of this document.

Each IXP usually deploys a LAN of routers that enables the ISP peering, fulfilling the data exchange between ISP backbones [1]. In this way, the Internet routing scheme highly relies upon this ISP and IXP structure to forward a packet. During this exchange within and between ISPs and IXPs, several routing protocols and schemes take into account the most disparate network metrics. Once there is no consensus on the best way to forward a packet, the routing does not always yield in the shortest path possible.

1.1.4 Routing Protocols, Autonomous Systems, and Interconnection

The TCP/IP's Internet Layer ensures the packet transfer from the source until its desired destination. As a result, the network becomes an interface with the required set of hops through ISPs and IXPs routers to fulfill the end-to-end data delivery. As the physical network is unreliable, this layer must be aware beforehand of the physical topology and the links conditions. For example, network applications must carefully select the best routes to transmit a packet without overloading the ISP backbones, which can impact the overall infrastructure's Quality of Service (QoS). Considering the routing algorithms, they can be static or dynamic. The main difference is that the dynamic routing considers the network performance metrics to build a forwarding strategy. Therefore, it adapts its decision as the network changes in terms of topology and traffic. Some routing algorithms examples are the Directed Acyclic Graph (DAG), shortest path, flooding, distance-vector distance, and link-state. Meanwhile, the routing algorithms can consider some network performance as the number of hops, link delay, and even the charges that a IXP may ask to deliver some packet.

Commonly, the intra-AS routing exploits the Open Shortest Path First (OS) e Intermediate System to Intermediate System (IS-IS) [1]. Meanwhile, inter-AS communication must comply with the commercial arrangements between ISPs and countries. Hence, the local routing policies might be incomplete because a given ISP does not have the complete network topology awareness. In the current Internet, the inter-domain communication is generally the Border Gateway Protocol (BGP), which takes into account each AS political, security, and economic policies [1, 3].

The BGP protocol exploits IXP connections, wherein an ISP is the client of ISP provider. The ISP provider becomes responsible for delivering the received data traffic to the desired destination without receiving meaningful data that may compromise the ISP client. In short, the BGP applies the distance-vector algorithm, which considers the ISPs policies to forward data between IXPs. Each BGP router stores the used path that the packet requires through the ASs, where this router communicates through TCP/IP. The more specific details are out of the scope of this thesis, yet BGP presents other variations as Internal Border Gateway Protocol (IBGP) and External Border Gateway Protocol (EBGP).

1.1.5 Closing Remarks

This short presentation ascertains the complexity of the current Internet and its routing scheme. This infrastructure encompasses heterogeneous devices, protocols, and access methodologies that grant global connectivity through the TCP/IP cornerstone. The Internet architecture standardizes the way to address and locate any entity, as well as the way that they must exchange their data through ASs. Nonetheless, the IP routing differences are crucial for the overall network performance and QoS.

The current routing structure considers high-level policies to fulfill data delivery. In other words, this must comply with its ISP business, political, and economic policies to support the end-to-end connectivity. In this top-down routing, it may not exploit the best strategy to ensure the best QoS.

In addition, the legacy network infrastructure presents a Control and Data Plane that are vertically bound. This fact impacts the network evolvability, hindering new applications, protocols, and services, which could optimize the current scheme. Nonetheless, some efforts are dealing with the rigidity and ossification of the current Internet. These evolutionary and revolutionary proposals are the core theme of the next sub-chapter.

1.2 Future Internet

This section exposes some of the current Internet model challenges, the current two philosophies to evolve the Internet infrastructure, and some future scenarios where the Internet advancement is mandatory.

1.2.1 Current Pitfalls

In its nearly 60 years, the Internet infrastructure has been a success since its inception. Starting from a network that connected academic and military centers, ARPANET has reached unimaginable dimensions from what its designers considered. Who could dream that this invention would change

and shape society so profoundly in the 1960s/70s? As John Day, one of the ARPANET pioneers, states in [2], clear limitations have marked the scope of the project in terms of hardware, connections, and applications. Consequently, many shortcuts, mistakes, and successes taken at that time aimed at validating a prototype to foster further research and, thus, planned organic development. Nevertheless, the proposal was so successful at first [4] that it soon conflicted with commercial and government interests, which derailed drastic changes in architectural design.

One assumption relates to its host-centric design [5], which has made the Internet only a way of delivering data. During Internet's conception, the hardware was restricted. For instance, computers had similar specifications as the TX-2 with 320KB of memory, which has simulated the theory of packet networks [6]. Hence, the network model suffered a segmentation into layers wherein the upper layers perform more intelligent functions than the lower ones that handle data transfer, flow control, and error.

However, this no longer reflects the current reality, as technological development followed an advance based on Moore's Law or, perhaps, the Law of Accelerated Returns. There is enough technological abundance to fuel the most revolutionary ideas. For example, the Internet is seen as one of the drivers of what we are discussing today as Smart Cities, Industry 4.0 and, eHealth, increasingly fostering the convergence of the digital and physical world. Given this scenario, new questions arise regarding the feasibility of continuing to use the current Internet architecture in the future [7]. There is a list below containing five weaknesses of the current Internet.

Naming, Addressing, and Name Resolution

From the start, naming refers to the identification of an object in a given scope. Addressing, on the other hand, represents a way of locating it. Thus, a name is not necessarily an address [2]. Generally speaking, addresses are ways of assigning references to connection points in a topology, facilitating their discovery. Given this more ethereal aspect, naming and addressing concepts still confuse several people and they impact on establishing an effective and efficient project structure. As stated by John Day, this aspect is directly crucial to the success of a network proposal [2]. Returning to ARPANET, the scenario by then was simpler to establish a relationship between naming and addressing. This fact was due to technological and geographical restrictions with homogeneous connection points regarding hardware, quantity, and connectivity technologies. Thus, the adopted system was to enumerate networks and the equipment interfaces that constituted them [2].

Nonetheless, the current scenario presents heterogeneous communication devices, such as computers, smartphones and sensors, and physical means, like wired, mobile and satellite communication networks. In addi-

tion to this, current networks are complex, interconnecting countries by the most diverse providers and network paths. As its coverage expanded, the Internet has received mechanisms to facilitate and guarantee the insertion of new connections, the routing of packets, and the overall name resolution on the network [4].

Mobility e Accountability

In the early days of the Internet, hosts were geographically fixed and static. Hence, the researchers have discarded a scenario where mobile devices exist [2]. Consequently, today's infrastructure has been shaped to provide addresses that would virtually represent the same hosts over time. However, the Internet also started to connect dynamically mobile objects in space due to the massification of smartphones and other mobile proposals in recent years, such as autonomous cars, drones and sensors.

Currently, hosts experience address changes when switching across network domains. Even though the application addresses are location independent, the devices' physical addresses alternate during this intermittent exchange, causing route changes to ensure data delivery [8]. Even if a name doesn't change (what we want), its address (where should we look) and the required route (how can we reach it) are dynamic due to intrinsic characteristics of the TCP/IP stack.

This fact undermines the devices' accountability on the Internet since their addresses change over time. For example, when we request content in the city of Santa Rita do Sapucaí, we have an IP address associated with our device from a provider in that given area. When we move this same device to another city, like São Paulo, we will have another IP address available.

Now consider a malicious device on the network. This feature of disassociating addresses and non-perpetual naming and addresses to devices can help other possible attacks. In addition, some networking techniques encourage the concealment of the attackers' real addresses, hindering their traceability and restricting possible sanctions to prevent new attacks. Associated with this, we can have several identical names in the same domain. This fact increases the network's ability to resolve names to reach the proper destination, wakening the security of applications [2].

Content Replication

In the same way that there is redundancy associated with names, today's Internet presents a high number of replied content around the network. On the one hand, this is important to ensure continuous access to given content, improving the quality of service seen by the consumer. As the Internet's framework does not require a direct connection to the destination on an ongoing basis, the network presents copies of content spread across

different servers or network caches. However, can we say that this content replication in the network is optimized?

In an ever increasingly complex world, this question does not only impact storage capacity or network performance. Environmental impacts must also be considered in this equation, such as energy expenditure, carbon dioxide emissions, and solid waste that keeps this storage infrastructure always available and updated.

1.2.2 Internet Evolution

Considering these and other various weaknesses of the Internet, many researchers have dedicated themselves to researching ways to evolve and prepare an ecosystem capable of supporting the future's trends. On the one hand, proponents of the current architecture advocate incremental improvements that update such infrastructure as new problems become unbearable. On the other, revolutionaries develop proposals for network architectures from scratch. Through these clean-slate architectures, developers choose contemporary concepts to foster the development of their projects [7].

Through evolutionary projects, we can mention the evolution that the Internet architecture underwent when going from a geographically restricted communication network to a global one [9]. Under this new scale, the architecture has received new features for delivering packages, such as support for mobility, multicast, and anycast, enabling distributed networks [4]. Consequently, these improvements have conflicted with the host-centric nature of the network, which assumes that hosts are immovables and that there is only one destination per request. Thus, DNS emerged to resolve names, attuning the new features with the architecture. Another example lies in the protocol update from IPv4 to IPv6 due to the scarcity of available addresses to support new applications and devices in the contemporary world [8, 10]. Again, new challenges arise in terms of the scale of such challenging transition, routing scalability, security, mobility, and quality of service [8, 11, 12].

Revolutionaries, on the other hand, propose disruptive Future Internet Architecture (FIA) as clean-slate designs. In this research area, the design incorporates paradigms that may reshape our concept of the Internet from the physical layer to its application layer. Some examples break away from the host-centric paradigm to focus on content-, mobility-, or service-centric [4]. Furthermore, they may exploit contemporary concepts such as Software Defined Network (SDN), resource virtualization, and other innovative features in their conception [7]. Ergo, someone can expect that these projects are more suitable for the future. After all, they take advantage of the best that current technology has to offer while learning from the current Internet takeaways. As can be seen, it is exceptionally troublesome

to generalize all the FIA with their particular ingredients under a single umbrella that fits all their particularities.

1.2.3 Fortune Telling and Closing Remarks

Promising new technologies have come to revolutionize the world. Among these, we can cite the Internet of Things [13] as one of the main drivers of the convergence of the physical and digital world, as it is one of the pillars of trends in Autonomous Cars, Industry 4.0 [14] and Medicine 4.0 [15] . Other technologies support this scenario, such as Big Data, Artificial Intelligence, and Cloud Computing. As a result, some expect over 20 billion IP connections on the Internet by 2023 [16]. Probably because they may share today's Internet as a way to interconnect everything, we're back to questioning whether this infrastructure will harmoniously accommodate or limit the future's potential.

As much as we have revolutionary proposals of Internet architectures, we must also consider that perhaps the complete replacement of the current architecture is not feasible [11]. Many investments have gotten us where we are. Besides, an entire legacy infrastructure keeps the world connected. Furthermore, we are used to such technology and have known the current system's robustness for years. Thus and for other reasons, a doubt arises regarding the assumption that a single FIA may completely replace the current Internet someday.

Despite this, a path to the coexistence of multiple network architectures has been drawn through virtualization technologies, SDN, and post Fifth Generation (5G) mobile networks lately [17]. In this heterogeneous ecosystem, diverse architectures provide their best service by network slicing, ensuring distinct operations under disjoint and varied requirements and QoS [18, 19]. All this in an autonomous way, orchestrated in an imperceptible way to the end-user or application, while exploring the most diverse communication technologies [7, 20].

1.3 Technology Evolution and a Possible Future

In this atypical current pandemic scenario, the world has relied essentially on the Internet infrastructure to keep up with the "old world." Through this global network, society has adapted in the best way possible. For example, videoconferences have supported from work to education meetings. Suddenly, some tendencies that would take years to be studied and implemented have become our reality in a short period to guarantee the established deadlines, adapting some tasks at a pace never seen before.

Some surveys show an increase in the Internet demand from 40% to 50%, a 16TB/s traffic peak in 2021 [21], and a proportional rise of complaints related to the QoS provided. These facts are just a reflection of the sudden reality, where we were able to adapt our social and professional life through the Internet (e.g. education, leisure, and/or e-commerce/delivery). However, what can we expect from the future?

Looking to the future, we can find academic and commercial proposals to make environments "smart". As an example, we can mention the trends of Smart Cities, Industry 4.0, and eHealth. These and other Smart-* aim to insert "intelligent" devices into our daily lives to facilitate contemporary life. These systems are proposals distributed with a range of actuators and sensors that allow automatic/autonomous decision-making. In whichever trend, communication networks connect these devices in a myriad of ways. Even more, we are talking about the Internet as the way to exchange data in most of the cases, e.g. from a set IoTs devices or even the Internet of Everything (IoT). Through all this kaleidoscope of globally interconnected devices, technologies, and systems, something even more disruptive will arise: the metaverse. This new frontier considers the total convergence of the physical and the digital world, such as depicted in the Matrix, Ready Player 1, or Tron movies. In sum, real entities have digital twins, which are the virtual counterparts of reality [22].

Comparing an atypical situation to such disruptive proposals, what can we expect from the current infrastructure of the Internet in this context? Still, to what extent can evolutionary measures truly optimize an infrastructure that bases on assumptions that reflect a limited hardware reality at its conception time? Even more, how will we enable the evolution of the Internet that is harmonious with these disruptive visions? After all, will those proposals perform their best possible service, or should they conform to what they can obtain if we continue with the same architecture?

Therefore, we return to the topic of proposals for FIAs, SDNs, and other technologies that are efforts to not only optimize the network infrastructure but also to promote better support for this future. FIAs focus on the evolution of the Internet architecture itself, SDNs foster the better agility and flexibility of the infrastructure of networks and future mobile networks (post 5G) aim to encompass more areas, like the countryside. Through these and other disruptive concepts, the future presents trends like network slicing [19], wherein the network can be divided into several slices to prioritize the transmitted data, and Intent-Based Networks, which can be autonomous networks that are orchestrated through a high-level language and Application Programming Interfaces (APIs). Nonetheless, there are still several discussions on how this environment will be autonomously composed and orchestrated.

Finally, what can we expect from the Internet of the future? Can we

imagine that an FIA will be able to replace the entire TCP/IP model someday? We can argue that this proves unfeasible for financial, political, and many other reasons that permeate an infrastructure of such global scale. Thus, a more coherent strategy is to focus on proposals that enable the democratization of the Internet to take advantage of multi-architectures. As it is presented on Chapter 3, the current FIAs are disjointed, advocating for disparate concepts. Therefore is an open challenge to design an environment that synergetically aggregates those revolutionary proposals.

1.4 Objective

As a consequence, software-based IXPs can become crucial and promote this multi-architectural network. In these future environments, control agents dynamically leverage programmable network elements. These intelligent controllers draw specific strategies to fulfill their architecture's data exchange, adapting their knowledge to meet the most diverse application requirements at the Data Plane. Therefore, this multi-architectural network connects clients of different and specialized architectures for unique services by exploring software-based techniques, specialized controllers, and programmable devices. In summary, each architecture performs its best service harmonically with other proposals through programmable Future Internet Exchange Points (FIXPs) and by their control agents.

Given all that has been exposed in this chapter so far, the primary objective of this work is to conceive a control agent for the revolutionary FIA called NovaGenesis (NG). This architecture presents several features that are common to other revolutionary proposals found in the literature. Briefly, we can cite its name resolution, flat identifiers, and support for mobility. Moreover, we seek to design a native NG SDN controller for the FIXP project, i.e. an SDN exchange point focused on multi-architectures.

This approach is new in the literature. So far, it seems that any related work has covered the conception of a native SDN controller for FIA. Every proposal from Chapter 3 combines FIAs and SDN, yet all of them focuses on the Data Plane programmability. None presents a native controller.

We also present a programmable NG Programming Protocol-Independent Packet Processors (P4)-based Data Plane. Through this, we can create a methodology that supports NG novelties.

Finally, a scientific methodology validates the NG control agent prototype for FIXP. For this, we explore the NG Content and Distribution Application scenario, measuring the NG, the FIXP, and the host performances.

1.5 Main Contributions

In brief, the main contributions of this work are the following:

1. Design of a native NG control agent for FIXP.
2. Design of a programmable Data Plane for NG, which supports all of its novelties.
3. Propose an evaluation methodology that covers the NG, FIXP, and its host machine performance.

1.6 Publications, Projects, and Scientific Divulgation

During the development of this work, there were some opportunities to present NG, FIA, and other subjects. These opportunities ranges from scientific publications, collaboration in other projects, and some diverse presentations. From these, two conference papers were awarded as best paper of their workshops. Meanwhile, the author has received one award as feature student co-author from the 2016-2020 quadrennium at Inatel. The projects were supported by the partnership between ICT Lab, Inatel Competence Center, and Inatel. From these, the Smart Garden project received media coverage from national television stations.

The complete masters' journey is summarized in the following lists:

1.6.1 Publications

(i) T. B. da Silva, R. S. Chaib, A. C. S., R. d. R. Righi and A. M. Alberti, "Towards Future Internet of Things Experimentation and Evaluation," in IEEE Internet of Things Journal, doi: 10.1109/JIOT.2021.3114540.

(ii) SILVA, Thiago B. da; GAVAZZA, José A.T.; VERDI, Fábio L.; SURUAGY, José A.; MELO, Juliano Coelho; SILVA, Flávio de O.; ALBERTI, Antonio M. Plano de Controle da Arquitetura NovaGenesis para um Ponto de Interconexão de Tráfego Multi-Arquitetura. In: WORKSHOP DE PESQUISA EXPERIMENTAL DA INTERNET DO FUTURO (WPEIF), 12., 2021, Evento Online. **Anais do XI Workshop de Pesquisa Experimental da Internet do Futuro**. Porto Alegre: Sociedade Brasileira de Computação, 2021 . p. 31-36. ISSN 2595-2692.

(iii) SANTOS, Jose R.; REZENDE, Tiberio T.; SILVA, Thiago B. da; ROSARIO, Elcio C.; ALBERTI, Antonio M. Proposta de Arquitetura para Distribui- çao de Conteudos Nomeados em NovaGenesis com P4. **Anais do XI Workshop de Pesquisa Experimental da Internet do Futuro**, Rio de Janeiro, p. (32-37), 2020. Available in: https://sol.sbc.org.br/index.php/wpeif/article/view/12472.

(iv) LUGLI, Alexandre B.; ALBERTI, Antônio M.; PIMENTA, Tales C.; SILVA, Thiago B. da. A Study on Some Industry 4.0 Key Technologies. ICIC Express Letters, p. (713–720), Volume 11, Issue 8, Aug. 2020. Available in: http://www.icicelb.org/ellb/contents/2020/8/elb-11-08-01.pdf.

(v) SILVA, Thiago B. da; MORAIS, Everton S. de; ALMEIDA, Luiz F. F. de; RIGHI, Rodrigo da R.; ALBERTI, Antônio M. Blockchain and Industry 4.0: Overview, Convergence, and Analysis. In: Rosa Righi R., Alberti A., Singh M. (eds) Blockchain Technology for Industry 4.0. Blockchain Technologies. Springer, Singapore. Available in: https://doi.org/10.1007/978-981-15-1137-0_2.

(vi) GAVAZZA, José A.T.; MELO, Juliano Coelho; SILVA, Thiago Bueno da; ALBERTI, Antônio Marcos; ROSA, Pedro F.; SILVA, Flávio de O.; VERDI, Fábio L.; SURUAGY, José A. Future Internet Exchange Point (FIXP): Enabling Future Internet Architectures Interconnection. In: Barolli L., Amato F., Moscato F., Enokido T., Takizawa M. (eds) Advanced Information Networking and Applications. AINA 2020. Advances in Intelligent Systems and Computing, vol 1151. Springer, Cham. https://doi.org/10.1007/978-3-030-44041-1_62.

(vii) ROSÁRIO, Élcio C.; D'ÁVILA, Victor H., SILVA, Thiago B. da; ALBERTI, Antônio M. A Docker-Based Platform for Future Internet Experimentation: Test-ing NovaGenesis Name Resolution.IEEE Latin-American Confer-ence on Communications (LATINCOM), Salvador, p. (1-5), 2019. Available in: https://ieeexplore.ieee.org/document/8937898.

(viii) SILVA, Thiago B. da; ALBERTI, Antônio M., LUGLI, Alexandre B. Introspecções da Indústria 4.0: Um estudo sobre a convergência de tecnologias disruptivas e o seu impacto na indústria do futuro. In: Congresso Brasileiro de Instrumentação, Sistemas e Automação. Campinas, São Paulo, Brazil, 2019. Available in: https://proceedings.science/cobisa-2019/papers/introspeccoes-da-industria-4-0–um-estudo-sobre-a-convergencia-de-tecnologias-disruptivas-e-o-seu-impacto-na-industria-d.

1.6.2 Submitted Manuscripts

(i) T. B. da Silva, L. Verdi, Fabio, Coelho Gonçalves de Melo, Juliano, Augusto Suruagy, José, Silva, Flavio, and A. M. Alberti, "A NovaGenesis enabled Future Internet eXchange Point," in IEEE Network Magazine.

1.6.3 Awards

(i) Best article award at the XII Experimental Research Workshop on the Internet of the Future (WPEIF 2021). Authors: Thiago Bueno da Silva, Jose Gavazza, Fabio Verdi, José Augusto Suruagy Monteiro, Juliano

Melo, Flavio Silva. Work: "Plano de Controle da Arquitetura NovaGenesis para um Ponto de Interconexão de Tráfego Multi-Arquitetura".

(ii) Feature co-author student of outstanding bibliographic and technical productions. Quadrennium 2016-2020, Inatel.

(iii) Best article award at the XI Experimental Research Workshop on the Internet of the Future (WPEIF 2020). Authors: José Rodrigo Santos, Tibério Tavares Rezende, Thiago Bueno da Silva, Elcio Carlos Rosário, Antônio Marcos Alberti. Work: "Proposta de Arquitetura para Distribuição de Conteúdos Nomeados em NovaGenesis com P4".

1.6.4 Scientific Presententations

(i) SILVA, Thiago B. da; ALBERTI, Antônio M.. Redes Programáveis de Próxima Geração: Habilitando Múltiplas Internets e Suportando a Proposta NovaGenesis. TDC Transformation Conference, 2021.

(ii) PADINHA, Mariana; SILVA, Thiago B. da; ALBERTI, Antônio M.. Papo Ciência e Tecnologia - ICT Lab e RenaSCidade: Qual é o futuro da Internet e das redes programáveis? 2021. Available in: .

(iii) SILVA, Thiago B. da; ALBERTI, Antônio M., LUGLI, Alexandre B. Introspecções da Indústria 4.0: Um estudo sobre a convergência de tecnologias disruptivas e o seu impacto na indústria do futuro. In: Congresso Brasileiro de Instrumentação, Sistemas e Automação. Campinas, São Paulo, Brazil, 2019.

(iv) Globo Rural. Extra Globo Rural: internet ampla pode levar mais cidadania ao campo. 2020. Available in: https://eubrasilcloudforum.eu/en/service-demo/demonstration-future-internet-things-novagenesis.

(v) RNP. Projeto usa tecnologia 5G para conectar escola rural em Minas Gerais. 2019. Available in: .

(vi) Inatel. Inatel recebe pesquisadores do Brasil e do exterior para reunião do projeto 5G Range. 2019. Available in: .

1.7 Thesis Organization

This dissertation has six more chapters. Chapter 2 addresses the relevant works to understand the proposal of this document, these being trends, technologies, and architectures of SDN, P4, FIXP, and NG. After this theoretical presentation, Chapter 3 presents a literary review of the current state of the art of FIA and SDN, focusing on proposals with P4. Given the theoretical basis and what is being proposed by the academic community, Chapter 4 presents the proposal of a NovaGenesis Control Agent with Future Internet eXchange Point. Then, Chapter 5 presents the tools and methodology to validate the proposal of this work. In turn, Chapter 6

presents the results obtained through the adopted methodology and considers what was validated. Finally, Chapter 7 presents the conclusions of this work, together with possible future works to advance this proposal even further and the lessons learned.

Chapter 2

Background

THIS chapter presents the required background to support this Masters' proposal. Therefore, it firstly focuses on the SDN technology and its benefits. After this, it explores a SDN architecture called P4, which models and enables programmable forwarding network devices. Following, it introduces the FIXP project, introducing this multi-architecture SDN environment. At last, it shows the NG architecture, covering its main concepts, benefits, and communication scheme.

2.1 Software Defined Network

This section explores the SDN technology, exposing what has led to its development, its concepts, and some of its advantages.

2.1.1 Network Infrastructure Ossification

Along to the Internet's structural rigidity, the network devices that operate in Internet's lower layers are also ossified. Historically, switching devices and routers are inflexible, virtually immutable due to the intrinsic traits associated with their chips [23]. Hence, the conventional network hardware is a vertical device, which performs functions from the data and the control planes. Furthermore, any innovation must come firstly from their developers and manufacturers, following a top-down model. Consequently, this fact limits possible improvements during the device's lifetime (excepting firmware updates). Moreover, it also hinders the experimentation with new communication protocols and architectures.

Besides, we can mention the complexity of managing the current IP networks as well. This feature goes beyond the aspect of controlling a highly decentralized infrastructure [11]. Despite having been an assumption that has increased the robustness of the entire network to date, the contemporary heterogeneity of network solutions hampers the network op-

erators [11]. For example, these operators need to individually change the operating rules of each device present in a network scope to change a single network policy. Moreover, this adjustment must follow the vendor-specific standard commands [11]. When we consider a geographically ample topology, this impacts the scalability and agility of updating a network domain because of its various distinct equipment.

Grasping the future, there is a considerable challenge for developers and network operators. How shall we move towards a society relying notably on a virtual world when we have not only a limited Internet architecture but also an ossified infrastructure at hand? By converging the physical and digital worlds through Smart Cities [24], Industry 4.0 [25], and other proposals [26, 27], we need to consider not only new personal devices, but also sensors, applications, and other entities that represent animate or inanimate physical objects connected globally [28, 29]. Consequently, devices and network addresses will increase exponentially, as well as the network traffic associated with the actual data and the management data to ensure the smooth operation of the entire network infrastructure [7].

2.1.2 Network Infrastructure Evolution

Addressing these challenges, SDN proposes the vertical separation of the control and data planes by employing techniques and abstractions to virtualize the network [11]. Therefore, the network infrastructure becomes divided into flexible, programmable, and customized forwarding and controlling devices. This means that SDN programmable forwarding devices can be used for different network applications by only updating its source code. At the same time, controllers centralize the required intelligence to manage several programmable forwarding devices [11, 30]. Consequently, centralized controllers optimizes the network operation, increasing the agility, granularity, and evolvability while reducing overall costs [4]. OpenFlow protocol is one of the first SDN technologies released in 2011 [31].

Figure 2.1 illustrates the generic architecture of an SDN topology. In this, there are 3 specific planes with equipment spread in each layer and standardized interfaces, interconnecting each sphere. The list below characterizes the most common SDN concepts [11, 30, 31]:

1. Data Plane: Features programmable forwarding devices that are physically and/or virtually interconnected to exchange data.
2. SDN Forwarding Devices: Programmable forwarding devices in the Data Plane. These devices can be either physical hardware or virtualized software. In addition, these devices are modeled by the basic operations that a network equipment has to perform. Furthermore, these devices are dynamic, configured with new network rules in real-time during their activity by the SDN controllers.

3. Open Southbound API: Interconnects the Control and Data Planes, standardizing the communication between data devices and controllers.
4. Control Plane: Presents the SDN controllers spread geographically and interconnected horizontally, exchanging data between controllers, or vertically, interacting with the Management and Data Planes.
5. SDN Controllers: Manage the SDN forwarding devices, dynamically updating their network rules. Furthermore, SDN controllers cooperate with network applications in the Management Plane to optimize their intelligence and management of the entire topology.
6. Open Northbound API: Establishes the interconnection between controllers and network applications.
7. Management Plane: Presents network applications to manage network policies and update SDN controllers in real-time.
8. Network Applications: Leverages the control and logical operation of the network. Through the northbound interface API, network applications can update the network policies of each SDN controller, e.g. routing, load balancing, and monitoring definitions.

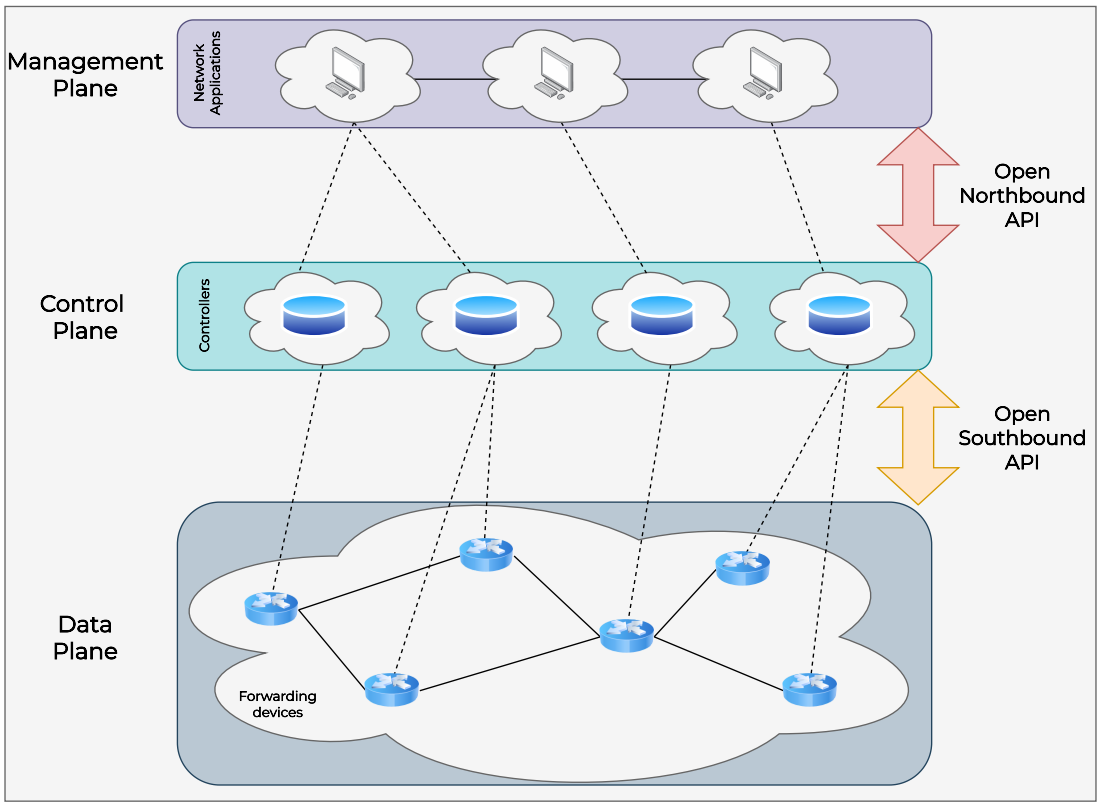

Figure 2.1: *Generic SDN architecture. Adapted from [11].*

2.1.3 Fortune Telling and Closing Remarks

SDN technology disrupts the network field by separating the Data and Control Planes, dealing with the ossification of network equipment. Programmable forwarding devices are modeled for network applications, ensuring the independent network evolution while enabling the experimentation with new protocols and communication technologies [23, 32]. Such flexibility facilitates the granular control of the network, modeling it by abstractions that help the management of each equipment for a homogeneous operation of the entire infrastructure [11].

This technology has attracted attention from both academically and commercially [33]. Despite presenting expensive hardware, these devices are becoming more affordable as it becomes more popular and acceptable [11]. In such a way, it can be noticed its adoption in infrastructures of the current Internet, in data-center networks, and industrial networks [31].

Concerning future scenarios, the SDN technology guarantees not only the optimization of data traffic, which tends to increase exponentially but also encourages the development of new technologies. Hence, both the FIA and communication technology communities benefit as they can now mold equipment capable of operating under their requirements. Furthermore, centralized controllers or Network Operating System (NOS) oversee the physical topology, orchestrating both the dynamic reconfiguration and enforcing the network policies [11]. Consequently, both the network performance and the use of resources are optimized, which are configured under metrics for balancing traffic according to its operation, e.g. aiming at the best possible QoS. Furthermore, it grants greater security to the entire infrastructure by enabling granular control of devices, identifying possible attacks, and countering attacks in real-time.

2.2 P4

This sub-chapter presents a brief P4 introduction, covering its highlights, the P4's pipeline, language, compilers, and some additional remarks.

2.2.1 P4 highlights

Bosshart et al. propose P4 as a path of how OpenFlow should evolve, taking into account that the increased complexity of OpenFlow's fields did not add any flexibility to add new protocol headers [30]. Ergo, P4 has posed as a feasible alternative, advocating for a remarkable level of abstraction, customizability, and programmability of network equipment [23, 33, 34]. In this view, P4 is similar to what C/C++ languages are for embedded devices, where the P4 language abstracts the target device, which can be a physical

or software switch, network interface card, router, or network appliance [35]. Since 2014, P4 Language Consortium and Open Network Foundation develop and support this proposal [36], focusing on advancing the language, API, architecture, and education of this framework.

P4 also establishes a Southbound API [30], standardizing how to manage P4-enabled devices on the fly. For example, this aspect enables the modification of the P4 devices' routing tables, adding new forwarding rules, or retrieving some feedback from their status. All of this has arisen over the years with improvements in the P4 framework or introducing a new switch Operational Systems (OSs) that cooperates with P4 like Stratum [37]. Nonetheless, every new P4 update follows three major goals that are [30]:

- **Reconfigurability**: P4 shall enable changes on how the forwarding devices process packets during their execution.
- **Protocol Independence**: P4 shall support any network protocol.
- **Target Independence**: P4 shall program the packet processing behavior of any hardware, without knowing the details of target devices.

2.2.2 Protocol Independent Switch Architecture

P4 follows the Protocol Independent Switch Architecture (PISA) model to establish a forwarding device behavior. This standard sets up how P4 enables the Data Plane (DP) customization of a target device [35], in terms of how the hardware or software must process the packets, the Control Plane (CP) and Data Plane (DP) intercommunication, as well as its limitations. One of P4 restrictions regards the inability of Control Plane (CP) modeling behavior. In other words, P4 solely models the DP devices, whereas a Southbound API called P4 Runtime must ensure the dynamical CP management. Figure 2.2 illustrates the PISA model. This has been adapted from [30].

The PISA ensures target and protocol independence once it designs P4 behavior as a finite state machine [30]. Following the packet input flow, packets arrive at a given Ingress Port. Next, the Parser handles the arriving packets, which translates bytes into the modeled protocol headers from the P4 program model. After this, a set of serial or parallel matches/actions fulfill a determined operation. For example, this stage may modify the incoming data to adapt it to a given different output protocol or replicate the packet into several Egress Ports for a multicast rule. Likewise, the Egress Pipeline presents another set of match/action that may modify the received packet. After this, it assembles the data into a packet to forward it to the proper Egress Port(s). Along to this, P4 Runtime can modify the P4 target Ingress and Egress Pipelines through delineated primitives received.

Notice that the Buffer and Output blocks can implement queues to control the data traffic. Along through these blocks, additional metadata are

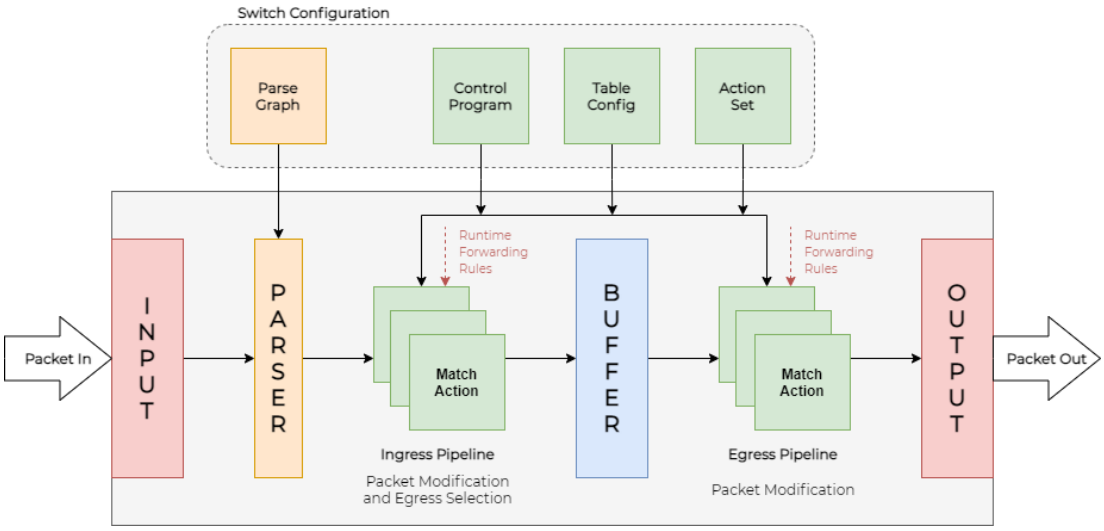

Figure 2.2: *P4 abstract forwarding model. Adapted from [29].*

bound to the packet, which retrieves information regarding the packet's Ingress Port or timestamp for packet scheduling [30].

2.2.3 P4 Language

Following the PISA organization, P4 language models the packet processing behavior as a finite state machine. Each state represents a block of the Input, Parser, Ingress, Egress, and Deparser (Output). In addition, a developer can implement some Checksum verification and add some custom definitions that describes the P4 variables, registers, structs, and headers. Therefore, the P4 program present some key components that represents these states/blocks, as well as what triggers the transition from a state to another, like from the Parser to the Ingress Pipeline.

The full P4 program and syntax description is out of the scope of this dissertation (you can check the P4 16 specification in [35]). Nonetheless, the following list summarizes the main concepts of a P4 program [30,33,35]:

- **Header**: This structure describes the set of fields that comprises a protocol. In other words, the developer must know in depth what each group of bytes means to model a protocol header. For example, an Ethernet P4 header must present the definition of how many bytes and their order to construct the Destination Media Access Control (MAC), the Source MAC, the Ethertype, and the Ethernet payload.
- **Parsers**: This establishes the procedures to parse an incoming packet, where the header fields receive their specified values. For example, P4 parses an Ethernet packet following the protocol order, i.e. the first 6 bytes represents the Destination MAC, the following 6 bytes defines the Source MAC, and so on. Considering protocols that exploits other standard as a encapsulating mean, such as Low-Power Wide-Area Net-

works (LoWPANs) over Ethernet, the developer can add intermediary parsing from the outer stacks to the inner ones. For example, the first parsing can consider the IEEE802.11 structure as the first parser and, then, follow to the specific LoWPAN fields in a second parser.

- **Tables**: Match and Action Tables outline the keys and actions for packet processing. This structure links header variables to a set of keys to define processing actions for a protocol. For example, an IP table may resolve the IP destination address as a key to forward an incoming IP packet. This table also lists the actions to forward, drop, or a the default action if the key does not match any knowledge.
- **Actions**: Actions establishes processing functions. For example, an action can receive parameters, modify fields in the received packet, and set the egress port. There are several P4 primitives to allow a miscellaneous set of packet operation and forwarding.

It is important to note that the P4 structure ensures a high level of abstractions, fostering compatibility with vast targets, regardless of their physical or virtual nature [23, 33]. This pipeline enables compilers to generate files to be executed on actual targets. The compiler's takes as an input the P4 program and outputs a C source code, a JSON, binary executable, or any other chosen format. This output embodies the P4 target behavior, a programmable parser that process packets based on customized settings.

2.2.4 Control Plane, P4Runtime, and P4 Primitives

As can be seen so far, P4 is a framework to program data plane devices without deep knowledge of the actual targets. In broad, P4 models the packet processing of data plane devices in terms of protocols headers, which parses packets into meaningful P4 variables. Moreover, P4 tables that lists all the actions for a given architecture, as well as the relevant keys to select and execute each action. Then, P4 actions may modify the incoming packets and generate egress packets properly. Notwithstanding, one question can come to mind... How can one supply instructions for the P4 on the fly? How a P4 control plane can manage its data plane?

Nowadays, there is more than one choice to perform the communication between the data plane devices and their controllers. Some controller framework examples ranges from the P4Runtime [38], the Behavioral Model version 2 (BMv2) [39], and Open vSwitch for P4 [40, 41]. In brief, any of these tools are standards to set the P4 targets in real-time. Meanwhile, the in-depth explanation of these are out of the scope of this dissertation. Nonetheless, they provide P4 primitives to configure P4 tables, keys, and actions to support the appropriate deployment and operation of the targets.

2.2.5 Closing Remarks

P4 has received momentum from the network community. The overall P4 architecture has matured over the years, evolving from the initial proposal [30] to broaden its scope for other network devices [33]. This standard has also evolved to surpass new challenges, being currently in the P4 16 version [35]. Some tools are proposed to encourage and ease the P4 deployment as the mentioned controller framework, as well as contemporary OS like Stratum [37] from Open Networking Foundation (ONF).

2.3 FIXP

FIXP [42, 43] is a project developed taking advantage of the SDN technology [42]. Therefore, it leverages a multi-architec- ture network environment, wherein its core is P4-based forwarding devices, i.e. FIXP switches capable of receiving, processing, and forwarding packets from distinct architecture sources. Therefore, this Hardware (HW) is flexible enough to accommodate various functionalities and architectures. Currently, FIXP forwards data from the NG, ETArch, and TCP/IP architectures.

It is also important to mention that the FIXP proposal is sponsored by the Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), through the grant number #2015/24518-4. This study was also financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. Moreover, it is has the collaboration between the Universidade Federal de Uberlândia (UFU), Universidade Federal de São Carlos - Sorocaba (UFSCar - Sorocaba), Universidade Federal de Pernambuco (UFPE), and Instituto Nacional de Telecomunicações (INATEL).

2.3.1 Hardware Design

Regarding the FIXP project, it has three different layers. The first is named Physical Layer (PL), which focuses on the logical connection of devices and data exchange. The second is the Abstraction Layer (AL), which is a middleware between the Data and the Control Planes. Finally, the Control Layer (CL) has the main function to receive unknown packets from the underlying CP and discover the best way to set the underlying FIXP switches. Figure 2.3 illustrates this concept and Figure 2.5 shows the interaction between each layer, component, and services inside the FIXP architecture.

Firstly, the PL fosters the interconnection of the network elements, interconnecting distinct network domains, exchanging data from different architectures (1) (12). Meanwhile, it is crucial to note that this layer can be composed of several FIXPs, as illustrated in Figure 2.4.

Moreover, the FIXPs are modeled by the **FIXP.P4** (Figure 2.5). This

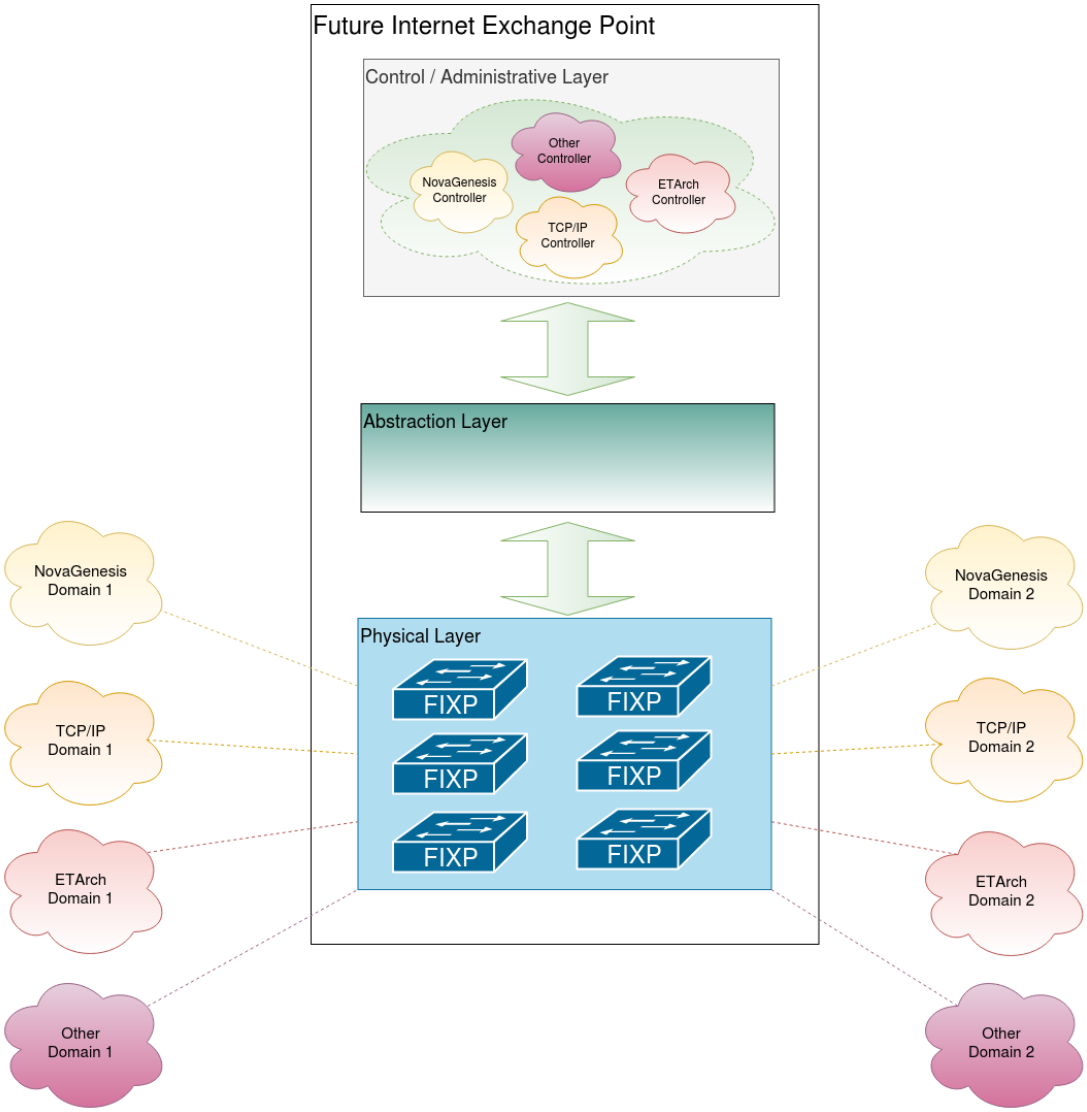

Figure 2.3: *FIXP architecture overview. Adapted from [40].*

source code defines the overall operation of a FIXP, providing the framework that understands the architectures' headers and provides the required Routing Tables (RTs). Whenever a new packet is received, FIXP.P4 parses this packet, checking if the routing key has a match in a RT. If there is a match, it forwards the packet to the respective egress port to reach its destination (12). Otherwise, the data goes to the AL (2). Alongside FIXP.P4, another program called FIXP Rule Handler Service (FRHS) executes in parallel. This Python program deals only with FIXP incoming primitives. In its operation scope, it can set the FIXP on the run (6) and notify the CP about the success of any configuration by an Acknowledgement (Ack) packet (8).

In its turn, the AL has the essential purpose in coordinating different architectures packets, regardless of the architecture or the underlying switch. In this way, this layer has the Python scripts FIXP Switch Packet Handler (FSPH) and FIXP Controller Packet Handler (FCPH). The first application

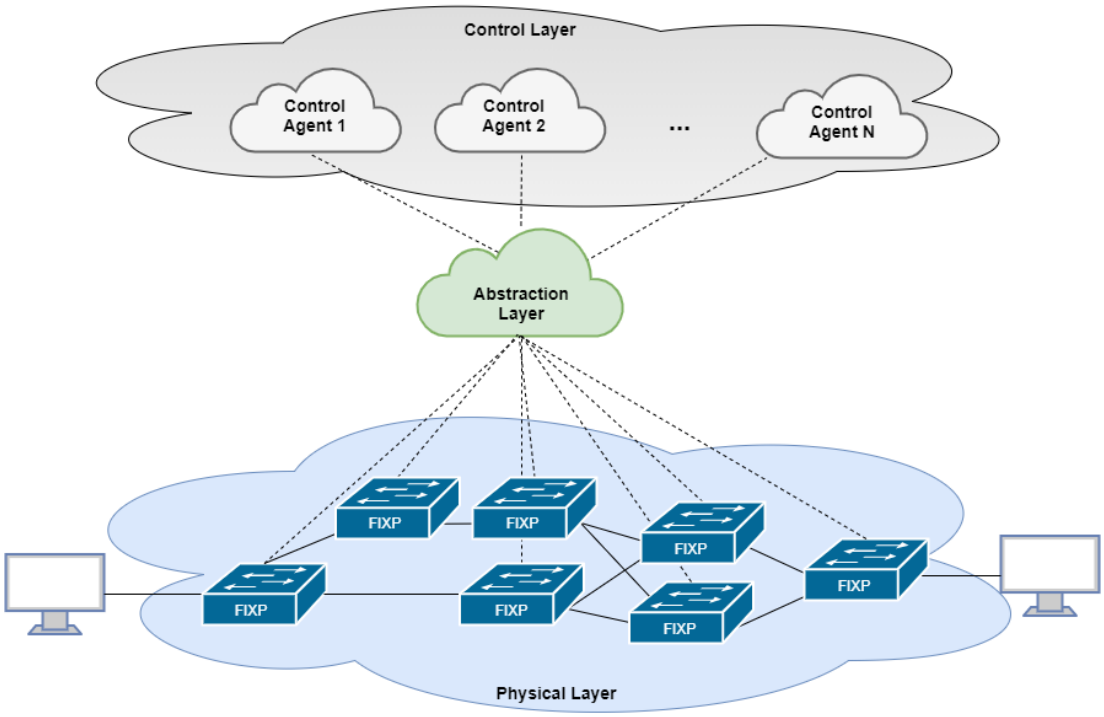

Figure 2.4: *Example of any FIXP logical connection and its concepts of Data, Abstraction, and Control Layers.*

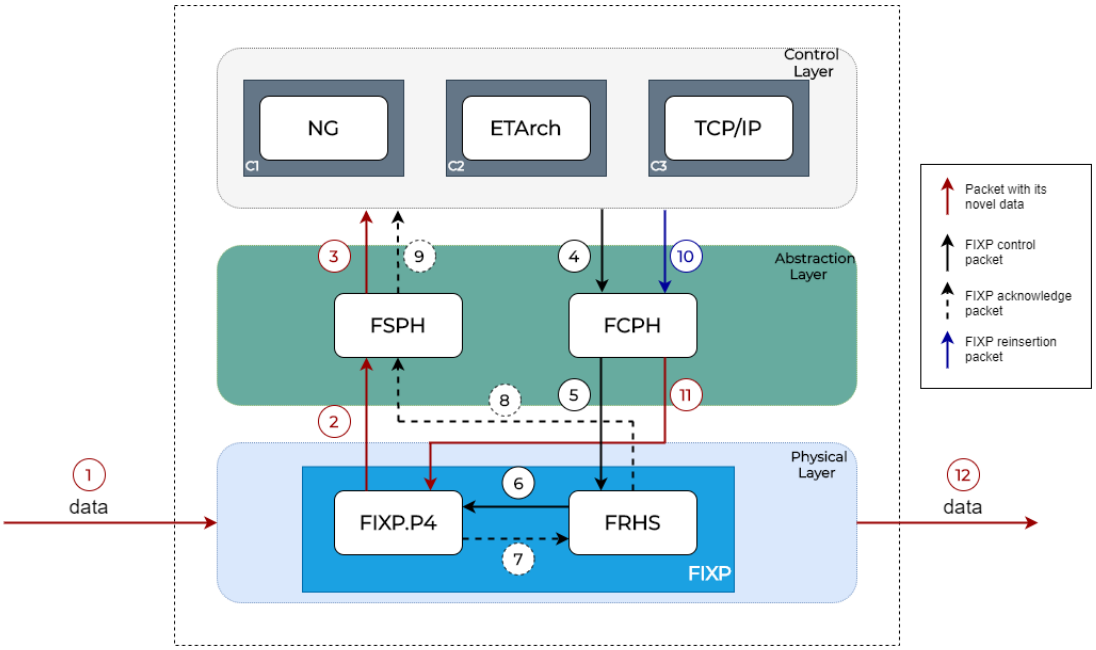

Figure 2.5: *FIXP internal architecture. Adapted from [49].*

inspects the received packets from the PL and decides which controller must receive a given request. If this data is incoming from a host (2) or represents a Ack primitive (8), it selects the proper network interface for the desired controller (3). On the other hand, FCPH deals with the contrary

flow from CP to the DP, forwarding the FIXP primitives (4) to its proper underlying FIXP Switch (5) or translating the reinsertion primitive (10) into the novel architecture standard to forward to the desired FIXP Switch (11).

Finally, CL accommodates the required controllers to support the FIXP operation. These controllers are disjointed, where each one implements an independent strategy to support its architecture. Regardless of the specific architecture, the controllers provide the necessary intelligence for the configuration of FIXPs in real-time. At the moment, the scenario runs with three different controllers, one for NG, one for ETArch and one for TCP/IP.

To summarize, each layer has its purposes, exchanging data from the original architectures or FIXP control data necessary to to leverage the network communication. Moreover, the CL can present several control agents that oversees the PL according to a specific strategy. In this case, FIXP does not define the best way to build this knowledge in the CP, it does not even require the reinsertion packet if the developers decide not to encompass this trickery. If they choose so, the successful Ack packet is the threshold to discard the original packet anytime. Therefore, the path of (10) and (11) does not exist in this case. After the proper configuration of the underlying FIXP, the underlying switch directly forwards any incoming packet (1).

2.3.2 FIXP Control Primitives

FIXP characterizes some primitive patterns to allow the configuration of the underlying devices in a transparent, flexible, and suitable manner to any architecture. Therefore, the controllers must model such primitives, encapsulating them in an Ethernet frame for the inter-FIXP communication.

Regardless of the primitive type, the controller agent must follow a common header that is applied to the Ethernet frame. Table 2.1 briefs the generic FIXP primitive fields, which are explained as follows below:

- **Destination Address** [1] : Value representing the destination MAC of the Ethernet packet with 6 bytes.
- **Source Address**: Value representing the source MAC of the Ethernet packet. Like the Ethernet Protocol standard, it has 6 bytes.
- **Ethertype**: Value representing the protocol type encapsulated in the Ethernet frame with 2 bytes. This field may take the specific Ethertypes from NG, ETArch, TCP/IP, or “0x0900” for the FIXP control protocol. Like the Ethernet Protocol standard, it has 2 bytes.
- **Payload**: Encapsulates the FIXP primitives, ranging from 46 to 1500 bytes.

[1] Whenever a MAC is configured as this, it follows the ASCII table pattern. This yields that ‘F’ = 0x46, ‘I’ = 0x49, ‘X’=0x58, ‘P’=0x50, ‘0’=0x30, and so on. In other words, ‘F’:‘I’:‘X’:‘P’:‘0’:‘0’ is the same as 46:49:58:50:30:30.

Table 2.1: *FIXP Generic Control Primitive Example.*

Ethernet Frame		
Description	**Content**	**Example**
Destination Address	FIXP MAC	‘F’:‘I’:‘X’:‘P’:‘0’:‘0’
Source Address	Controller MAC	AA:BB:CC:DD:EE:FF
Ethertype	FIXP Protocol	0x0900
Payload	FIXP Primitive	Desired Primitive

Furthermore, each Ethernet frame’s payload must comply with the FIXP primitive standard. For this, JSON is the standard adopted to set these fields flexibly and acceptably, as it is also a widespread format in other Internet data structures and applications. Figure 2.6 conveys the three types of possible data, which are individually explained in the next subsections.

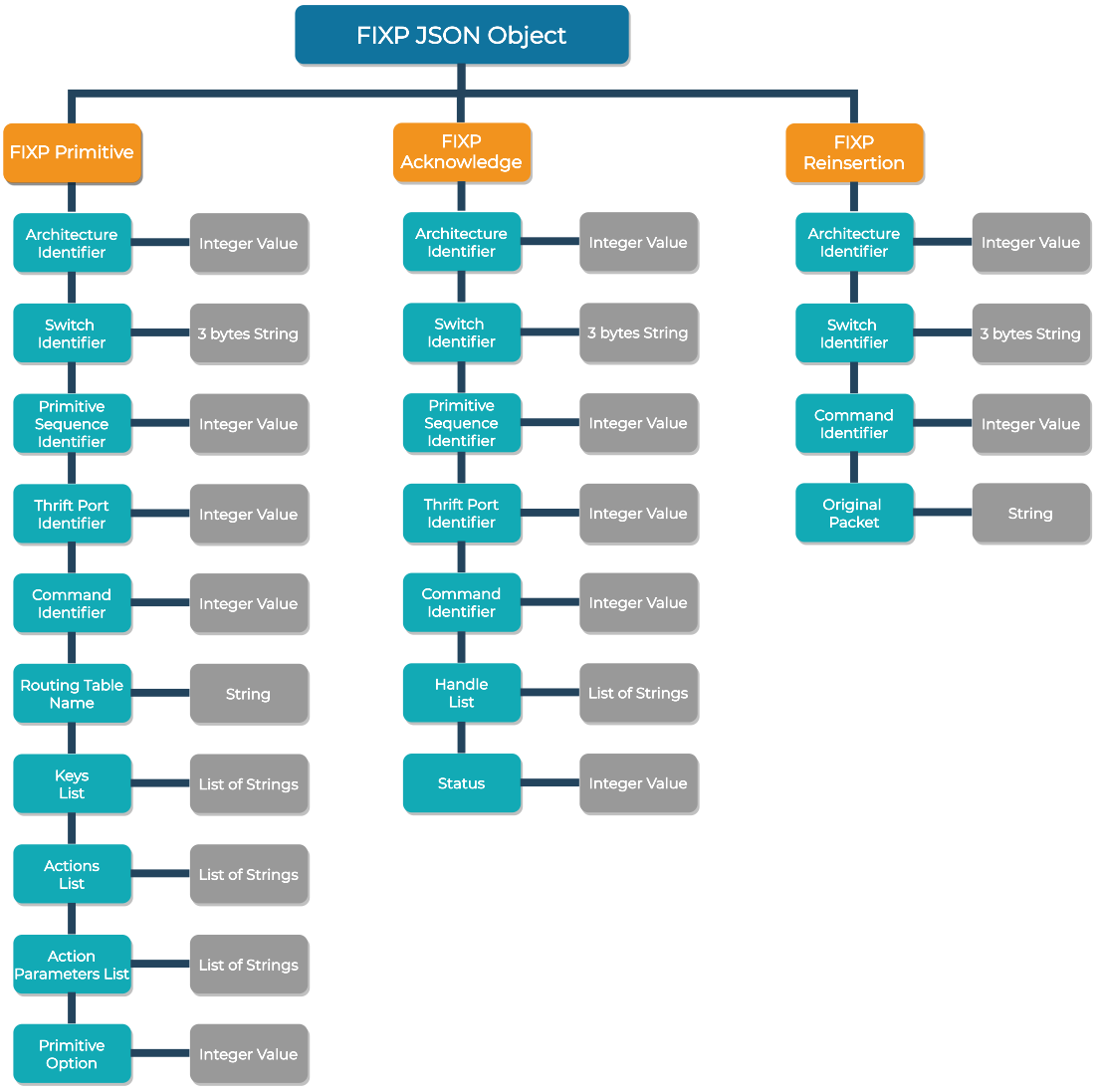

Figure 2.6: *FIXP Primitives JSON tree structure.*

2.3.3 Table Add-Modify

Through the *Table Add-Modify* primitive, controllers can modify or add an entry in their architecture's FIXP RTs. Table 2.2 sets out the parameters needed to perform this function. The required fields are explained bellow:

- **Ethertype**: Designates the architecture that will be affected by the primitive. In other words, it must be the value of 2 bytes that yields to the *Ethertype* of the desired protocol, e.g. NG is 0x1234.
- **Switch Identifier**: Points to which switch shall receive the primitive. This field is expressed on a 3 bytes string that conveys to the pattern "s<switchNumber>", wherein switchNumber varies from 00 to 99.
- **Primitive Sequence Identifier**: Represents a sequential number that traces the primitive packets, linking Ack and setting primitives. This field must be an integer number that fits 1 byte.
- **Thrift-Port Identifier**: Designates which thrift port is used to configure the switch. This fields must be an integer number.
- **Command Identifier**: Designates the primitive command, so the FRHS decodes it into the required P4 action. This field is an integer number.
- **Routing Table Name**: Specifies which P4 Routing Table should be modified, being vital for FIXP multi-architecture environment. It follows the BMv2 pattern "**FIXP_Switch_Ingress.Table_Name**."
- **Keys List**: Highlights the parameters to be entered or modified in P4 RT. In the case of NG architecture, this field specifies the Host Identifier (HID) routing key. This field must convey a list of strings.
- **Actions List**: Delineates the P4 action that must be modified. This follows the BMv2 pattern "**FIXP_Switch_Ingress.Architecture_Action_Name**" and it conveys a list of strings.
- **Action Parameters List**: Drawns the parameters to be inserted in the required P4 action. For *Table Add* or *Table Modify*, it conveys the *Egress Ports*. This field is a list of strings.
- **Primitive Option**: For the *Table Add-Modify* primitive, it distinguishes two distinct actions: the value "0" adds a new key on its RT, while "1" modifies a previous entry. This field is an integer of 1 byte.

2.3.4 Primitive Acknowledgement Primitive

Whenever FSPH translates a primitive into a BMv2 pattern and sets the FIXP switch through its Thrift Port, it also retrieves the configuration status. At this moment, BMv2 can return a success or a failure for any reason. Based on this, FSPH generates an Ack packet to the controller and this controller becomes aware of the advancements in the PL.

Table 2.3 illustrates the Ack of a Table Add-Modify primitive, encoded in

Table 2.2: *Table Add-Modify Primitive Parameters.*

Ethernet Frame			
Description		**Content**	**Example**
Destination Address		6 bytes	‘F’:‘I’:‘X’:‘P’:‘0’:‘1’
Source Address		6 bytes	‘N’:‘G’:‘C’:‘O’:‘N’:‘T’
Ethertype		2 bytes	0x0900
Payload	Architecture Identifier	Integer	0x1234
	Primitive Sequence Identifier	Integer	1
	Switch Identifier	3 bytes string	“s01"
	Thrift-Port Identifier	Integer	9090
	Command Identifier	Integer	1
	Routing Table Name	String	“FIXP_Switch_Ingress.novagenesis_ forward"
	Key List	List of strings	{ [“HID_1"] }
	Actions List	List of strings	{ [“FIXP_Switch_Ingress.novagenesis_ SetSpec"] }
	Action Parameters List	List of strings	{ [3] }
	Primitive Option	Integer	0 (add) ou 1 (modify)

the JSON format. As some parameters are common to the *Table Add-Modify* primitive, this table focuses on the specific Ack fields.

- **Handle List**: It is an performed action identifier, which is another way to access this operation later.
- **Status**: FCPH sets this field with “1” to convey a successful command and “0” for a failure.

Table 2.3: *Primitive Acknowledgment Parameters Reply.*

Ethernet Frame			
Description		**Content**	**Example**
Destination Address		Hexadecimal	‘N’:‘G’:‘C’:‘O’:‘N’:‘T’
Source Address		Hexadecimal	‘F’:‘I’:‘X’:‘P’:‘0’:‘1’
Ethertype		Hexadecimal	0x0900
Payload	Architecture Identifier	Integer	0x1234
	Primitive Sequence Identifier	Integer	1
	Switch Identifier	3 bytes String	“s01”
	Thrift-Port Identifier	Integer	9090
	Command Identifier	Integer	1
	Handle List	String	[...]
	Status	Integer	0 (fail) ou 1 (success)

2.3.5 Reinsertion Packet

FIXP considers the likelihood of a controller reinserting an original packet. For every unknown packet, a controller can discard or store this data for reinsertion after receiving a successful Ack. For the latter, Table 2.4 describes the Reinsertion Primitive packet, encoded in JSON. This primitive shares some common fields and the list below presents its specifics fields.

- **Original Packet**: This field must be the exact original packet received by the controller. Through this field, the AL deparses the JSON object and forwards the original packet to the FIXP.

Table 2.4: *Reinsertion Primitive Parameters.*

Ethernet Frame			
Description		**Content**	**Example**
Destination Address		Hexadecimal	'F':'I':'X':'P':'0':'1'
Source Address		Hexadecimal	'N':'G':'C':'O':'N':'T'
Ethertype		Hexadecimal	0x0900
PAY LOAD	Architecture Identifier	Integer	0x1234
	Switch Identifier	3 bytes String	"s01"
	Command Identifier	Integer	'5'
	Original Packet	Multiple Bytes	Original Packet Content

2.3.6 Example of a Packet Flow in the FIXP scenario

This section presents an example of how FIXP forwards a hypothetical and generic packet through the interaction between its layers and software. Notwithstanding, it is worth mentioning that there is no concern in actually portraying how a given architecture behaves in this environment at this moment. This is going to be exploited in the further sections to explain how NG manages the FIXP environment. Figure 2.7 depicts the interaction of the solution demonstrated under the layered vision of FIXP.

For instance, consider that a host wishes to reach Host 2 through the FIXP HW. This communicating device sends a packet in its novel architecture protocol, i.e. with its known ethertype, and CL receives this frame (1) on its specific FIXP Ingress Port (IngPort). Since P4 language has already a model of this architecture, the **FIXP.P4** parsers the packet, decoding the frame fields based on the precise headers. Thus, it gathers the destination Identifier (ID) and other relevant data. Through these data, FIXP looks up at its RT and checks whether there is a match for the obtained keys. As we consider a not configured FIXP, it forwards this packet to the CP (2).

After this, FSPH forwards the received data to its proper controller (3). At CL, the controller receives the packet and proceeds according to its specific strategies. In this example, the controller discovers how to achieve the packet's destination and generate the Table Add-Modify primitive (4).

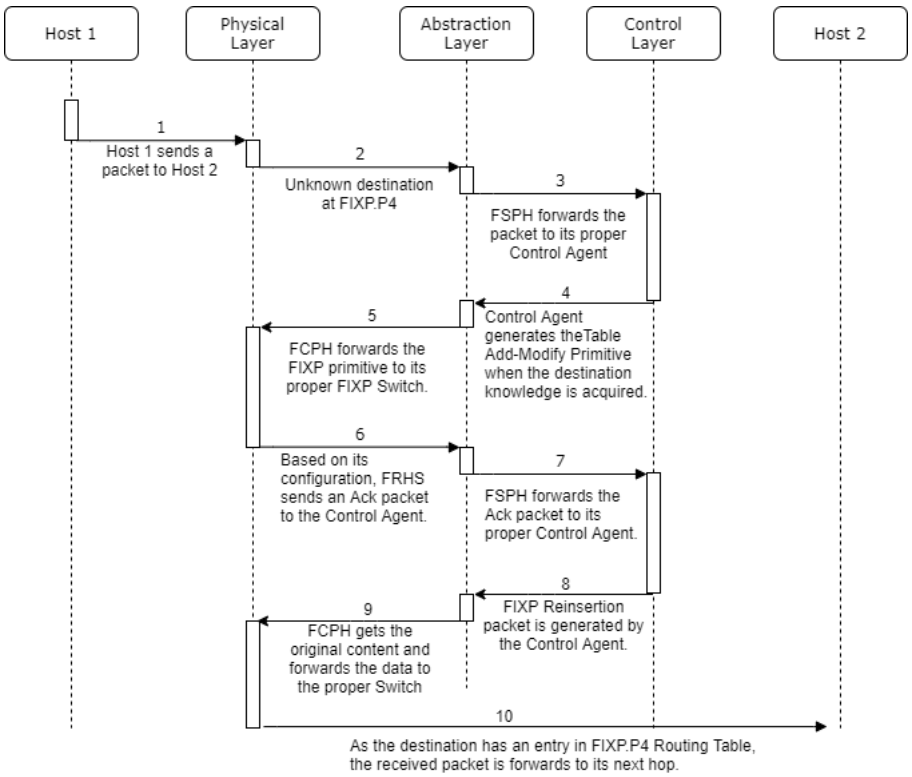

Figure 2.7: *Example of a hypothetical architecture communicating through FIXP.*

After that, AL's FCPH forwards the control packet to the desired switch (5). Upon receiving the packet, FRHS translates the Table Add-Modify primitive into the BMv2 pattern and sets the desired architecture's RT. Then, FRHS retrieves the operation status from the Thrift Server to create the Ack packet, sending it to the specific controller (6). The controller receives this Ack (7) and it can later reinsert a packet into the network for a success status (8) or resend the primitive again for a failure.

The FCPH receives the reinsertion packet (8), retrieves the original content, and the FIXP switch destination from this primitive. Through this data, FCPH forwards the original packet to the specific switch (9), where FIXP.P4 receives as the original packet. As it has already the rule for routing, the packet is forwarded to the appropriate Egress Port (EP) interface (10). From this moment on, any packet aiming the same destination will be forwarded to the next hop without involving AL and CP.

2.3.7 Closing Remarks

FIXP is a Brazilian SDN project that promotes a multi-architecture Internet network. At this moment, it acts as an interconnection point for 3 disparate architectures, being these the NG, ETArch, and TCP/IP. As it has been proposed, any novel communication architecture can join this scope with ease. This fact is first because P4 is an architecture flexible enough

to accommodate and model essentially any standard. Secondly, the NG version of the FIXP protocol applies the JavaScript Object Notation (JSON) standard, which is a solid and well-known lightweight data format that most of the programming languages and devices can exploit.

2.4 NovaGenesis

Throughout this sub-chapter, NG is presented in-depth, delimiting its main components, services, and philosophy. Given the aim of the developed work, this sub-chapter also focuses on the NG communicating protocol, covering topics of NG messages, fragmentation, and actions. Therefore, this chapter expected takeaways concern more with preparing the reader to understand the future discussions, mainly with the collaboration between NG and the P4-based *hardware* requirements, solutions, and challenges.

Therefore, the crucial question to be addressed by this chapter relates to something even more important than the NG concepts, more than its ingredients, more than every technology that can be encompassed into its scope. The question is quite simple, yet complex: how NG is relevant to our society? How can it be generic enough to accommodate new technologies as they arise?

2.4.1 Project Design

The NG project bases on a 2010 study about Future Internet [44]. This article outlines the importance of a “clean slate” architecture and the technological requirements, challenges, and trends that a new infrastructure must have to accommodate the applications of a foreseen future. Moreover, it addresses the inefficiencies of the current Internet model, which has a level of heterogeneity and inconsistency that has increased as new solutions have been proposed and incorporated into its design [45]. Thus, this reference was one of the main inspirations to establish the best requirements, components, and ingredients that would compose the FIA called NG.

Briefly, NG focuses on integrating processing, storage, data exchange, visualization, virtualization, and several other elements and applications, seeking to be flexible enough to accommodate another future emerging concepts [46]. In other words, one of its cornerstones is to be generic enough to provide a communication infrastructure that meets not only the present applications, but also new future trends that arise from society’s demands [47]. In this way, it can keep relevant and avoid being obsolete. Another pillar is its self-organizing structure. This concept enables the architecture to dynamically allocate individual resources to supply the execution and needs of a given application [45].

Concerning some current trends, NG covers aspects of Information Cen-

tric Network (ICN), Self-Verifying Name (SVN), SDN, Service-Oriented Architecture (SOA), Service-Centric Network (SCN), Network Function Virtualization (NFV), IoT, Self-Organizing Network (SON), distributed name resolution, Identifier/Locator (ID/Loc) splitting [48]... We truly believe that we can keep adding new topics to this list and it might be probably unlimited. This is because technology is an infinite game of perpetual development that improves itself as new challenges arise. These disjointed ingredients are cohesively added into NG core. Anything that boosts and does not go against NG project scope can be adapted into this idea. Briefly, five design pillars were chosen to drive this work. These are explained bellow [49]. Nonetheless, there are about 15 design ingredients and pillars for NG.

Entities

First of all, we must clearly define the entity nomenclature. In the NG architecture scope, entity designates both physical and virtual existences that have the function of dealing with information or data in general [45].

For example, let us consider the Hubble Space Observatory that observes our Solar System in outer space. This infrastructure acquires infrared, visible, and ultraviolet light based on its optical system called Optical Telescope Assembly [50]. Moreover, other systems monitor outer space objects and their traits, such as stars, quasars, celestial object's temperature, chemical composition, density, and motion [51]. Besides the image instruments, Hubble's presents an efficient spacecraft system able to harvest the solar energy and produce electricity, thermally protect the telescope, control the equipment and transmit data [51]. Several digital processes are happening inside each system, e.g. data processing, exchanging, or storing.

Based on this example, a NG Hubble system can address each real or virtual entity by its nature. For instance, there would be NG entities that represent each embedded system that controls the instruments, data acquisition, or pointing system, as well as NG entities that exchange the visual and other crucial spacecraft data between NG agents. Right after this, we explore how NG address each entity and its naming structure.

Naming and Naming Resolution

In NG, each entity has a specific perpetual name that identifies it within any scope. In its turn, each entity can produce a particular service and/or generate content, which is a set of data. This data can be the result of any input processing information, data exchange, or content storage [45]. Each data or content can also be named within a NG scope.

As expected, any entity must be named in the domains of NG (scopes) to distinguish each existence. For this, there is the flexibility to use both Natural Language Naming (NLN) and SVN [45]. The first concerns ap-

pointments that reflect semantically meaningful assignments for human beings. To exemplify this, Hubble can represent the cosmic environment by the proper current human astronomical convention to name a space object detected by its lenses. If Hubble monitors the Solar System's celestial bodies, there could be digital entities that represent the planets with their individual well-known names (Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, and Neptune) or the whole group of Solar System's Planets. Contrary to the current Internet, that recognizes names of a few set of entities, NG provides an unlimited (in terms of namespaces) support for naming.

On the other hand, SVN increases network security. These logical names result from hash functions that process an input and convert them into a defined pattern [45] – a name. Such functions accept a random amount of variable-sized input and standardize them on fixed-length output as if it were data encryption. When performing such a procedure, a name that could express something semantically rich like "Saturn" becomes something equivalent in a given binary pattern, such as "233ea1805dab5120a0-ecc49b71a48063" in the MD5 hash function. Therefore, SVNes guarantee the anonymity of an entity to external eyes once an application can use a wide range of hash techniques to generate a SVN. Moreover, SVNes avoids data duplication, based on the fact that a given input has a single output in the employed standard generally. Besides, authentication of SVNes ensures the data integrity [49]. When applied to services and operating systems, SVNes allow to determine whether such software are still intact.

Therefore, NG naming concerns nothing more than the attribution of symbols to one or more entities, carrying a sense of individual identity to each. This naming is mostly unlimited, accepting NLN or SVNes.

Identifiers, Locators, Name-Bindings, and Name Resolution

In NG, a name by itself is not an identifier. For instance, it can be really confusing to identify a person if we only consider his/her first name in a given geographical area. This naming ambiguity is because there must be several people with the same first name. How many people with the first name as "Thiago," "João," or "Maria" can be found in a single city like Santa Rita do Sapucaí? If we expand or decrease the analyzed geographical scope, the number of occurrences will also change. This yields ambiguity to ensure the proper identification of an entity in the given area. Therefore, a name is only an identifier if it is unique in a given scope/domain that unambiguously determine an entity in NG's scope.

Regarding the location in the context of NG, it occurs through another name called "locator." This name binds to an entity to reflect its current location. Such name-bindings give a sense of the relationship between two or more distinct named entities. Tuples delimit these name-bindings in the form of in the format "<key, value (s)>" in NG [44].

Through name bindings, NG detaches identifiers from locators. This is called ID/Loc splitting, fostering the entities' mobility in several scopes without losing track of their identities. In this way, NG ensures entities' traceability and full accountability in the network by their name's immutability. For example, the tuple <Thiago, Melbourne, Australia> represents the location of the entity "Thiago" in the scope "Melbourne" in "Australia." In case of displacement, this tuple modifies to reflect the new position like <Thiago, Sydney, Australia>, without losing the entities identity.

NG Smart Objects, Gateways, and Controllers

NG encourages a distributed environment where each entity can request and/or accomplish an action. In this way, a rich communication network is dynamically orchestrated, based on entities' agents that foster this rendezvous among seekers and providers.

Each NG entity presents a Proxy/Gateway Controller Service (PGCS) for dynamic resource composition [46], representing a NG domain, a single service, or a single entity, connecting this existence to the NG infrastructure. This service is also a network gateway that manages the different communication between distinct technologies, translating NG messages to any modeled link layer technology. At last, it also is a controller for managing programmable physical or virtual resources [52].

Servitization, Life-Cycles and Contract-based Orchestration

Following the SOA premises, NG considers "everything-as-a-service" to dynamically compose its functionalities [48]. Any entity is capable of performing a task must offer this ability to the network. In its turn, whichever entity that requires something to fulfill an action must request it from the network. Through this concept, the rendezvous between service offers and requests fosters the dynamical network composition by each entity proxy. Meanwhile, the concept of service is nothing less than a virtual entity focused on processing, exchanging, or storing information [46].

Taking Hubble's environment as an example, a NG entity that oversees the Milky Way's collects data that may require temporary data storage. This data acquisition can request this storage from a NG data center onboard and establish a storage service. After a while, a NG entity that represents this Hubble's data recorders can now contact a NG service that manages the communication antennas and satellites to transmit the gathered data to NASA. After exposing this need and setting up this service, the storage agent can exploit the transmission channels. On the other hand, a NG agent located in NASA can request an adjustment on the Hubble's pointing system by a specific NG agent that controls this system.

Through this resource discovery, NG fosters a context-based orchestra-

tion of services and applications based on the name bindings [46]. Each entity's PGCS manage this communication, dealing with service proposals and establishing contracts to perform a task. This yields in another NG concept called Service Level Agreement (SLA), which represents the contract between two entities. In this negotiation, the peers define the overall service QoS, as well as services' boundaries, i.e. what each service is allowed to operate, their responsibilities, liabilities, penalties, and any other clause [52]. As the network expands, micro-services builds the network stack dynamically [46], since many protocol implementations today are virtual network functions. NG has envisioned this trend first.

After establishing the SLA, the peers carry out the service. During this stage, the peers consider the terms of quality and possible penalties due to faulty operation. In some applications, it is even feasible to develop mechanisms that handle this feedback and adjust the performance to improve the overall quality of service. Through this concept, NG achieves an operation with minimum humane interference, as the players can act autonomously based on the preceding SLA and possible adjustment controls [46].

When established service finishes, the peers dismiss their SLA and revoke their credentials to access each other domains. This service's dismissal stage only happens when the task is concluded or, perhaps, when the QoS is so meager that the peer has to shut down the operation. After this, the capabilities are released and made available again to the network, restarting the life cycle anew.

As examples, NG see event protocol implementations in software as services and, deriving from this, any service, protocol, or application are seen as services and must attend to this life cycle. Through this, NG foster hardware programmability, control plane dynamical management, and any virtual function into its scope [48].

2.4.2 NovaGenesis Main Services and Processes

This section focuses on the main NG services. In parallel, it aims at highlighting the relationship of each service's role and the NG cornerstones.

Proxy/Gateway Controller Service

The PGCS is one of the primary NG core services, encompassing the functions of proxy, gateway, and controller. This service is the access point between the NG inter-communication, wherein it adapts NG raw messages into modern or legacy link layer technologies and vice versa. In other words, one of PGCS main functions is related in translating the NG protocol into established technologies, such as Ethernet, Wi-Fi, ZigBee, and so on [48]. In this premise, PGCS deals with encapsulation, fragmentation and re-assemblage of raw NG messages into the desired protocol parameters.

As a proxy, PGCS represents ordinary things at software layer [48]. This fosters a smart service's ecosystem, where distinct and disjointed peers exposes their available capabilities. Through this exposure, NG stack is dynamically composed. Moreover, PGCS fosters the programmability of connected devices, such as sensors, atuators, switches or any other physical or virtual programmable thing. Besides, it can also manage general configurations of represented devices on the fly [48].

Based on the ID-oriented naming structure and routing, PGCS forwards NG messages over the link layer. Therefore, PGCS relies on the message's destination SVN to decide whether a message is intra- or inter-domain communication. In both cases, PGCS can be seen as a software routing service that manages the exchange data traffic inside a NG domain [47]. At last, PGCS provides bootstrapping functionalities to enable other service's discovery within the same domain and fully initialize a domain.

Gateway

The Gateway is a internal component (block) from a NG service. This component interacts directly with the operating system Shared Memory (SHM), boosting inter service communication in an OS. This means that any message or data exchange between NG services within a same OS happens through the SHM, wherein the Gateway (GW) puts the NG message into the SHM's and the destination GW takes the required message out of the SHM. Moreover, the GW fosters the inter component communication inside a service, adopting event-driven dispatching of messages and callback of service actions [48]. NG is also an event driven emulator/simulator that can be adopted to evaluate network performance similarly to NS3 or OPNET.

Hash Table

The Hash Table (HT) internal component (block) stores any kind of name binding. For example, it can register the peer's SVNes of each individual block. Therefore, it collaborates with the GW process at forwarding message, wherein HT has the other services and processes SVNes.

Name Resolution and Networking Cache System

The Name Resolution and Networking Cache System (NRNCS) encompasses other three NG services [48]: Publish/Subscribe Service (PSS), Generic Indirection Resolution System (GIRS), and Hash Table Service (HTS). The role of NRNCS is to store SVNs, provide networking cache of content, and improve the data exchange, fostering the decentralization of content.

The PSS is a distributed API for service discovery and access based on the naming structure. This service performs the following tasks [48]:

- NB and content publishing without notification of other services;
- NB and content publishing based on the request of interested services;
- NB and content subscription;
- NB and content subscription based on the request of a publisher;
- NB and content delivering to subscribed peers;
- Revokes published NB and content, if any.

The second service of NRNCS is GIRS. This service receives pub/sub messages from PSS and forwards them to the best hash table instance.

The third, and last, service of NRNCS is HTS. This service is responsible to store Name-Binding (NB)s and associated contents. Through the NG premises, NRNCS is capable of creating a data structure to build distributed hash tables based on the acquired SVNs. As a result, NBs and related contents are stored in a distributed way.

2.4.3 NovaGenesis Protocol and Layered Model

Now that a glimpse of the main NG concepts has been shown, we are going to focus on the protocol specifics. Firstly, this sub-section presents the NG Layer Model, organizing each main NG service in a coherent manner. After establishing this concept, it depicts how each player interacts within a given domain, focusing on the NG inter-communication. Following, it discusses the NG protocol in broad, exposing the main NG messages and actions. Finally, it outlines the cases where fragmentation occurs and how the NG fragmented messages are structured. This section is crucial to understand the NovaGenesis Control Agent for Future Internet eXchange Point.

NG Layering

NG is a convergent data architecture that aims at processing, storing, and exchanging data. This project takes advantage of several legacy Link Layer technologies to connect its hosts, such as IEEE 802.3, LoRa, or Bluetooth. In other words, NG does not establish a specific NG protocol to be sent over the Physical Layer yet. Every Link Layer technology must be priory modeled into the NG stack to exploit its characteristics. In broad, there is no difference in these technologies control headers whenever someone applies NG. Figure 2.8 depicts the NG generic communication layer model, considering the Link Layer, Raw Socket, Convergence Layer, NovaGenesis Layer, and the NovaGenesis Application Layer.

A specific Raw Socket retrieves the data at the Physical Layer and delivers it to the Convergence Layer. Meanwhile, raw sockets receive packets directly from the Link Layer [53]. These sockets sidestep any data process-

ing and deliver the raw data directly to the application on the upper layers.

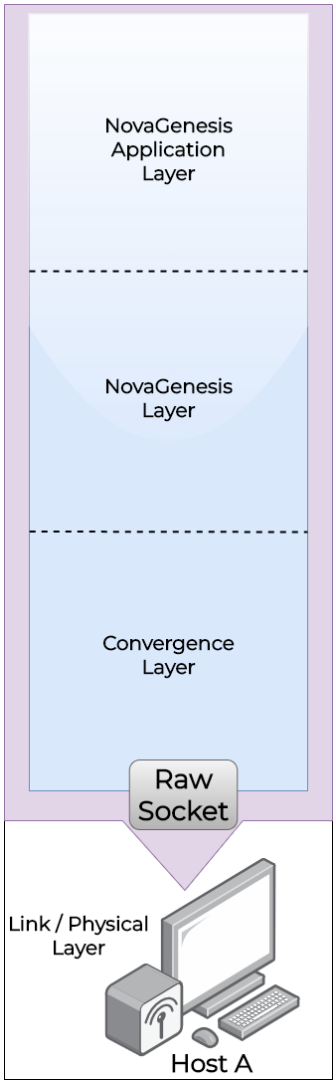

Figure 2.8: *NG layered communication model.*

Interacting directly with this socket, the PGCS is the only NG service that performs inter-domain communication. This service receives the raw data and assigns a data buffer to store the received packet at the NovaGenesis Layer.

The NG Layer inserts control headers to identify each unique NG message. These extra headers enables the fragmentation and reassembling of NG messages. Moreover, this layer provides the intra-processes communication. At the NG Application Layer, novel NG applications execute their functions based on the NG messages.

Intra-Domain Communication

Figure 2.9 depicts a hypothetical NG host internal structure. Notice that every data exchange between NG processes takes advantage of each GW and the SHM. Upon verifying an existing process in its service's scope, the GW enables the data forwarding to the specific process at the same OS. At last, every host must present a PGCS to receive or forward data to/from its domain. In this way, NG has a fractal structure in terms of its main processes. In other words, there is a self-similar internal organization that can be replied to several distinct applications. For instance, every NG service must deploy the GW and HT core blocks to communicate inside or outside an OS.

On the other hand, inter-communication data is produced on an high-level specialized NG service and goes through inter-processes inside a host to adapt a sequence of NG messages and send them to another host domain, based on the desired Link Layer technology. In this communication, every outbound-destined data interacts directly with the NG PGCS. As explained, this block interfaces with inter- and intra- domain communications, adapting the received NG messages to a given legacy network protocol or translating received packets into NG pattern. In specific, PGCS has a core block called Proxy/Gateway (PG) that interacts directly with the Raw Sockets. These sockets must be specific for each distinct communication protocol. Through these, PGCS receives or forwards data from/to a desired communicating peer.

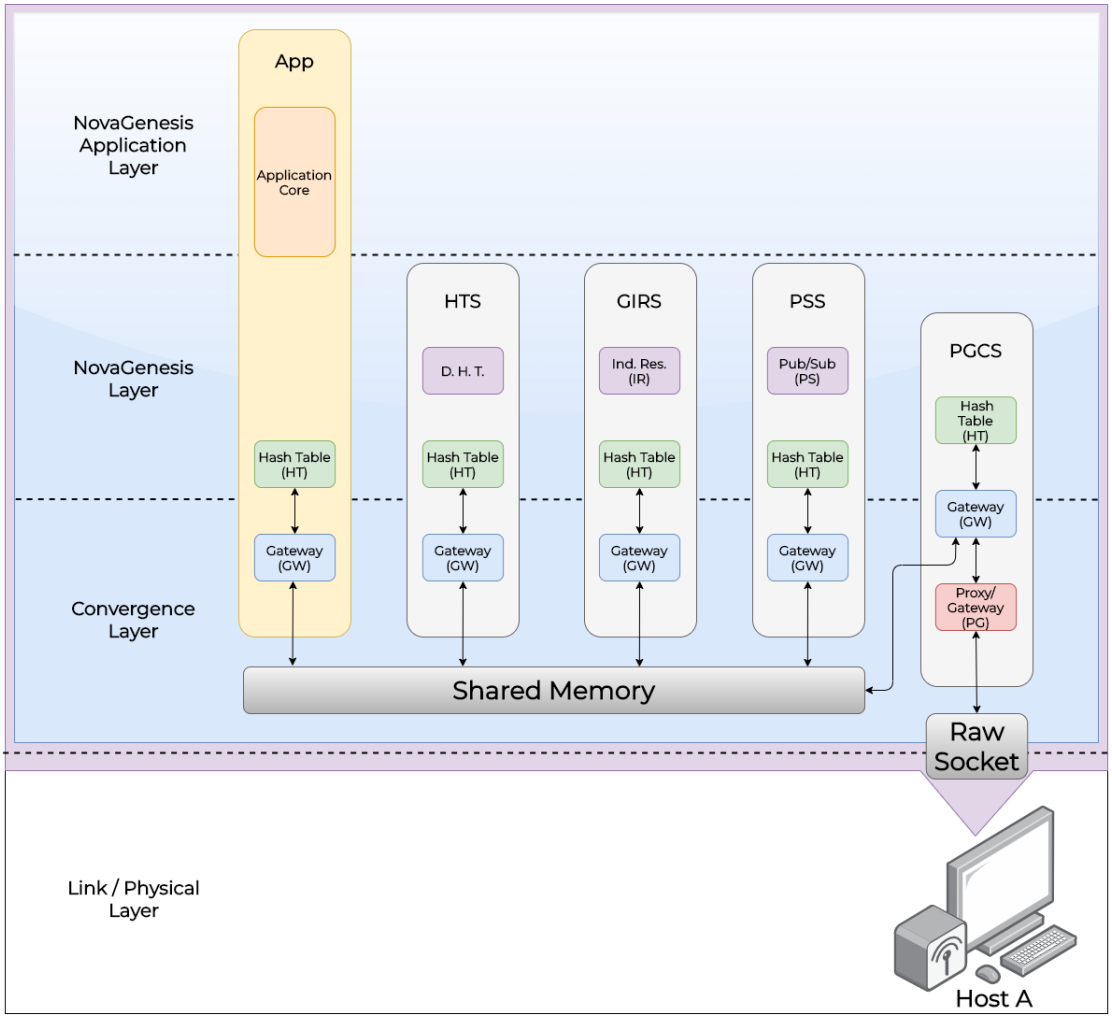

Figure 2.9: *Depiction of the NG structure based on a layered model with its novel components and its main processes.*

NovaGenesis Messages and Actions

NG exploits NG Messages to exchange data through NG peers. This message conveys actions that are based on NG Command Lines, specifying the operations to be performed [46]. On the other hand, these command lines might present payloads that act as the parameters of a given action [48]. For instance, the NG Message can be seen as the behavior pattern that a NG peer generate, which conveys the desired functions to be performed by a NG process, service, or host at destination. Each function is a specific NG Command Line that may present a given payload, i.e. a set of input data or parameters to generate an aimed output through that action. The ASCII format encodes the whole NG Message, splitting each NG Command Line through the “End of Line” character, i.e. the ASCII ‘\n’ or 0x0A [48]. The following box depicts the generic form of a NG Command Line:

ng −command −−alternative version [< n type E1 E2 E3 E4 ... En >] ‘\n’

, where:

- **ng**: two characters that mark the start of a NG Command Line.
- −**command**: Points to which action must be performed at the destination through this command line.
- −−**alternative**: Selects one option for a command, customizing this action.
- **version**: Highlights the actual version of a command and alternative.
- **[...]**: Delimits the vectorial arguments that the action needs to consider at the destination.
- **<...>**: Delimits the arguments for a given action.
- **n**: Indicates the number of elements in an argument.
- **type**: Expresses the type of elements in an argument.
- **n type E1 E2 E3 E4 ... En**: Represents the elements of an argument.
- **‘\n’**: The End of Line character symbolizes the end of a command line.

Examples of NG Actions

This subchapter focuses on explaining the NG main actions for routing data and exposing resources.

- Forwarding/Routing Line:

  Usually, this is the first command line of every NG message, once every data content has a inter- or intra-domain communication goal. In this way, the NG architecture establishes this pattern for a routing/forwarding primitive that generally presents three arguments, with two tuples to identify the source and destination [46].

  ng −m −−cl 0.1 [< Domain Arguments > < Source IDs> < Destination ID] ‘\n’

  , where:

  - −**m**: The ‘−m’
  - −−**cl**: The ‘−−cl’
  - **0.1**: The action version of the Forwarding/Routing Line.
  - **Domain Arguments**: This parameter delimits the communication domain, in which the data exchange happens.
  - **Source IDs**: Represents the Source’s SVNes values.

– **Destination IDs**: Represents the Destination's SVNes values.

There are four SVNes values that represent host traits when considering intra-domain applications. The first value represents the Host Identifier (HID), a value that identifies the source or destination host for a NG message. The second points to the Operational System Identifier (OSID), an identifier for the OS. The third aims at the Process Identifier (PID), a process that generates or consumes the NG message. Lastly, the Block Identifier (BID) is an SVN that specifies which NG internal component must receive or have generated a message. These four identifiers are crucial to establish any NG inter-host communication. Once a given NG peer can present several NG internal components, running in several distinct OSs, and performing distinct NG processes in the same HID hardware resource. Through these, NG can currently route any NG message based on this tuple of SVNs by software. To exemplify, consider this real example of a intra-domain communication for this line is represented bellow:

ng −m −−cl 0.1 [1 s 28FD4420 > < 4 s 0BD95286 ED12F3ED 7E764DC1 4D623F20 > < 4 s 0BD95286 ED12F3ED 342DD4C5 B8101939 >] '\n'

, where:

– **ng −m −−cl** : Represents the forwarding/routing command line.

– **< 1 s 28FD4420 >**: Limits the NG communication scope.

– **< 4 s 0BD95286 ED12F3ED 7E764DC1 4D623F20 >**: Presents a tuple that identifies the source's HID, the OSID, PID, and BID.

– **< 4 s 0BD95286 ED12F3ED 449B0B0C 6FDF0A76 >** : Identifies the tuple of the destination's HID, the OSID, PID, and BID.

In other words, this means that a NG message forwarding/routing must happen within a local domain 28FD4420, wherein the NG block 6FDF0A76 from the 7E764DC1 process in the ED12F3ED OS of the 0BD95286 host sends a message to the NG block 4D623F20 from the 449B0B0C process in the ED12F3ED OS of the 0BD95286 host.

Considering NG broadcast messages, the Destination IDs are filled with the "FFFFFFFF" value. This scheme indicates that the destination is every NG peer in the network.

- Hello Line

  Hello Command Lines exposes periodically the traits of a given entity [49]. This enables the NG Initialization procedure, wherein internal components exchanges their SVNs, for example the HID, OSID, and PID. This primitive reveals the SVNs related to the PGCS's internal GW, HT, and Core components [46]. Moreover, it also draws the

details of the link layer technology applied to the communication, the network interface at the OS, and the MAC address from the PGCS's host [46]. In addition, it also displays four SVNs from the PSS, allowing the destination to publish and subscribe data to the exposing peer. In general, the Hello Command Line presents the following structure:

> ng −hello −−xxx 0.y [< 6 s PGCS SVNs Host SVNs > < 4 s PSS SVNs >] '\n'

, where:

- −**hello**: Specifies the NG Hello command.
- −−**xxx**: Expresses the type of communication: "–ipc" designates a intra-host hello and "–ihc" for inter-host communication, exposing traits through PGCS to another NG host.
- **0.x**: Determines the version of the Hello Command Line which can be 0.1 or 0.2.
- **< 6 s PGCS SVNs Host SVNs >**: This parameter delimits six string values of SVNs that exposes the name bindings related to the internal PGCS's components and the PGCS's host.
- **4 s PSS SVNs**: This vector draws out four string SVNs values from the host PSS. This enables the destination to subscribe and publish data and name bindings directly to the exposing peer.

The following box presents a real example of a exchanged "Hello" Command Line:

> ng -hello –ihc 0.2 [< 6 s A4324A2D AB9B70B4 57ECEB4F Wi-Fi wlan0 ac:22:0b:c9:df:3b > < 4 s 0BD95286 ED12F3ED 8E8B52EC 7EA46815 >] '\n'

This primitive is the second version of Hello for inter-host communication. In this line, a peer is exposing its PGCS internal components SVNs in the argument vector: A4324A2D (GW), AB9B70B4 (HT), and 57ECEB4F (Core). Moreover, this same vector details that the host uses the Wi-Fi link layer technology, through its "wlan0" network interface with the "ac:22:0b:c9:df:3b" MAC address. Following this, the third argument points out four SVNs related to its PSS.

2.4.4 NovaGenesis Message Fragmentation

In order to achieve the convergence between NG and any legacy Link Layer technology, NG applies its three layers to handle the message processing since conceiving a NG message until its translation into any communication protocol. Figure 2.10 presents an example of the creation, adap-

tation, and shaping of a NG message into an Ethernet stack example.

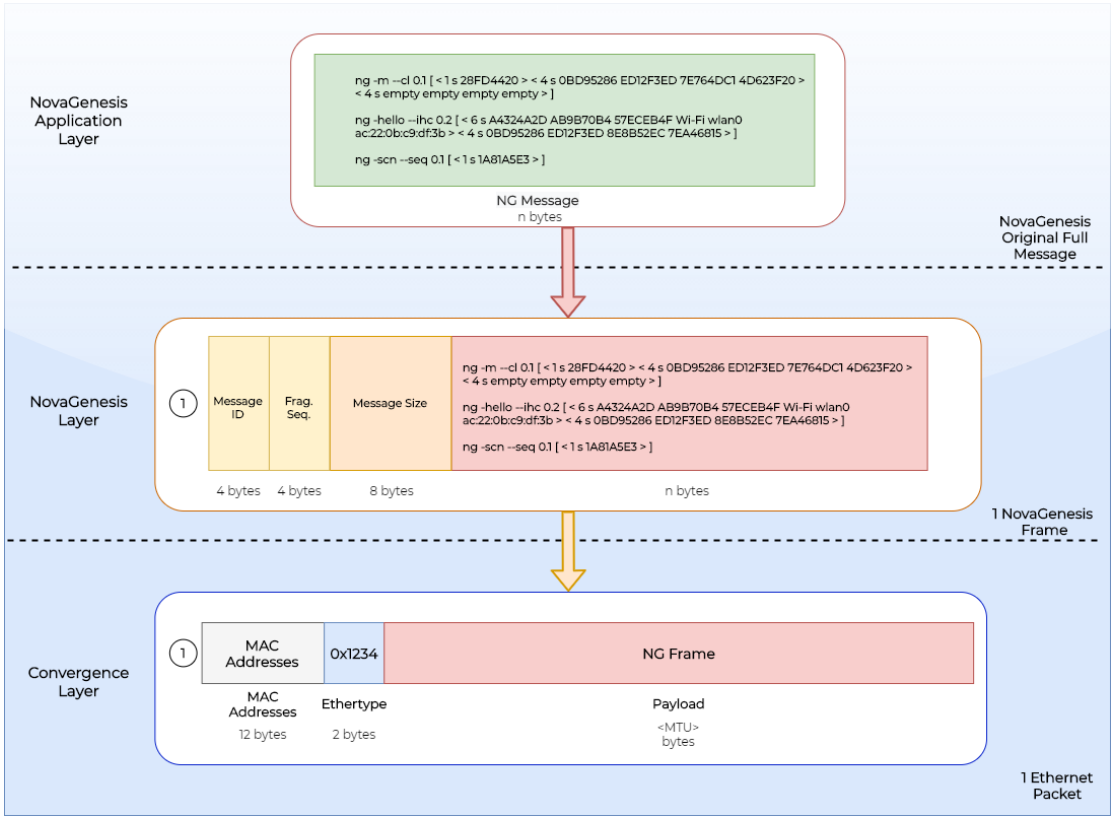

Figure 2.10: *Example of a full NG message.*

At the **NovaGenesis Application Layer**, NG conceives its messages to enable the data exchange. These messages act as the NG language standard that conveys the ontology between NG instances, processes, and peers. Each line is a NG command line that represents an NG action.

The **NovaGenesis Layer** receives the NG messages and adapts them to comply with the given legacy Link Layer technology parameters. This implies that it can fragment NG messages into smaller frames. Figure 2.11 exemplifies a hypothetical NG message fragmentation. Moreover, **NovaGenesis Layer** is crucial to enable the fragmentation and reassembling. This layer inserts a control header with the following fields:

- **Message Identifier** is a randomly generated number of 4 bytes. This field identifies each generated NG message and acts as an index to reassemble fragments. Consequently, every Message Identifier field of every fragment of a fragmented message presents the same value.
- **Fragment Sequence** is a 4 bytes values that convey the sequence of fragments generated. This value is incremented by a unit as fragments increases, starting from the 0 value. Through this field, the end peer can reassembly any NG fragmented message, even in cases of receiving random fragments in an unordered manner.

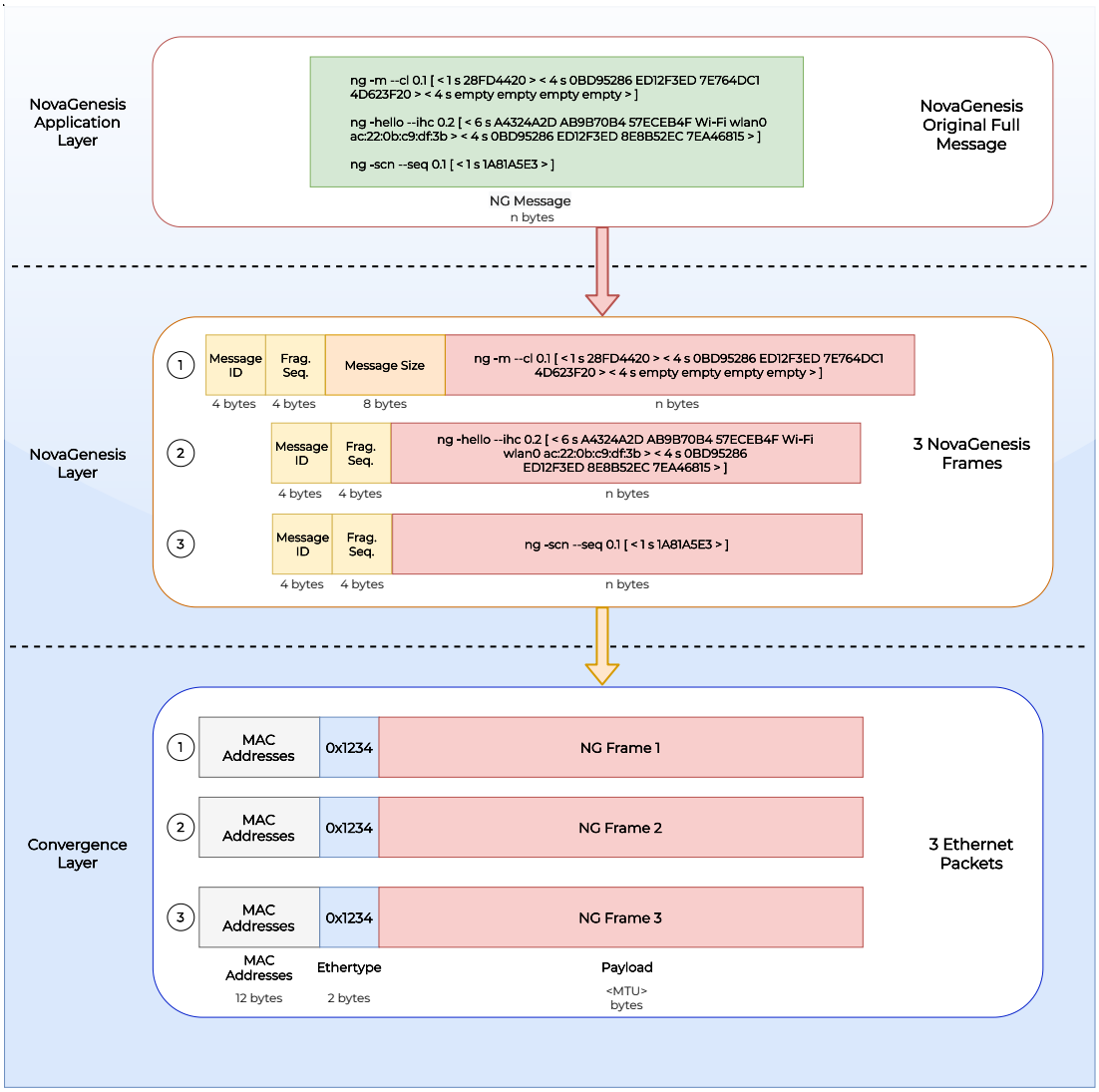

Figure 2.11: *Example of a fragmented NG message.*

- **Message Size** is an 8 bytes value that represents the full length of the original NG message. Each NG frame can present a variable size. This field is crucial to remount fragmented packets in the communication end, once it points the expected full size of the NG message.

In order to exemplify the fragmentation and reassembling of NG messages, Figure 2.12 illustrates the process of data exchange between Host A and Host B with an unordered receiving packet process. Notice that Host A's generates a "Hello" message to expose its traits to the Host B. To conform this message, NG Layer fragments the 273 bytes into 3 frames. At this moment, it inserts the control header for re-assembling at Host B. The **Message ID** is the hexadecimal value of 0x9C4, the **Fragment Sequence** is incremented as new frames are being conceived, and **Message Size** depicts the full length of 273 bytes or 0x111. After this, the Convergence Layer parses each NG frame into the desired communicating protocol

stack. In this example, we are considering an Ethernet stack. Therefore, this last layer inserts the required header fields of Ethernet, such as the MAC addresses and Ethertype, to conform each NG frame to this standard.

For some unknown reason, consider that the 3 Ethernet packets sent by Host A are received unordered by Host B. In this hypothetical example, Host B receives fragment 3, fragment 1, and fragment 2, respectively. As Host B receives each fragment in distinct moments, NG Convergence Layer first parses each Ethernet packet. At this primary Layer, NG considers the Ethernet stack standard to take NG fragments from its payload up.

After translating the received packet into frames, the resulted frame are processed by the NG Layer based on its control headers. Through the **Message ID**, **Fragment Sequence**, and **Message Size**, NG remounts the fragments into the full NG message. This layer assigns a buffer with the **Message Size** to mount the full NG message based on the Message ID header field. This buffer is filled with each received packet's payload, ordering them by the Fragment Sequence field. After mounting the full NG message, NG Layer processes the “hello" message.

2.4.5 Closing Remarks

NovaGenesis is a convergent information architecture focused on a multi-strategy approach. As seen, we can relate its concepts with other proposals like ICN, SCN, SOA, digital twins, and others. This is an architecture that supports protocol implementation as services, mobility, unlimited namespaces, contract-based operation, semantic rich self-organization, and so on.

Through its self-similar and self-organizing structure, basic and specialized services composes mostly any NG application and its protocol can be converted into any legacy technology. Each entity can present an identifier if it has a unique name in a given domain, which can be represented in NLN or SVN convention. Each entity has a well-defined life cycle to act within a NG domain, covering from NG Bootstrapping until its regular operation. This characteristic yields in two distinctive NG operation phases: initialization and operation. During the initialization, NG hosts create its domain with the desired traits and expose its SVNes to other NG peers, offering a service. Whenever a resource is found in the network, the NG players can establish SLAs to negotiate the terms of a cooperation.

Finally, the operation phase starts once the hosts have established a SLA. After they fulfill the agreed task, NG follows to disclosure and forbid their cooperation once their partnership is completed. In latter states, the own service's dismissal is considered in the NG Life Cycle.

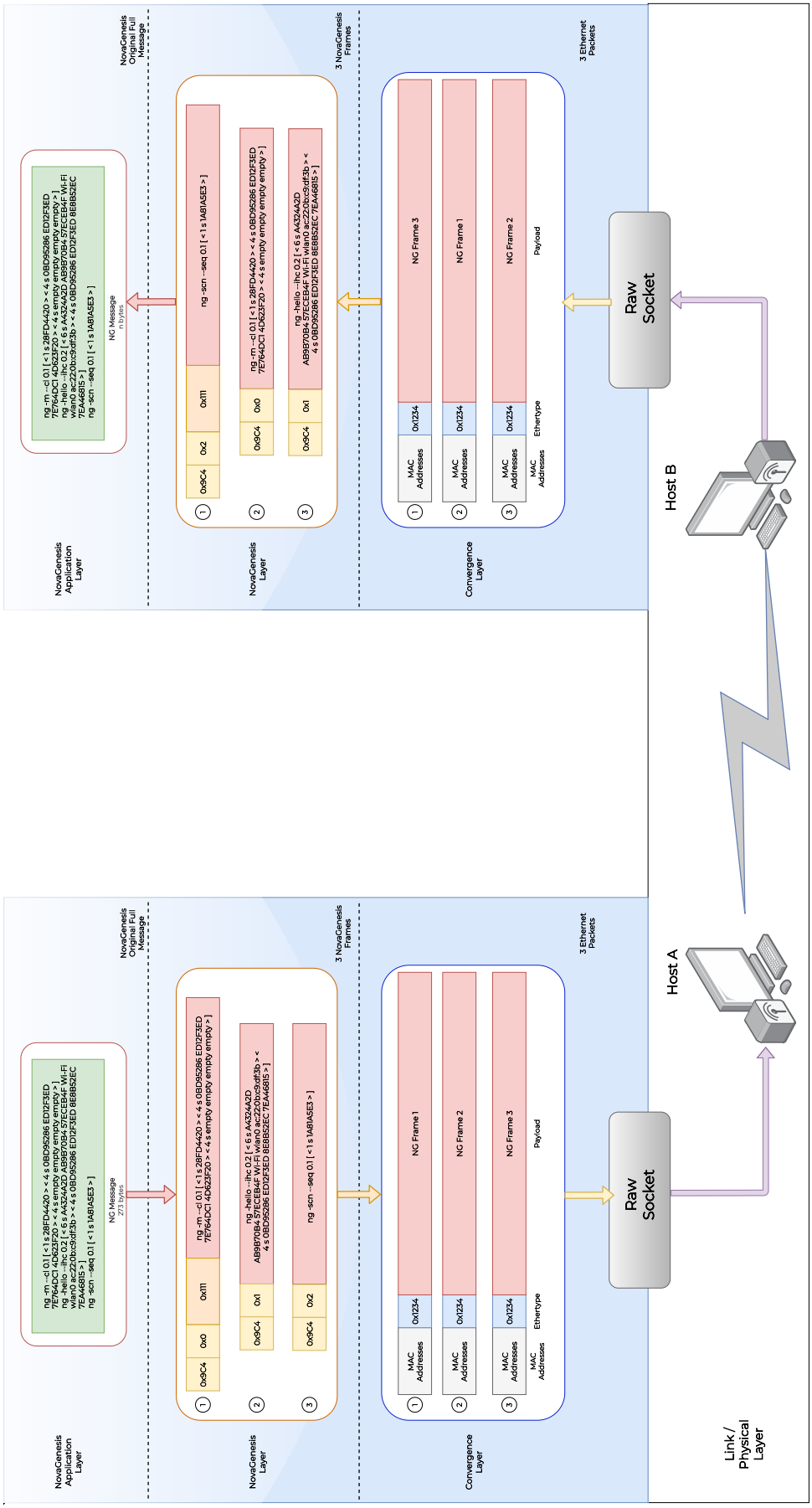

Figure 2.12: *Example of the fragmentation and reassembling of a NG message.*

Chapter 3

Related Work

THE Internet is a landmark of science. This infrastructure has shortened distances and fostered new models of society. Nonetheless, contemporary trends stress the current Internet core, hindering its full potential. On other hand, revolutionary proposals remodels the Internet from scratch, using clean-slate architectures to address the current Internet challenges.

In this aspect, this chapter first approaches a research methodology to carry out a literature review, highlighting the search strategy and the articles' selection. Based on this, synergistic FIA works that have explored SDN are explored to contrast with NG architecture. Finally, the final considerations are presented, summarizing this chapter, and presenting the current convergence of FIAs and SDN state of the art.

3.1 Research Methodology

This chapter focuses on analyzing FIA proposals that explore SDN. In this way, it bases on a systematic literature review that follows the given procedure:

- *Research Questions*: Questions that guide the research and selection of articles to compose this chapter;
- *Search Strategy*: It presents the adopted strategy and the data collection sources;
- *Article Selection*: Displays the criteria used to select the articles;
- *Quality Assessment*: It highlights a filter performed to determine the general quality of the selected related works.

3.1.1 Research Questions

The FIA field presents several projects to remodel the Internet under clean-slate proposals. Given this collection, many of these projects were

supported by funds earmarked by private and governmental research funds. For example, we can mention programs promoted by the North American National Science Foundation (NSF) and by the European Union. Of the latter, the Framework Program 7, Horizon 2020, and Horizon Europe stand out. Nonetheless, several of these proposals have been abandoned when the research funding is concluded and, thus, the financial resources or any other reason to study them have became scarce.

Given the impact of this research area, research questions filter the research range, narrowing it to the FIA works that have covered SDN. Therefore, two types of research questions are presented: the General Research Questions (GRQ) and the Specific Research Questions (SRQ).

Contemplating this methodology, Table 3.1 outlines the research questions. Firstly, GRQs aim to narrow FIA projects. Firstly, GRQ1 selects the FIA that are harmonious with the NovaGenesis architecture and have explored SDN. After this, GRQ2 focuses on GRQ1's architecture structural principles relevant to SDN. On the other hand, SRQs further restrict the scope of research by proposing a systematic literature review of architectures that explore or have explored SDN technology. SRQ1 analyzes the convergence of FIA and SDN, highlighting each related work and discussing its methodology to fulfill its goal. Finally, SRQ2 characterizes, if possible, each proposal's control agent assumptions.

Table 3.1: *Research Questions.*

Group and Identifier	**Issue**
General Research Questions	
GRQ1	Given the area of FIA, what are the most synergistic proposals to the NovaGenesis architecture and have explored SDN?
GRQ2	Through the GRQ1 filter, what are the operating principles of these architectures?
Specific Research Questions	
SRQ1	Based on the FIAs collection given by GRQs, how SDN is applied in their design? What are the tools, technologies, and methodology used in these related works?
SRQ2	If possible, what are the main control agents' assumptions to oversee an SDN network infrastructure?
SRQ3	Based on the related work, what are the main takeaways and research opportunities for FIA and SDN convergence?

3.1.2 Search Strategy

Keywords were associated with scientific databases to perform this research. Therefore, the present study bases on terms such as: FIA, SDN, P4, OpenFlow, and other related keywords. These words were logically combined to compose this data collection. Concerning the data sources, the *IEEE Xplore* from Institute of Electrical and Electronics Engineers (IEEE), *ScienceDirect* from Elsevier and *ACM Digital Library* from Association for Computing Machinery (ACM) digital repositories were consulted. In these

websites, the keywords combination narrows the search and enables selecting the most relevant works associated with this work.

3.1.3 Article Selection

Through this methodology, relevant articles were found in the desired area. Moreover, only the works with more than 4 pages were considered for this study, ranging from periodicals, conferences, and workshops.

3.1.4 Quality Assessment

The quality standard covers a superficial analysis of the selected articles. At this point, the abstract, introduction, and conclusion sections were considered to determine the relevance of each proposal. If there was potential, the rest of the article was contemplated to provide material for this chapter.

Before going into the literature review, someone can note that not every covered proposal presents the tools used, lacking the data to understand what its authors employ. In other words, some do not specify the technologies, architectures, or even the programming language. In the same way, some do not have the future works section. Hence, it is not known whether a project continued or not. Some attempts to search for new articles by the authors' names were made, yet it has not presented satisfactory results.

Even so, the explained filter made it possible to find 4 FIA harmonious to NovaGenesis. Through this search, 3 projects are relevant to the current scientific community and present continuous work in different areas and technological trends. As for the proposals that explore the convergence of Internet and SDN architectures, the search has found 8 works. From this total, 5 seek to optimize the communication infrastructure using SDN technology, while 3 seek the interoperation of architectures.

3.2 GRQ1 and GRQ2: FIAs and their Principles

This sub-chapter presents the alternative Future Internet Architectures to NovaGenesis, answering the GRQ1. Moreover, it presents each proposal principles that relates the most with NG, addressing GRQ2.

3.2.1 Named Data Network

Disrupting from the host-centric paradigm, Named Data Network (NDN) proposes that data must drive any communication. In this view, this FIA is an example of ICN, wherein named data is the cornerstone of a data-centric design [54]. In its design, NDN packets address content objects rather than communication endpoints. In other words, every communicating entity

identifies the desired content and does not a server's address, like in IP [55]. Moreover, NDN's naming structure encompasses everything in its global hierarchical scope, from raw contents to endpoints. To cite some of its benefits, NDN improves security, confidentiality, traceability, and anonymity through named data signatures. This architecture also dynamically self-regulates the network traffic by its stateful communication scheme [54–56]. Besides that, NDN might be the most important contemporary example of an ICN, being founded by the U.S. National Science Foundation under its Future Internet Architecture Program [54].

Focusing on its communication scheme, NDN applications drive the data-pulling from the network [56, 57]. Moreover, it enables effortlessly in-networking cache and multicast data delivery, once NDN routers apply stateful tables to forward packets throughout the network to meet requests [56–58]. Therefore, the architecture reacts wisely to sudden network problems by applying a forwarding strategy in a hop-by-hop manner. In this way, it can avoid network congestion and link failures while optimizing the overall QoS related to round-trip time, throughput, and packet loss.

Concerning NDN forwarding structure, it bases on only two types of packets: Interest and Data [57]. Whenever a receiver (originally, an application) wants content, it issues an Interest to the network. In this datagram, the only required field is the unique content's name [54, 59]. This name follows a hierarchical structure that expresses the relationship between the data and the network elements. For example, a picture called "Inatel.jpg" from the ICT Lab stored in a server at Inatel can present the following NDN name structure "*/inatel/ictlab/Inatel.jpg*." Based on this definition, every forwarding element analyzes the domain boundaries to fulfill the request.

Focusing on the NDN stateful forwarding scheme, it presents 3 structural components called Content Storage (CS), Pending Interest Table (PIT), and Forwarding Interest Base (FIB). Alongside a Forwarding Strategy, NDN enables the adaptive routing per hop. The list below presents further details about these routing blocks, as well as the forwarding strategy:

- **PIT** stores all the Interests that a router has forwarded, but not satisfied yet [54]. This table records the requested content and the incoming and outgoing interfaces that received/forwarded the Interest.
- **FIB** bases on an adaptive Forwarding Strategy when PIT does not present an entry for a received Interest. This strategy can encompass any routing or self-learning scheme that fills the FIB [58].
- **CS** is a temporary network cache, wherein producers store content.
- **Forwarding Strategy** manages the Interest packets, forwarding or dropping them.

When an Interest reaches an NDN router, this device first checks if the desired content name exists in the CS to fulfill or forward the request. If a

match happens, this network node forwards a Data packet on the Interest incoming interface, satisfying the request [54]. Otherwise, it needs to forward to another network node that may fulfill the Interest request [58]. In this case, it adds a new entry for the requested data in its PIT with a determined lifetime and demands an action from FIB. This block verifies which is the best path to forward that Interest packet based on the *Forwarding Strategy*. If the forwarding module detects that all links are congested or any other anomaly, it probably drops the given Interest packet [54]. On different occasions, it validates the Interest, forwarding the Interest based on the best effort, i.e. the strategy retrieves the longest prefix known in the FIB and decides how, when, and where to forward the received packet. When the PIT entry lifetime expires, it removes the associated data. Figure 3.1 depicts this upstream Interest flow.

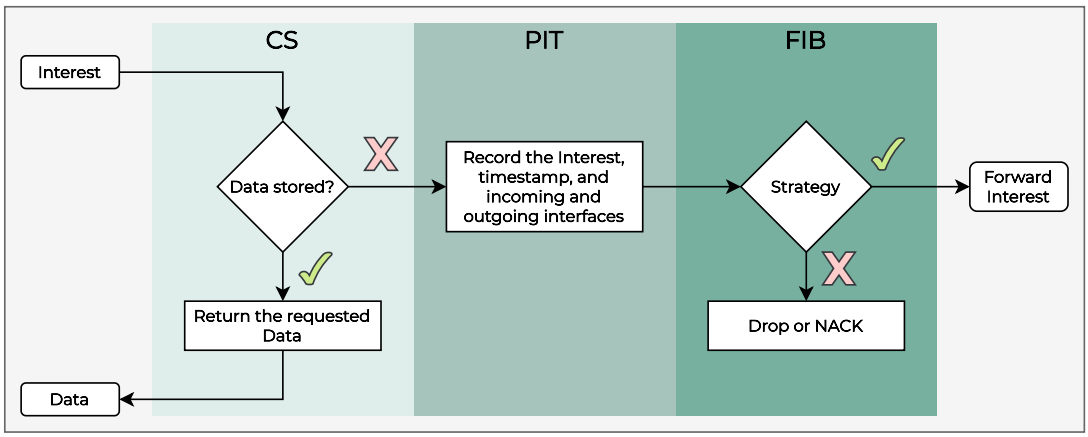

Figure 3.1: *NDN Upstream Communication Model. Adapted from [54,58].*

The Interest packet continues flowing until it reaches a server with the required data or a router with the content cached. Upon finding this, the server issues a Data packet that backtracks the Interest path [58]. This reverse path is only possible due to the PIT in each NDN forwarding element that had recorded the Interest's incoming interface. When a Data packet reaches these elements, the device checks its PIT to determine if it knows how to forward this packet back to its consumer. At this moment, the forwarding element can discard the Data packet if the device does not find any entry in its PIT [54]. Otherwise, it removes the unsatisfied Interest entry from its FIB, stores the related data in its CS temporarily for further requests, and forwards it to the next-hop closer to the requester. Figure 3.2 illustrates the downstream path of a Data packet to reach its consumer.

Through this communication scheme and the symmetry between Data and Interest packets, NDN grants a stateful, intelligent, and adaptive forwarding data plane [56]. Moreover, it can optimize the quality of data exchange by making contents more available in distributed caches throughout the network while increasing its responsiveness upon supervising the PIT's entries lifetime and Round Trip Time (RTT) between the Interest and Data

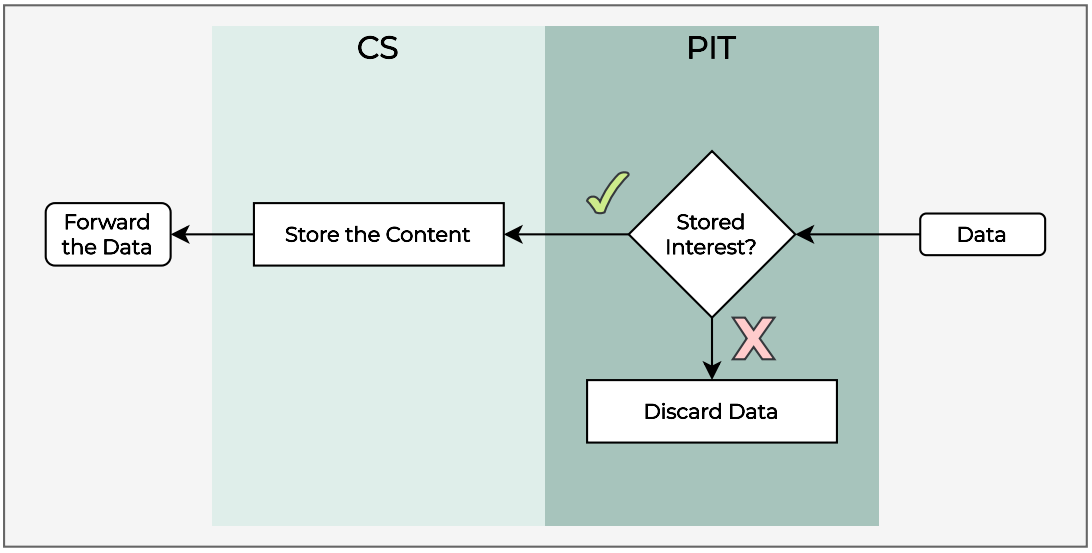

Figure 3.2: *NDN Downstream Communication Model. Adapted from [54,58].*

packets [54]. These two metrics also aid in discovering packet losses, self-configuring the best paths if Data does not come back in the expected time.

On the other hand, the naming structure ensures networking independence, enabling its evolvability once the names are decoupled from the network itself [57]. This design choice contrasts with the current Internet design, in which TCP/IP is strongly bound to how the packets flow. In this way, NDN is a better fit for future applications, where heterogeneous networks and technologies may connect different domains. Additionally, it avoids address space exhaustion and boosts address management. Once a name is unique globally, the NDN can explore dynamical routes to fulfill any request and provides better support for multicast, mobility, and delay-tolerant applications [56, 58]. NDN has rethought the current Internet and some of its published works has included research on IoT [60–62], Vehicular Networks [63–65], and Big Data [66–68].

3.2.2 MobilityFirst

MobilityFirst (MF) is a FIA project that rethinks the current Internet by proposing a design centered on the mobility of entities, as well as on supporting wireless connections [69, 70]. Hence, this clean-slate ICN architecture focuses on optimizing the security, reliability, and management of services provided by large-scale mobile devices, applications, and networks [71]. This focus bases on studies that point to the predominance of this niche in the communication networks of the future, which makes sense considering that we see examples of technologies that encompasses IoT, autonomous cars, and other smart ecosystems [72]. Therefore, it is reasonable to predict that fixed devices, such as personal computers and servers, will be a quantitative minority [69].

MF has as pillars the concept of ID/Loc splitting to link network objects to their domains [69]. Through this, it guarantees not only the mobility of entities but also the scalability of the entire network, encouraging communication paradigms such as multicast, anycast, multi-path, and context-aware services [71, 73]. As a result, network objects need to have unique and location-independent names. Thus, MF proposes public keys, or Globally Unique IDentifiers (GUIDs), of 160 bytes based on SVNs are linked to each existence [73]. These, in turn, are mapped to sets of routable network addresses, or Network Addresses (NAs), by Global Name Resolution Service (GNRS) (similar to traditional DNS). If there are copies of the same content on the network, i.e. GUIDs identical, the relationship of each replica with its domain draws the distinction of content, i.e. NAs distinct [73]. Thus, large-scale routing is guaranteed and effective.

As for routing, MF natively provides a separate data management plan that provides visibility of the [69] network. Specifically, the resolution of GUIDs into NAs enables the data routing, which is accessible to any network element through the GNRS service. Therefore, the source must specify at least the destination GUID to point to a destination [73]. As a consequence, the destination GUID is resolved at each router node to find the location of a given content [73]. Each router has information about the network graph, the quality of the links around it, and the connection probabilities of adjacent nodes. As for the data itself, it can be sent via large data blocks between nodes, which can also be stored in the network cache to optimize transmission, content accessibility, and delay through the routing scheme called Generalized STorage-Aware Routing (GSTAR).

As seen, MF is a clean-slate FIA centered on [69] mobility. To catalize such a vision, it establishes robust support for wireless, security, and privacy to ensure data transmission and reliability, management, and scalability of the network as a whole [69]. Therefore, mobile entities connect seamlessly through a large-scale infrastructure with fluctuations in the quality of wireless links [69]. Thus, its purpose expands to simple smartphone-Internet integration but also encompasses wireless peer-to-peer connections, vehicular networks, and machine-to-machine applications [70, 71].

3.2.3 PURSUIT

Another proposal is Publish Subscribe Internet Technology (PURSUIT), an ICN continuation of the Publish Subscribe Internet Routing Paradigm (PSIRP) project funded by the European FP7 program [74]. This clean-slate architecture shifts the focus from the send-receive to the publish/subscribe (pub-sub) communication paradigm. In this way, this FIA conceives a novel network core to enable this form of communication where there are publishers, subscribers, and network brokers. Briefly, the following list explains the role of each agent in pub-sub [74] communication:

- **Publishers** are entities that own the data and periodically disclose the data availability through publication messages.
- **Subscribers** are entities that search the network for information, request the data when they detect a publication message, and send a subscribe message to signal interest in such content.
- **Network Brokers** are analogous to network routers, having the purpose of forwarding pub-sub messages and the requested content along with the network, making the meeting of the 2 players.

Through these elements, PURSUIT can natively support contemporary types of communication such as multicast, anycast, multihoming [75]. Furthermore, it also bases on ICN pillars, providing a robust, reliable, and secure communication system with the possibility of replicating content over the network through network caching and traceability of mobile entities. As for its naming, each object has 2 IDs that specify the scope (SID) and rendezvous (RID) of each point [76]. Meanwhile, the scope ID names a domain, where it groups information objects. The rendezvous ID would be analogous to the address to obtain unique content on the network.

As for the PURSUIT routing, it relies on the SID and RID. Name resolution works as a hierarchical DHT through the Rendezvous Network, composed by a set of Rendezvous Nodes (RNes) [74]. When a publisher issues a publish message, the local RN forwards this packet through the DHT to the desired SID. Thus, this message must specify the destination SID to where it wants to publish its content [75]. On the other hand, subscription messages present the desired content SID and RID, which similarly enables the routing as the publish messages [76]. There are other functions and structural mechanisms that aid the rendezvous, routing, and name resolution function, but these are beyond the scope of this work.

3.2.4 RINA

The Recursive InterNetwork Architecture (RINA) was conceived in 2010 by computer scientist John Day [77]. This FIA proposal remodels the Internet from scratch by considering that any form of interaction over a computer network is based solely and exclusively on Inter-Process Communication (IPC) [78, 79]. Thus, the network is responsible for making possible the communication of applications in distributed systems, integrating different solutions to solve intrinsic problems of the current Internet [80]. Furthermore, each application service or process has globally unique names [78].

The clean-slate RINA creates recursive layers called Distributed IPC Facilitys (DIFs) which are combined to provide functionality needed for the network [81]. Each layer is self-contained, creating a substrate that provides resources specific to layers and higher applications [80]. For example, the network might create one DIFs to provide data transport, one higher

than the first to ensure security, and one higher up to manage traffic and avoid data congestion. In this way, each level or layer DIF shares the same protocols to perform a service but may have different policies to optimize the regional QoS. On the other hand, each DIF also encompasses several entities to perform a certain function [82].

Considering inter-domain communications, RINA implements distributed applications called Inter-DIF Directorys (IDDs) that are responsible for searching for entities outside the scope of a DIF [79]. At this moment, an IDD queries IDDs of DIFs neighbors until it finds an application or meets a pre-established condition to find the best possible path. When this happens, the IDD can establish a connection between the source and the destination, either through a new dedicated DIF or by expanding an existing one. In this way, applications start to communicate directly through a service distributed between IPCs processes [80]. Thus, data routing is performed at each hop, using the DIF created or the underlying DIFs [78]. Through this structure, the naming and addressing become relative to the context of the DIF, with a node identifier being considered a name for IPCs of DIFs lower and representing an address to the same time when viewed by an IPC from a layer above the target node [78].

Meanwhile, IDD is a distributed application or Distributed Application Facility (DAF). In other words, a DAF is a set of two or more processes of an application distributed in two or more processing systems in RINA. These DAFs exchange data for their respective IPCs [79]. Furthermore, Distributed Aplication Processs (DAPs) are used to provide point-to-point connectivity between applications, being responsible for discovering and making available applications throughout the network. To this end, the DAPs has two routing tables that help in finding the IDD destination. One of these tables is called *Neighboor Table*, which registers the nearest neighbors, and the second is *Search Table*, intended to search how to reach the destination of the application. Thus, the IDD DAPs of the same DAF exchange discovery messages [79]. Figure 3.3 illustrates this interaction of the framework for routing and application discovery of the RINA architecture, where *app 1* makes use of IDD DAF 1 to reach *app 2*. Besides, note that DIFs 1 and 2 partially connect each host.

In summary, clean-slate RINA proposes to rethink how the Internet infrastructure could be conceived contemporarily. By shifting the focus from host-centrism, this architecture develops all its pillars to support IPC and support all its applications recursively. In this way, RINA encourages the implementation of networks with heterogeneous technologies, simply creating DIFs that promote their integration with the infrastructure. Furthermore, it fosters the mobility of entities without loss of identity by using this ambiguous naming and addressing structure. When having a mobile entity, it decouples from its DIF and connects through a new one, being promoted

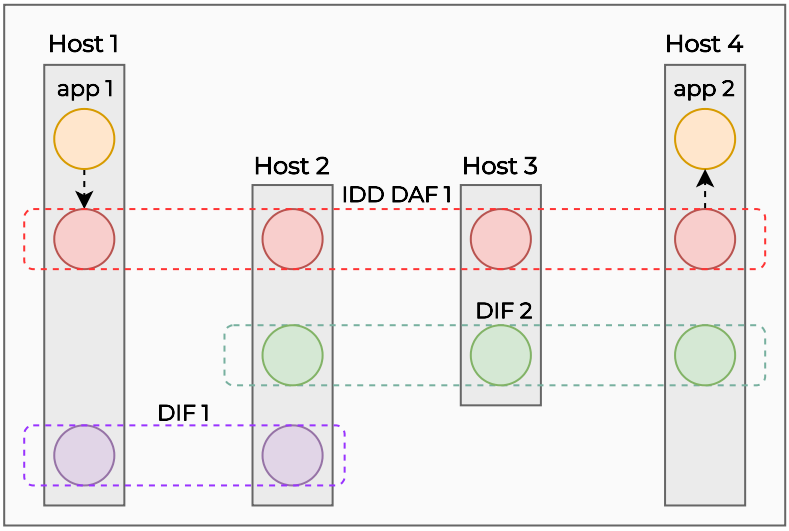

Figure 3.3: *RINA communication model. Adapted from [79].*

by the communication and routing structure of the architecture.

3.3 SRQ1: FIA and SDN convergence

This sub-chapter focuses on the related works involving FIA with SDN, exploring proposals to enable a given FIA data exchange or interoperation.

3.3.1 *ICN.P4*

Focusing on the video traffic advance in the current Internet, Feng et al. [83] aims at creating a cooperative framework between Hypertext Transfer Protocol (HTTP) and ICN protocols. The project's goal is to evaluate the network performance and interoperation between the ICN and the HTTP protocols. Through this, they can optimize the current HTTP transmission by decoupling it from the current TCP/IP architecture. Taking advantage of the ICN infrastructure and its benefits, the authors replace the IP protocol with a scheme that ensures the ICN content delivery.

In this proposal, the authors created a P4-based Switches (SWs) capable of exchanging ICN packets. Based on distinct forwarding actions, this novel HW replicates the ICN structure, providing customized CS, FIB, and other ICN components. Focusing on the synergy between architectures, ICN's Interest packets are similar to the HTTP's request and the Data packet to its response [58]. Alongside the P4 SW, a custom proxy agent converts the HTTP protocol into ICN before transmitting the packets into the network. Hence, both server and client hosts must present a proxy application before properly transferring packets in the network. Following, the in-network data flow takes the form of ICN custom packets for routing while ensuring

the inter-host communication with the traits of the HTTP protocol.

Regarding the network communicating protocol, the authors have proposed a protocol with headers to aid P4 forwarding at intermediate nodes that are not P4-based. In this way, the packet exchanged is a hybrid protocol that considers both the TCP/IP and ICN stack. Concerning the HW, all of this proposal's infrastructure exploits Virtual Machines (VMs) to validate the proposal on Ubuntu 16.04. Through this, the authors point a better transmission efficiency and decrease of redundant traffic in the network.

3.3.2 NDN.p4

Signorello et al. [84] created a P4 based SW capable of matching NDN Interest and Data packets called *NDN.p4* [1]. Upon parsing an incoming Interest or Data packet, P4 looks up at the P4 action tables to manipulate the simulated FIB and PIT tables, which replicates NDN forwarding scheme as P4 registers. Nonetheless, it lacks the CS structure in the SW. The authors deploy this SW in a infrastructure with Mininet hosts capable of using the NDN applications called *ndnpeek* and *ndnpoke*.

NDN.p4 parser structure follows a finite-state machine algorithm that analyzes the incoming packet. The SW considers fixed values of Type Length Value, i.e. the payload of each NDN packet with its header and payload; a maximum number of NDN name components to route, i.e. a name component is a portion of the content name; and six match-action tables to act based on the Longest-Prefix Matching situation. Based on the number of name components, the SW selects the correct known FIB table entry. In the case of not matching the packet full name, the Longest-Prefix Matching happens, in which the SW forwards the Interest based on the best registered, or longest, entry and adding a new entry in the PIT with the incoming ingress port. On the other hand, the SW matches the Data content name with its PIT to provide the reverse route to its consumer.

Concerning its Control Plane, the authors employ a custom application to interact with the BMv2 framework on the fly and manage the entries in the underlying SW. As conclusions and lessons learned, the authors point that the current P4 must consider carefully how to expand its framework, once it abstracts the Data Plane functionalities with few native features.

3.3.3 Enhanced NDN

Differently from NDN.p4 [84], Enhanced NDN (ENDN) aims at setting the data plane for selected NDN content delivery services [85]. This architecture overcomes the P4 limitations of dealing with string-based content and updating the entities source code without losing the infrastructure

[1] https://github.com/signorello

service. The designed architecture creates network slices by isolating P4 programs according to their functions, improving the dynamical network programmability. This means that ENDN takes advantages of several P4 functions that run in isolation per target.

As SDN cornerstone, ENDN detaches the control plane from the data plane. The first focuses on providing the catalog of content delivering services, adapting the latter through setting up each network SWs to fulfill a request. The evaluated services are: (i) Content Delivery Pattern, which delivers content in a namespace; (ii) Content Name Rewrite, which changes the content name of packets at specific network locations upon an application request; (iii) Adaptive Forwarding, which defines how routing happens at particular network locations; (iv) Customized Monitoring, which notifies the application of distinguished events; (v) In-Network Cache Control, which manages the cached content of Data packets; and (vi) Namespace Traffic Management, which specifies the requirements of a namespace. Moreover, ENDN achieves complex network behaviors by the combination of these services. The control plane sets the data plane upon receiving an application definition of a namespace and associated set of services.

For evaluation reasons, ENDN exploits a customized NDN Forwarding Daemon, a BMv2 model to conceive each P4 target, the P4 16 version, and *ndnSIM* simulator creates two test network topologies. In the first, a distributed data center manages the access certification of consumers. This case enables a contrast between ENDN and NDN, in which two SWs connects three network regions, granting connectivity among consumers and producers. In the second, the authors investigate ENDN adaptability through a topology that connects consumers and producers. Each network region has a border SW, interconnected by a central network link. Moreover, ENDN handles several applications with divergent requirements that may overload the central network link.

As a result of these evaluations, the authors draw some limitations in terms of northbound interface, i.e. the connection between data and control planes, scalability and resource management, consistency of settings, and overall ENDN security. Regardless of this, ENDN seems to achieve enough network efficiency with low latency, overcoming P4 string-based content limitation and providing a substantial set of customizable NDN services.

3.3.4 RINA and P4

Looking ahead in the future of communication networks, RINA aims at high-performance applications, such as 5G networks, communication service providers, and data centers. Nonetheless, the project needs to improve its infrastructure to deploy high-performance routers to accommodate such demand with considerable flexibility, throughput, and reduced

latency. Therefore, a path to enable this vision is to explore SDN devices to replace the standard RINA interior and border routers. In this way, these devices' programmability can customize future RINA prototypes. Gimenez et al. [86] design the first RINA interior router based on P4.

Following this proposal, the authors developed a P4 SW using the P416 version, ready to parse the RINA's data transfer protocol, Ethernet, VLAN, and IPv4 protocols. Hence, RINA and IP architectures exploit the Ethernet medium to exchange packets between two hosts and a BMv2 simple SW target. Focusing on the P4 device, it matches actions based on specific P4 action tables and reconstructs the packets as specified in the P4 program.

On the other hand, the Control Plane employs the Stratum Network OS, the P4Runtime framework, and a python script. The first one is a framework capable of managing and monitoring the hardware with support to multiple vendors. Moreover, Stratum deploys the gRPC Network Management Interface (gNMI) and gRPC Network Operations Interface (gNOI) that foster the device's programmability and streaming telemetry. Specifically, Stratum is a novel tool that eases the integration between P4 and P4Runtime, handing the required mechanisms to configure, control, and manage the P4 pipeline. In addition, the P4Runtime framework establishes primitives to control and configure each P4 target in the Data Plane. Lastly, the Python control interacts directly with the P4 Runtime API to open a session with the P4 Runtime API server, loading the necessary configurations.

After designing this proposal, the authors evaluated the device through Docker containers and Mininet network simulator. In both cases, the containers deploy Mininet, creating a network with two hosts and a BMv2/ Stratum SW to interconnect two hosts. Therefore, the SW handles the data exchange from either EFCP or IP packets. In order to validate the proposal, the work proposes connectivity and performance tests. The first focuses on exchanging EFCP and IP packets between the hosts, while the latter evaluates the throughput and the latency. Concerning the computational resources, the authors exploit a Linux VM with 2 GB of RAM and 2 CPU cores and an Amazon WS VM with unspecified hardware settings to deploy their environment. Through their results, the proof of concept is feasible. The authors also point to a future work where they will explore the open source Investigating RINA as an Alternative to TCP/IP (IRATI) RINA implementation [87], P4-Runtime API, and the Stratum Network Operating System to design and prototype a RINA border router.

3.3.5 P4 with TCP/IP

TCP/IP has become inadequate for handling some current disruptive technologies that diverge from its model. As an example, Baktir et al. [88] points to novel service-centric models, where smart contracts and devices

establish services and micro-services through the network. In the current network architecture, its premises compromise the orchestration of such dynamical operations. Therefore, the authors have created a P4 environment that eases the deployment of service-centric approaches with TCP/IP. Through P4, they can exploit a fully programmable environment that coordinates the network behavior customization on the fly.

In their proposal, the P4 devices identify the services and micro-services through two different 32-bit header fields in the Ethernet packet. Based on this, the authors design the P4 SW to handle TCP/IP, Ethernet, and UDP protocol stacks through four customized match/action flow tables by the parsed packets. Besides, they leverage the P4 Runtime API with the BMv2 to manage the entries at the routing tables and required registers. In sum, the goal is to route the incoming packets following a service-centric model that deals with the IP addresses of the destinations.

To evaluate their approach, the authors use Mininet to simulate a face recognition service. In this scenario, a P4 SW connects the face recognition service and has to deliver the packets to one of three available servers. Moreover, the programmable device must provide the reverse path from the server to the face recognition application. Nonetheless, the published work [88] does not addresses some metrics to evaluate its performance. Therefore, it seems more like a proof of concept in development.

3.3.6 Future Internet Fusion

Considering what the current Internet represents to society, replacing it for another architecture seems unlikely. The full replacement for a new architecture that solves everything may even be unfeasible because the current stack has been proved over time. Considering this, current efforts focus on creating a heterogeneous network, in which the cooperation between disjointed architectures foster the Internet of the future.

One example that have focused on the interoperability of different architectures is Future Internet Fusion (FIFu) [89]. In this work, a transparent framework has supported the inter-communication between different architectures' resources. Besides, it has provided backward compatibility with legacy technologies and global access to resources based on the project's programmability and flexibility. Moreover, FIFu has two-layer that compose its data and control planes, where a set of mechanisms enable the cross-mapping of resources and deliver any request of the focused architectures. At the time of the [89] publication, the authors focused on the current IP and two ICN cases in the form of NDN and PURSUIT architectures. Through these projects, the authors evaluate their proposal's performance under three distinct scenarios, namely, a web-browsing application, a live video streaming, and an on-demand video streaming.

Regarding FIFu architecture, the Adaptation Layer represents the data plane, wherein the network equipment acts forwarding and routing the data packets between isolated network clouds. In this, there are gateways called FIXPs that convert messages from/to those network clouds and emulate a communication endpoint. In other words, the FIXPs' role is to make the incoming packets compatible with its destination network architecture, regardless of its original nature. In broad, each FIXP has the required information to format each packet according to its destination architecture, including the semantic meaning of each protocol field and metadata.

On the other hand, the Intelligent Layer implements Future Internet Controller (FICs) that manages the underlying FIXPs. This layer is the FIFu's equivalent to the control plane. Each FIC in is charge of setting, managing, and supporting the FIXP operation, measuring their performance. Moreover, each FIC stores, manages and maintains an identifier for each resource of each network architecture. In this way, the Intelligent Layer has a list of the underlying FIXP topology and a repository of mappings between identifiers to assist the FIXP operation on the fly. Regarding the resource monitoring and configuration, a FIC can set a FIXP to notify the bandwidth usage, CPU usage and synchronize their operation.

FIFu identifies resources through Uniform Resource Identifiers (URIs) [90]. That is, a sequence of characters bound to a physical or virtual resource to identify and/or locate an entity. In this way, FIXPs look at their cache for a URI mapping to convert an equivalent output message upon receiving a valid packet. These representations are the basis for a recursive or iterative name resolution to forward packets. In the first, FICs search until finding an entity that has the required information to set an underlying FIXP, while, in the latter case, a FIC points to another FIC that may have the information. Focusing on these interactions between FIXPs and FICs and between FICs and FICs, FIFu establishes a tailor-made protocol that enables backward compatibility with legacy mechanisms and application while being flexible to encompass new primitives and architectures.

Taking advantage of this setup, the authors validate the proposal through 3 tests that exploit the interoperation between IP, NDN, PURSUIT architectures. In every test, three clients represent each stack, as well as three servers, wherein one server is a real IP web-server and the other two are NDN and PURSUIT video servers. Two routers interconnect these clients and servers at the edge of each network with a central FIXP that connects to the Intelligent Layer. Every entity in this network is a VM with Control Process Unit (CPU) cores of 3.33 GHz and 2GB of Random Access Memory (RAM), except the IP web-server that is a real existing server. Both NDN and PURSUIT chunk sizes are set with 4400 bytes. In the first scenario, NDN and PURSUIT clients fetch the IP server to test the web-browsing application. In the second, a PURSUIT video server provides a live video

stream for IP and NDN clients. In the last, the PURSUIT video server provides a fragmented video to maintain a video-on-demand application. The authors have evaluated performance metrics in terms of fetching and processing time, consumed bandwidth, and messages forwarded per second.

FIFu's future seems to rely on novel scenarios where this framework cooperates with real applications. In these future cases, FIFu would be deployed at edge networks, which it would redirect traffic from IXPs requiring conversion between architectures, or at 5G broadband cellular network slices, connecting different network architectures. Nonetheless, the authors acknowledge that the current design presents some limitations in terms of security, scalability, and lack of maturity.

3.3.7 Alloy

In the same line of multiple architectures interoperation, Jahanian et al. [91] develop a framework for seamless operation and content accessibility among content-oriented architectures . This project is COIN, which focuses on the current and future Internet architectures named NDN, MF, and IP. Their proposal is based on gateways that translate the communication protocols to consumers without modifying the original concepts.

COIN is a framework developed in JAVA that establishes three operational layers to achieve their goal. The first is the Routing Layer, an equivalent to the SDN's Data Plane. In this, the network devices forward and route the packets following their aimed recipient. The second layer is the Service Layer, an abstraction equivalent to the SDN's Control Plane. In this, mechanisms provide the required mapping for each data object at the upper layer. Finally, the third layer is the Information Layer, which is similar to the OSI's Application Layer and stores the data objects available in the network. In this way, the authors point that they ensure confidentiality, integrity, and provenance in their environment.

Besides, there are three essential components in their proposal. The first is the gateway, which translates and acts as a bridge between the different network domains. In some cases, the gateway can store content to ensure one of ICN pillars of in-network caching for the architectures. The second is the Name Resolution that ensures the naming structure of each architecture and provides an interface to achieve cross-domain applications. The third is the Object Resolution Service that retrieves the requested information, generating a coherent name to the recipient service domain.

Regarding the evaluation of their proof of concept, the authors employ the CCNx v0.8.0 for NDN, the latest version of MF, and a basic Linux implementation of IP forwarding. In this way, they exploit COIN's forwarding efficiency, latency, performance, and scalability. In one setup, they hand a personal testbed up with 5 VMs with 1 Gigabyte (GB) of memory and the

Ubuntu 14.04 to emulate a client, 2 routers, 1 COIN gateway, and 1 content provider. In another moment, they exploit the Open-Access Research Testbed for Next-Generation Wireless Networks (ORBIT) testbed [92] to emulate a grid topology of 400 nodes, each with 4 GB of memory and running Ubuntu 14.04. Nonetheless, the authors still point to the need for more evaluation with different services to investigate their proposal.

In another work, Jahanian et al. exploit an Alloy-based Information-Centric Interoperation framework [93]. In this model, they consider pull/-push -based communication models with static and dynamic contents, while analyzing failure and mobility. The authors do not address the COIN framework though, yet there are some synergies with both works.

3.4 SRQ2 and SRQ3: Takeaways and Opportunities

Several FIAs seek to redesign the Internet. Table 3.2 correlates each presented FIA and NG, summarizing their principle, communication model, and highlighting their work with SDN.

First of all, we can observe the FIA proposals heterogeneity. Even when we consider ICNs architectures like NDN, MF, and PURSUIT, they present disparities in their design principles and communication model. Considering the FIA and SDN convergence, the related works present a divergent approaches. Some proposals only seek to optimize the infrastructure of each architecture with SDN [83–86, 88], while others aim at the interoperability of FIAs [89, 91, 93]. Moreover, they have focused on different technologies, methodologies, and assumptions.

On one hand, [83], [84], and [88] focus on their Data Plane, lacking more data about their Control Plane. On the other hand, [85] presents a Control Plane that supports a catalog of content delivery services for its Data Plane. [86] combines Stratum O.S. and a Python script at its Control Plane to set its underlying Data Plane. [89] presents Future Internet Controllers with high-level functions to manage its Data Plane, provide a hierarchical distributed mapping between the available network resources, and decentralized knowledge over FICs. Once [93] and [91] apply a different framework based in JAVA, they do not require a Control Plane, exploiting a resolution service to foster the data exchange between their architectures.

In summary, the related work shows that the literature is lacking in a native FIA control agent for SDN. This is a major research opportunity, wherein a controller can support its architecture's features and share the same design principles. In other words, a native controller can present the same communication, security, and other relevant cornerstones in its design. Through exploiting NG in this work, we can broaden the FIA and SDN

convergence presenting a native SDN controller project that fosters NG's self-organization structure, contract-based operation, and sets the underlying Data Plane based on its naming structure.

Table 3.2: *Related Works.*

Aspect / FIA	**NDN**	**MF**	**PURSUIT**	**RINA**	**NG**
Cornerstone	Information-Centric Network.	Information-Centric Network.	Information-Centric Network.	Everything is IPC and recursive layers.	Convergent Architecture.
Design Principle	Data driven communication, stateful routing tables that fulfill an adaptive data delivery and network cache system.	Focus on mobility and wireless connections, where public identifiers called GUIDs are bound to domains' NAs. The GNRS foster the name resolution between these labels to fulfill data routing.	Design based on the publish/-subscribe model, wherein the communication is driven by publishers, subscribers, and network brokers. SID and RID for routing, where hierarchical DHT fosters the name resolution. Network cache possible.	IPC-based architecture, wherein recursive layers build the services and network. Flexible naming with globally unique names that can be either identifiers or addresses.	Integrates disjointed paradigms in its design, such as service-centric to build its applications, information-centric to data naming and exchange, life cycle to model its services behavior, and digital twin to represent entities.
Communication Model	Data-pulling system driven by the consumer.	Data-pulling system driven by the consumer.	Publish-subscribe model.	DIFs foster the communication, where the IDD DAF provides the end-to-end connectivity.	Publish/subscribe model, yet other models can be performed.
Approaches with SDN	• Even though [83] address the FIA as a generic ICN, it seems to present the interoperation between TCP/IP and NDN. In this, it conceives P4-based and custom SWes, a software proxy for the conversion between TCP/IP and NDN, and the control plane structure. • [84] creates a switch P4-based to fulfill the NDN forwarding structure. This proposal uses P4 and the BMv2 SW target to handle the data forwading, while developing a customized control plane for it. • Focusing on creating a SDN-based NDN data plane, [85] presents a architecture capable of managing network slices for NDN with P4-based forwarding elements.	• Focusing on the interoperation of ICN architectures, [91] and [93] creates a JAVA based framework to enable the ICN interoperation, exploiting the NDN, MF, and IP architectures.	• Even though it is not mentioned how they created their architecture in details, FIFu [89] focus on the interoperation between IP, NDN, and PURSUIT. This architectures deals with the Data and Control Planes, while also addressing some Management Plane aspects, e.g. overseeing the overall infrastructure performance.	• Proposal for creating interior and border SDN-based routers for RINA [86].	• Focusing on combining SDN, 5G, FIA, [94] presents an overview on the virtualization techniques for the 5G networks. In additions, it proposes a framework to apply NG in this context. • Exploring the OpenFlow technology, [95] proposes a framework of creating a SDN environment in which NG can establish contracts with OpenFlow resources. • Exploring the programmable NT20E2-PTP hardware, [96, 97] develops a framework to enable the NG data exchange through the KeyFlow architecture.

Approaches with SDN	• Even though it is not mentioned how they created their architecture in detail, FIFu [89] focus on the interoperation between IP, NDN, and PURSUIT. This architecture deals with the Data and Control Planes, while also addressing some Management Plane aspects, e.g. overseeing the overall infrastructure performance. • Focusing on the interoperation of ICN architectures, [91] and [93] creates a JAVA-based framework to enable the ICN interoperation, exploiting the NDN, MF, and IP architectures.				
General Computational Resources, Frameworks, and technologies to enable FIA on SDNs	Experimentation with the P4 16 language, BMv2 switch model, and some custom software to advance the NDN with SDN. In some works, it exploits *ndnSIM* network simulator, python and JAVA programming languages, Linux Ubuntu, and ORBIT testbed in [91, 93].	Develops a custom architecture to explore the ICN interoperation. In [91, 93], it exploits JAVA programming language, Linux Ubuntu 14.04, and the ORBIT testbed to evaluate their proposal.	FIFu [89] presents a custom architecture for ICN interoperation. Even thought the authors specify the network architecture versions, the presented work lacks on FIFu's tools, such as programming language, evaluation environments, and so on.	P4 architecture with the P416 language version, exploiting the BMv2 switch model, P4Runtime framework, and Stratum as the network OS [86]. Besides, some custom software to fulfill the proposal with Docker containers and Mininet network simulator.	So far, NG has explored the SDN technologies called OpenFlow and KeyFlow. In these works, the resources encompass the O.S. Ubuntu 16.04, the NT20EP-PTP programmable board with the Verilog modeling language, and some custom software. For the 5G scenario, it aims at exploring the GEO/MEO Satellites and the technologies OpenSAND, OpenDaylight, and Open vSwitch.

Chapter 4

NovaGenesis Control Agent with Future Internet eXchange Point

THIS chapter explores the design choices taken to develop a functional NovaGenesis Control Agent (NGCA). Under this scope, this chapter covers two major themes concerning an NG enabled Data Plane with P4 and the NG-based SDN controller for Future Internet eXchange Point (FIXP). It is crucial to highlight that this FIA Control Plane is the most relevant contribution once it seems to be the first native controller for a FIA.

4.1 A New NG Adaptation Header

As presented before, NG messages are encapsulated on legacy Link Layer technologies. Moreover, NG deals with software and hardware routing by its Forwarding/Routing Line. Formerly, NG protocol has not presented any reserved bytes. This yields that any data related to the network topology could not be retrieved. As a consequence, how can we develop a mechanism to retrieve the crucial IngPort and Future Internet Exchange Point Switch (FIXPSW) ID to set the underlying DP in the FIXP scenario? One of the first ideas to address this was to deploy a NG instance on each FIXPSW Virtual Machine VM to generate a NG message that would inform the CP and it should work alongside with P4. By all means, this idea was discarded once it is unfeasible, considering that every network element would require a NG instance network element. Therefore, how could we converge FIXP with minor adjustments to the NG protocol?

These questions have granted some insights on NG inter- and intra-domain routing applications, lessening major NG structural adjustments. Firstly, the NG communicating protocol has been modified, adding four reserved bytes to the NG Adaptation Header to depict and enable the easy assess of the FIXP network infrastructure, in terms of FIXPSW IDs, EPs, and IngPorts. These bytes foster a self-managed and self-aware environment,

yet they modify the structure for every NG use-case up to date. Nevertheless, reserved bytes are a common practice in several network protocols. For instance, Internet Protocol version 4 (IPv4) [98] and Internet Protocol version 6 (IPv6) [99] present optional fields for special cases of operation. For most NG cases, these new bytes are filled with the “0x44” value and the NG Adaptation Layer ignores them. In this way, the previous concepts remain intact, regardless of these four extra bytes.

Nonetheless, these four bytes takes a new role for the specific FIXP scenario. For now, the first reserved byte remains unused. On the other hand, FIXP.P4 fills the second byte with its Switch Identifier (SWID), a value that is the unique ID for a FIXPSW, and the third byte with the packet’s Ingress Port. At last, the fourth byte notifies the Last Fragment of a NG message, once FIXP.P4 must be aware of where a fragmented message ends. Figure 4.1 illustrates this reserved bytes sequences.

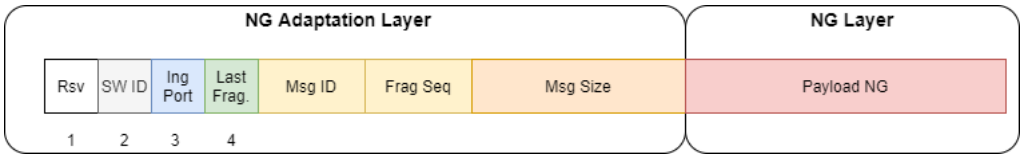

Figure 4.1: *NG Communicating Protocol and the Reserved Bytes.*

Therefore, either FIXP.P4 or the proper sender PGCS fills these reserved bytes. The next list summarizes the reserved bytes, while Figure 4.2 exemplifies a NG message that has 3 fragments with the new reserved bytes.

- **First reserved byte:** Unused byte.
- **SWID byte**: FIXP.P4 fills this field with its own SWID data when deparsing an incoming NG packet.
- **Ingress Port byte**: FIXP.P4 fills this field with the Ingress Port from the incoming packet when deparsing a NG packet.
- **Last Fragment byte**: The emitter PGCS fills this field with two values. If it is last fragment, it receives the value “0x46.” In cases of on-going fragments, it has the value “0x45.”

Even though these might seem limited at this moment, NG can employ these four new bytes in other future scenarios that we do not foresee yet. In the FIXP scenario, this method enables a NGCA that is aware of the underlying network topology, that can self-manage its data based on the incoming packets, and can be scalable to meet any type of topology layout without a previously configured routing table in its knowledge.

4.2 NG enabled Data Plane with P4

This section focuses on exploring the decisions taken on the P4 architecture to develop a functional FIXP that can manage NG data traffic. In this

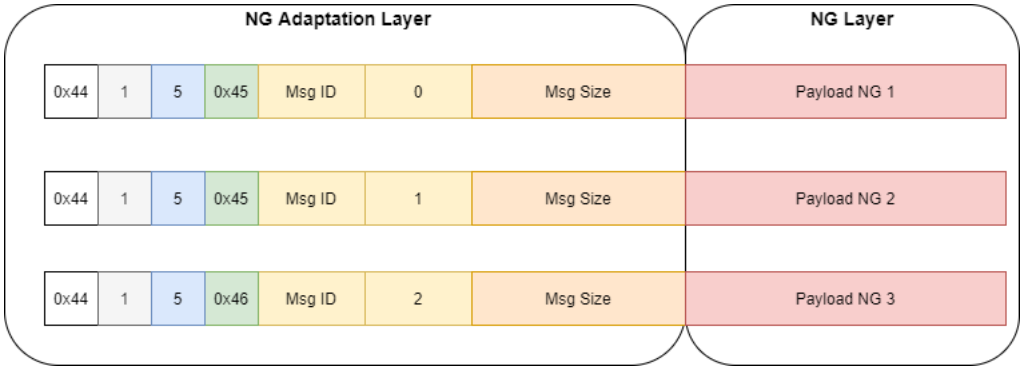

Figure 4.2: *Example of an ongoing NG message that has 3 fragments that has been sent by an PGCS and deparsed by FIXP.P4.*

way, it covers the P4 headers modeled for NG communication protocol, the fragmentation case, and the NG broadcast scenario.

4.2.1 NG Headers

As P4 proposes to handle any custom protocol, the first step is to enable NG communication standard. In this way, P4 can deal with any incoming NG byte stream and translate this sequence into meaningful NG values.

In the first place, it is important to once more highlight the existence of the fragmentation of NG messages. This fragmentation can impact on the NG Routing Line once the packets are divided and, so, fragments might not present the Routing Line and the Destination Host Identifier (DHID) after the first fragment. Figure 4.3 shows two distinct scenarios for a NG “Hello” of 273 bytes, wherein the case **A** shows a packet with the full NG Message, while case **B** depicts the fragmentation of this original message.

Considering NG scenarios that use the Ethernet protocol with a Maximum Transmission Unit (MTU) parameter bigger than 140 bytes, the first fragment presents the NG Routing Line, while the following fragments do not. As a consequence, P4 must consider two distinct NG headers for its modeling. For the full and complete NG messages or fragments with the Routing Line, P4 must model the NG bytes up to the NG Routing Line. In this way, P4 assesses the Reserved Bytes, Message Identifier (MsgID), Sequence Identifier (SeqID), and the Destination Host Identifier (DHID). At this moment, FIXP considers an intra-domain communication.

Listing 4.1 illustrates the P4 NG header for the first fragment of packets with MTU bigger than 140 bytes. Notice that this header models the encapsulated bytes in the Ethernet payload. From this starting point, the first 32 bits from the Ethernet packet payload are the four NG reserved bytes. The next 16 bytes represent the NG Adaptation Layer, i.e. 32 bits for MsgID, 32 bits for the Fragment Sequence, and 64 bits for Message Size. After this, the NG message begins. For full NG messages or first fragments, the first

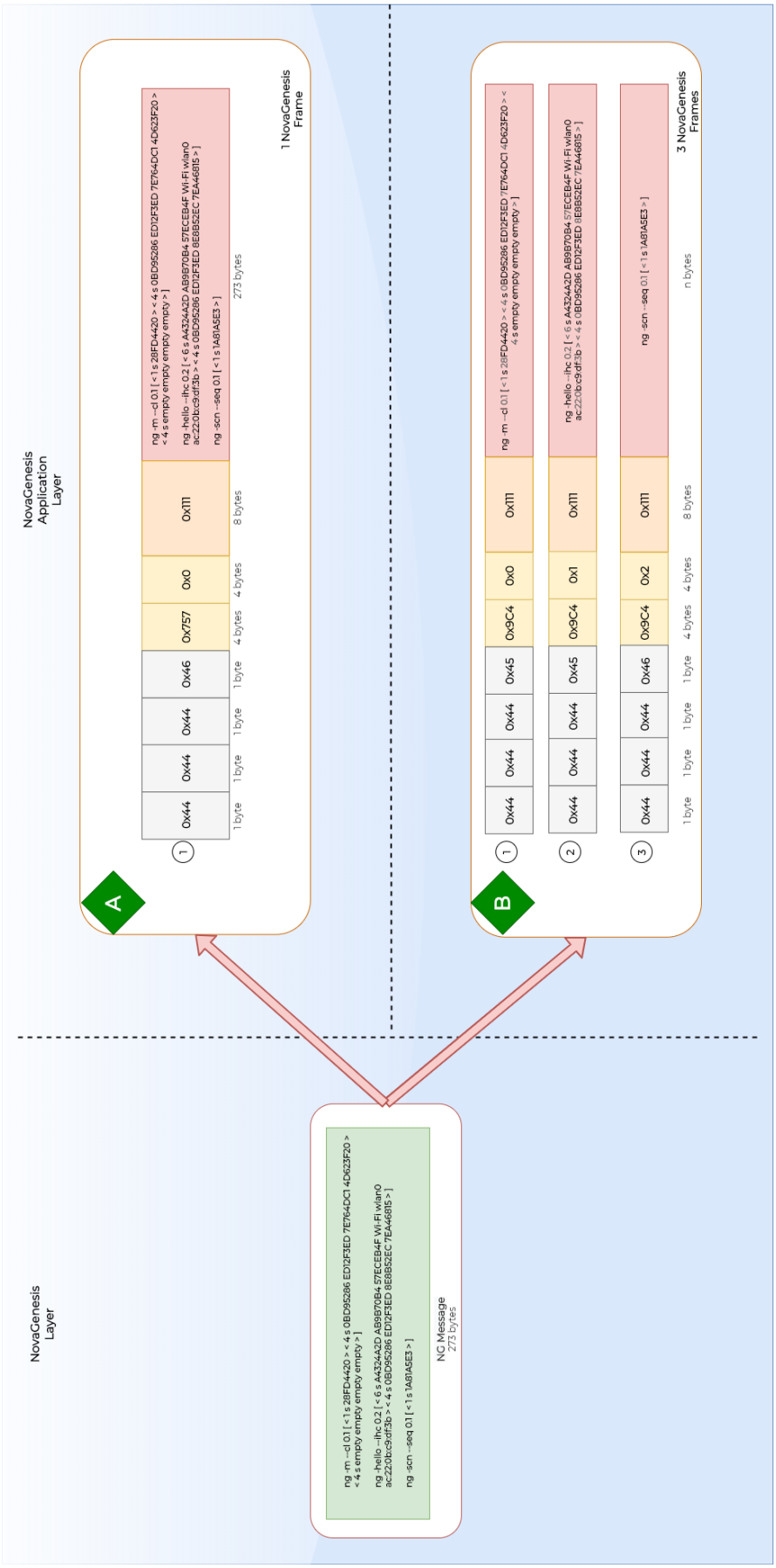

Figure 4.3: *NG possible transmission scenarios.*

Listing 4.1: *NG headers on P4 modeling.*

```
header novagenesis_hasntDHID_t
{
        bit<32>         rsvd;
        bit<32>         msgId;
        bit<32>         fraqSeq;
        bit<64>         msgSize;
        bit<672>        iniMsg;
        bit<64>         dhid;
}
```

NG Command Line specifies the NG Routing Line, which presents the Source and Destination Identifiers. Considering this, P4 must skip 672 bits to reach the exact position where the DHID is expected. After these 672 bits, the next 64 bits defines the DHID.

Nonetheless, NG presents scenarios where a full NG message must be fragmented once MTU limits the Ethernet payload length and the following fragments after the first (Figure 4.3's case **B**) do not present the NG Routing Line. In this way, the previous header in Listing 4.1 is valid for the first fragment and the first one only, once it presents the DHID on its expected position. On the other hand, the following fragments that are sent only match the fields up to the NG Adaptation Layer. In this case, Listing 4.2 models the following packets after the first fragment. In the Ethernet payload, the first 32 bits represents the 4 NG reserved bytes and the next 128 bits represent the NG Adaptation header. After these 160 bits, the proper NG message continues.

Listing 4.2: *NG headers on P4 modeling.*

```
header novagenesis_hasntDHID_t
{
        bit<32>         rsvd;
        bit<32>         msgId;
        bit<32>         fraqSeq;
        bit<64>         msgSize;
}
```

4.2.2 NG Packets and Fragmentation

After solving how to model two distinct NG headers, another challenge relates to how P4 could forward NG fragmented packets, once only the first fragment presents the NG Routing Line. The first approach to decide if a packet has a NG Routing Line is to check the Fragment Sequence control bytes. Notice that regardless of the case in Figure 4.3, the first fragment or a full message presents this value as "0." Upon assessing this field, P4 can

select which NG header must be applied to parse an incoming NG packet. Listing 4.3 illustrates the P4 algorithm to determine this.

Listing 4.3: *P4 selecting the proper NG header.*

```
bit<96> hasDHID = packet.lookahead<bit<96>>();

transition select (hasDHID & 0xffffffff)
{
        0: novagenesis_hasDHID_parse;
        default: novagenesis_hasntDHID_parse;
}
```

This algorithms selects the proper NG header to parse an incoming NG packet. As P4 does not incorporate complex mechanisms in its framework, notice that the structure *packet.lookahead<bit<96»()* shifts 96 bits from the Ethernet Payload start and stores these 96 bits on *hasDHID* variable. On the next line, transition acts as an decision structure to select the adequate header upon validating the Fragment Sequence. If this field is equal to 0, the bitwise AND operation *hasDHID & 0xffffffff* results in a 0 value, and the header *novagenesis_hasDHID_parse* is selected. For any other combination of Fragmentation Sequence, the header *novagenesis_hasntDHID_parse* is chosen.

This solves the first issue of dealing with distinct NG headers. Through this transition, P4 can parse the packets with the proper NG header. However, one can point that only the first fragment presents the Routing Line, so how can we forward ongoing fragments to their destination?

4.2.3 P4 enabled Future Internet (FI) fragmentation

PGCS can fragment NG messages into several NG frames and, so, send fragmented packets to its peers. Nonetheless, the NG Routing Line appears only in the first fragment for packets created on networks of over 140 bytes. Thus, any kind of Forwarding HW must either encompass the NG remounting scheme to retrieve this or create a novel way to assess this data.

Taking advantage of the NG headers structure, i.e. *novagenesis_hasDHID* and *novagenesis_hasntDHID*, P4 is aware of which fragment presents the NG Routing Line. Therefore, P4 is already ready to reach the DHID data. Besides, this language presents the **Register** data structure, which is a stateful memory that can be managed through actions [100]. Through this structure, P4 can store data for a unspecified time. Even though DHID and Registers can be applied to this solution, the mechanism still lacks on the aspect of retrieving which message the fragment belongs.

As a consequence, we must consider the NG Message ID as the index to retrieve the DHID for every fragment. In this way, a hash function is

applied on P4 to bind the DHID received on the first fragment, through *novagenesis_hasDHID*, and make it available for incoming fragments, on *novagenesis_hasntDHID*. Currently, combining Registers and hash function is the only way available for storing a data and retrieving it afterwards. Listing 4.4 defines the hash function to store the DHID and Message ID.

Listing 4.4: *P4 releasing storage.*

```
hash(hash_map_index, HashAlgorithm.crc32, (bit<10>)(0),
hdr.novagenesis_metadata.msgId}, (bit<10>)(1023));

hash_map.write((bit<32>)hash_map_index,
                    hdr.novagenesis_metadata.dhid);
```

Nonetheless, one can cite that we solved the aspect of making available this data and that P4 forwards every fragment based on the Message ID index. Although, for how long should we make this data available? Even though we are in a controlled environment, P4 devices are limited HW. Taking advantage of the reserved bytes, the fourth points the last fragment of a given message. Whenever a last fragment is sent, the sender's PGCS writes the hexadecimal value of 0x46 in this reserved byte. When P4 detects this value, it can clear the register through the hash function, releasing some storage space. Listing 4.5 illustrates the algorithm for this.

Listing 4.5: *P4 releasing storage.*

```
if(hdr.novagenesis_metadata.dhid != 0 &&
(hdr.novagenesis_hasntDHID.rsvd & 0xff) == 0x46)
{
        hash_map.write((bit<32>)hash_map_index, 0);
}
```

4.2.4 NG Broadcast Rule

As explained before, NG sends "Hello" messsage periodically for building a NG domain dynamically. During this startup process, every NG message exposes the origin's IDs to a communication peer through a series of NG messages. As a consequence, the NG Routing Line presents the source's IDs and the NG broadcast address "FFFFFFFF" as the destination ID. After discovering the resources available, every communication peer knows the related IDs of its partners, generating NG Routing Line for a specific destination host. In other words, we can summarize this by saying that NG presents two types of Routing Lines, one that presents the broadcast address for "Hello" and another with explicit the destination IDs.

As a consequence, NG must either set the P4 on the fly with this NG broadcast address or create a mechanism that establishes this entry at the FIXP's Routing Table during its startup. Simplifying the process for now, it was chosen to modify the FIXP startup to perform this task.

In this way, FRHS performs a NG Broadcast rule addition into NG RT as a startup procedure. This mechanism creates a P4 multicast group that maps every NG host, its FIXP network interface, and the CP interface to the broadcast "FFFFFFFF" address at FIXP startup. This method is based on the "ng_broadcast" P4 action that forwards any incoming NG "Hello" broadcast packet to the multiple NG EP when this DHID key matches the P4 multicast rule. In order to do so, the P4 program must have the required metadata fields to enable BMv2 multicast, which is the *mcast_grp*, a non-zero value that assigns a group of peers to a valid group ID.

intrinsic_metadata.mcast_grp : An ID to associate multiple EP to replicate a packet from the BMv2 ingress pipeline.

Nonetheless, BMv2 replication engine multicasts the packet to every Egress Ports pointed by this rule. Therefore, a packet back-flow happens when we consider this, forwarding a packet to its origin. For scenarios with more than one FIXP this yields in a sharp unwanted increase of traffic, where two neighbor FIXP will perennially exchange data due to this multicast rule. In order to avoid this, a filter in the Egress pipeline avoids to replicate multicast packets at the same Ingress Port. Currently, this can be done trough the following Algorithm depicted on Listing 4.6.

Listing 4.6: *P4 Multicast Filter.*

```
if(standard_metadata.egress_port == standard_metadata.
                                                ingress_port)
{
        mark_to_drop(standard_metadata);
}
```

In other words, this code segment avoids to replicate a multicast packet in the Egress Port that matches the packet's original Ingress Port. Another way to solve this issue is to consider a single multicast rule for each FIXP network interface, which has not been evaluated due to time constraints.

4.2.5 P4 Aware Control Agent

So far, the NG development addressed the different headers and the distinct purposes of "hello" and exchanging data messages. Through these, the DP forwards any incoming NG packet to its destination, multicasts a NG broadcast message, or requests a configuration from the CP.

Nonetheless, how can the controller become aware and take a decision to set the underlying DP, creating a dynamical knowledge to adjust the underlying network topology without human interference? Is it practical to set this data for each topology? Is it even sane to consider a future FIXP application in a real environment that one has to map every device, exchange point, and network topology?

In order to create such a NG Control Agent (NGCA) based on P4, *FIXP.P4* exploits the NG reserved bytes to write the SWID and the respective IngPort from a NG packet. After this, the packet is sent to the CP, carrying the meaningful data that depicts the network topology at the NGCA. Listing 4.7 illustrates the algorithm to write these data based on the P4 language.

Listing 4.7: *P4 writing the SWID and the packet's Ingress Port.*

```
if(hdr.novagenesis_hasDHID.isValid())
{
    hdr.novagenesis_hasDHID.rsvd = SWID*65536 +
    ((bit<32>)standard_metadata.ingress_port&0xff)*256 +
    (hdr.novagenesis_hasDHID.rsvd&0xff)
}
else if (hdr.novagenesis_hasntDHID.isValid())
{
    hdr.novagenesis_hasntDHID.rsvd = SWID*65536 +
    ((bit<32>)standard_metadata.ingress_port&0xff)*256 +
    (hdr.novagenesis_hasntDHID.rsvd&0xff);
}
```

Regardless of the NG header, P4 writes the same sequence of bytes of distinct data at the NG Adaptation Layer. The second NG Reserved Byte assumes the role of point the SWID, the third indicates the IngPort, and the fourth the Last Fragment. Notice that the bitwise logical AND operation acts as a forced cast to reduce bigger variables, as SWID, to an 8-bit value. Figure 4.4 illustrates graphically how P4 process the Reserved Bytes.

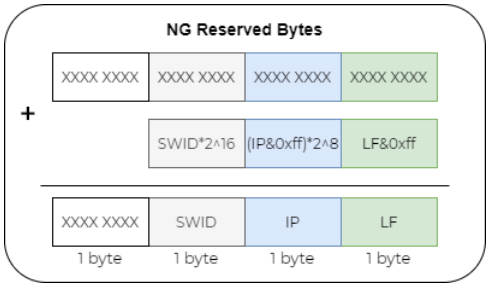

Figure 4.4: *NG possible transmission scenarios.*

After this, P4 forwards every packet to the NG CP as illustrated on Figure 4.5. Upon receiving this packet, the NGCA can now create a dynamical

knowledge based on these reserved bytes, managing each request based on its own data and through the SWID and incoming IngPort.

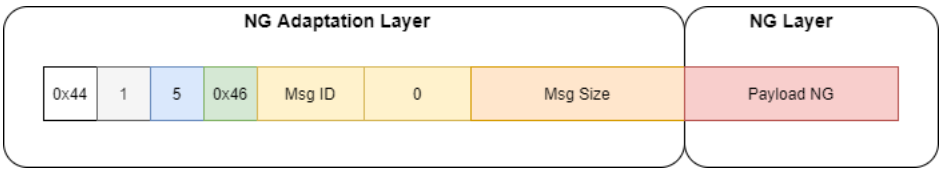

Figure 4.5: *Example of a NG packet forward to the CP by P4.*

4.2.6 NG P4-based Data Plane

Figure 4.6 illustrates the FIXP.P4 model for NG using the P4 pipeline. Meanwhile, Figure 4.7 presents the *FIXP.P4* event diagram, encompassing the actions from receiving a NG packet until the its Egress State. Notice that this figure has two different packet flows, where the left side considers the first NG packet fragment of a message, i.e. presenting the *DHID* in its fields. In this, FIXP.P4 applies the broadcast and *ng_SetSpec* rules for known *DHID* destinations. On the other hand, it forwards the unknown packets for the NG Control Agent rules. On the right hand side of the sequence diagram, FIXP.P4 considers the other NG fragments that does not determine the DHID field. Therefore, it has to retrieve the *MsgID* from the P4 hash function and decide if its a known or unknown packet. After this point, the sequence follows the same procedure of the left hand side.

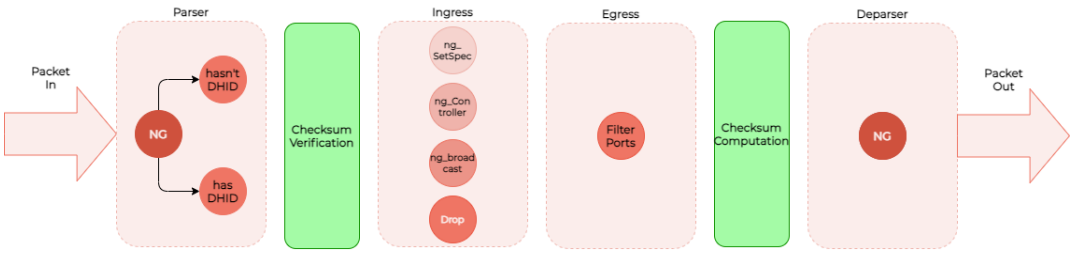

Figure 4.6: *FIXP diagram for NG Data Plane exploiting the P4 pipeline.*

4.3 NovaGenesis Control Agent

NG Control Agent (NGCA) has no prior knowledge of the physical network topology. Even though FRHS knows the NG peers' EP for the broadcast rule, this is the maximum knowledge that the environment has about NG peers. In this way, NGCA acquires all the data belonging to the NG peer hosts, FIXPSWes' ID and network interfaces on the fly. This sub-section presents the design choices to develop suc NGCA, covering the expansion of the PGCS, modeling the FIXP mechanisms into a new PGCS object, and building its knowledge to set the underlying network topology.

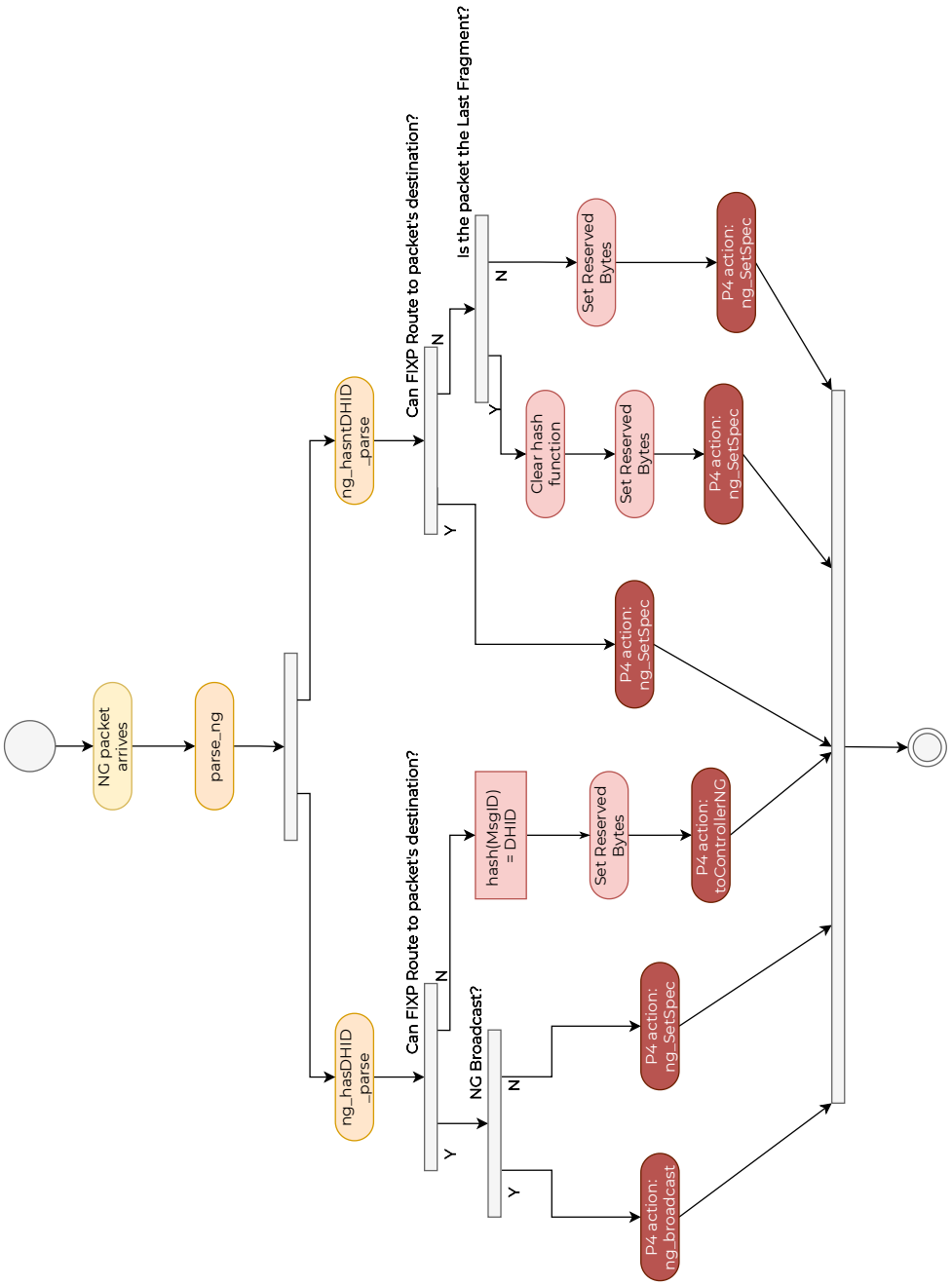

Figure 4.7: *FIXP model for NG Data Plane.*

4.3.1 PGCS Control Agent

To fulfill our goal, NGCA extends the current PGCS scope. Firstly, PGCS has received a new initialization for FIXP scenarios to create the Control Agent. Listing 4.8 exemplifies this startup process that establishes the communication port *0* for an NG intra-domain communication with Ethernet protocol. For this case, "*-fixp*" option sets the PGCS to become the FIXP controller, using its host "*Eth0*" network interface to communicate. Finally, this agent exploits Ethernet packets with up to *1400* bytes of MTU.

Listing 4.8: *NovaGenesis Control Agent Initialization Example.*

```
1 ./PGCS ./ 0 Intra-Domain Ethernet -fixp Eth0 1400
```

Upon initializing PGCS with such parameters, an instance of the NGCA is created referring to the specific PGCS FIXP Controller C++ class. This object has all the atributes and methods required to oversee FIXP operation and its Data Plane. For instance, it presents raw sockets for NG and FIXP communication protocols, a database to create a dynamical knowledge, called FIXP Knowledge, about the underlying network topology, and the required modeling to create the FIXP primitives.

4.3.2 FIXP Knowledge

At most of FIXP application, PGCS builds an internal database based on the received NG packets called "fixpKnowledge." This knowledge database exploits a C++ two-dimensional vector that conveys a N x 6 matrix [1]. The variable **N** determines the number of rows and 6 the fixed number of columns in this structure. Each row has a specific entry mapped for an unique set defined by HID, IngPort and SWID retrieved from every NG packet that reaches the NGCA, reflecting the underlying network topology. Moreover, assumes that the network infrastructure can encompass more than one underlying FIXPSW. In other words, this database can present several entries of a given HID, yet SWID row changes per SWID. Table 4.1 illustrates the structure and the following list explains each aspect of FIXP knowledge.

- **HID** - The index 0 column registers the Source Host Identifier (SHID) for each NG packet as a string type value.
- **MAC** - The index 1 column stores the source and destination Ethernet MAC address 12 bytes field as a string.
- **Switch Identifier** - The index 2 column collects the SWID data from the NG second reserved byte, storing it as a string type with the value of ("s" + SWID reserved byte).

[1] Notice that the most common vector is the uni-dimensional case, where we declare a variable with *n* lines. In C++, this could be done by specifying *"char variable1[n].´´* For the two-dimensional case, the variable represents a *n x m* matrix with n lines and m columns. In C++, this is done by determining *char variable2[n][m].*

Table 4.1: *FIXP Knowledge, a two-dimensional Vector Structure.*

fixpKnowledge	**HID**	**MAC**	**Switch Identifier**	**Ingress Port**	**Sequence ID**	**Valid Table Add**
fixpKnowledge	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	[0][5]
fixpKnowledge	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	[1][5]
fixpKnowledge	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]	[2][5]
...	...	...	...	...	...	...

- **Ingress Port** - The index 3 column picks the IngPort data up from the NG third reserved byte. This cell stores it as an string type variable.
- **SeqID** - The index 4 column represents the SeqID data that is generated upon sending a Table Add-Modify primitive. Therefore, it receives a null value at a new entry and receives a string value upon sending that primitive. This sequence ID is a value that retrieves the same row index number plus one. In other words, the SeqID is 1 for the entry at the line 0. Meanwhile, this field is important to track and handle the FIXP Primitive management.
- **Valid Table Add** - The index 5 column sustains the successful entry at FIXP RT data from the FIXP Ack packet. Therefore, it receives a string value “NOT” at a new entry and PGCS only changes this value to a string of “OK” when NG retrieves an Ack packet that points to a valid entry at FIXP SW. This also enables the reinsertion packet process.

As the network topology can present several FIXPSWes, NGCA associates the HID and SWID fields to map the network topology and generate new entries to its knowledge. This yields that fixpKnowledge can display more than one entry for the same HID, yet with distinct SWIDs.

Through the association of HID, SWID, and IngPort columns, NGCA becomes aware of the underlying network topology, in terms of how to reach a NG host based on the FIXP SW and its network interface. Figure 4.8 represents the flowchart for NGCA managing its knowledge as new packets arrive. It is necessary to highlight that this flowchart depicts only the processes related to analyzing an incoming packet, adding new entries, and modifying an existing entry on the FIXP Knowledge database.

As can be seen in Figure 4.8, the NGCA analyzes every packet that reaches the CP. This element first parsers each NG packet to retrieve the MAC address, the IngPort, the SWID, and the SHID fields. After this, NGCA looks up at the FIXP Knowledge table to verify if this SHID has already an entry related to the considered HID and SWID by looking the first and third columns up. In case of a new peer or a different FIXPSW, NGCA records this related data in a new FIXP Knowledge line, which looks similar to the n-th entry depicted on Table 4.2.

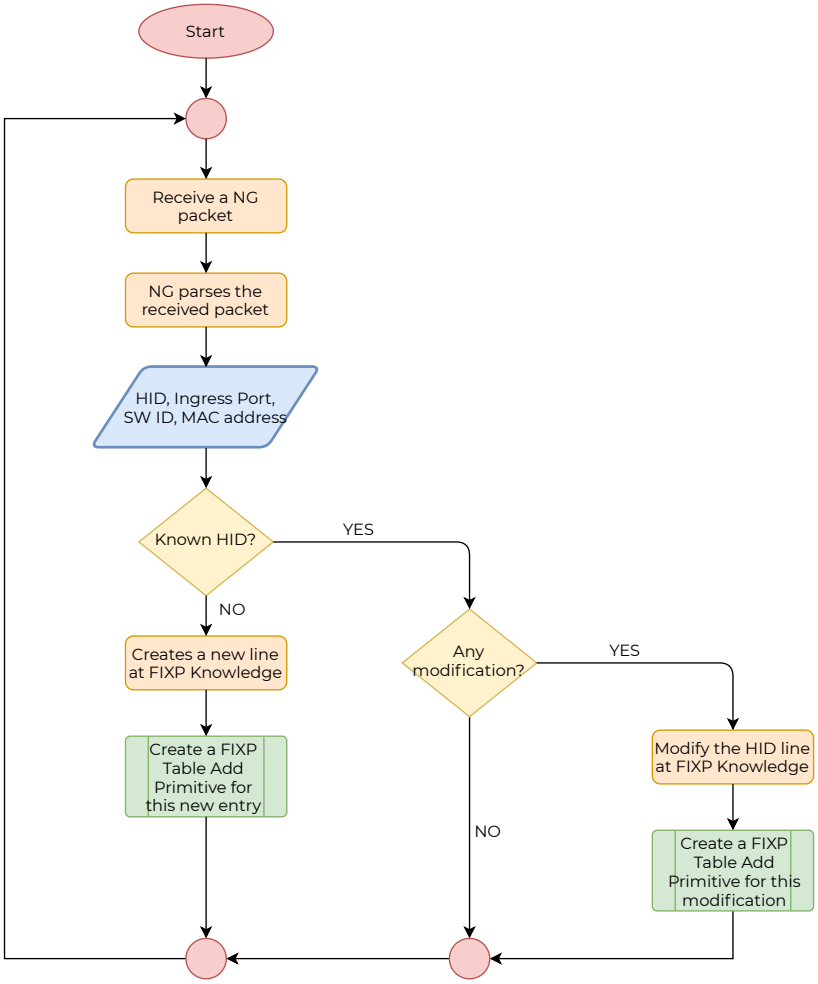

Figure 4.8: *NGCA Flowchart for building its FIXP Knowledge.*

Table 4.2: *N-th entry at FIXP Knowledge.*

Line Index	**HID**	**MAC**	**Switch Identifier**	**Ingress Port**	**Sequence ID**	**Valid Table Add**
n - 1	...	...	...	...	**n - 1**	...
n	**HID**	**MAC addresses**	**SWID**	**IngPort**	**n**	**"NOT"**
n + 1	...	...	...	...	**n + 1**	...

4.3.3 FIXP Primitive Conceiving and Processing

NGCA acquires its knowledge on the fly. At some point, it will receive a valid packet with the destination ID that matches its database or, very interestingly, PGCS can subscribe on NRNCS for clues that can help on configuring FIXP tables. Figure 4.9 summarizes the NGCA process to analyze a received packet with its database.

At first, PGCS awaits for new packets and parses it, retrieving its DHID. Then, it verifies if knows how to reach this destination or not based on the FIXP Knowledge. For a unknown destination, NGCA decides if this packet

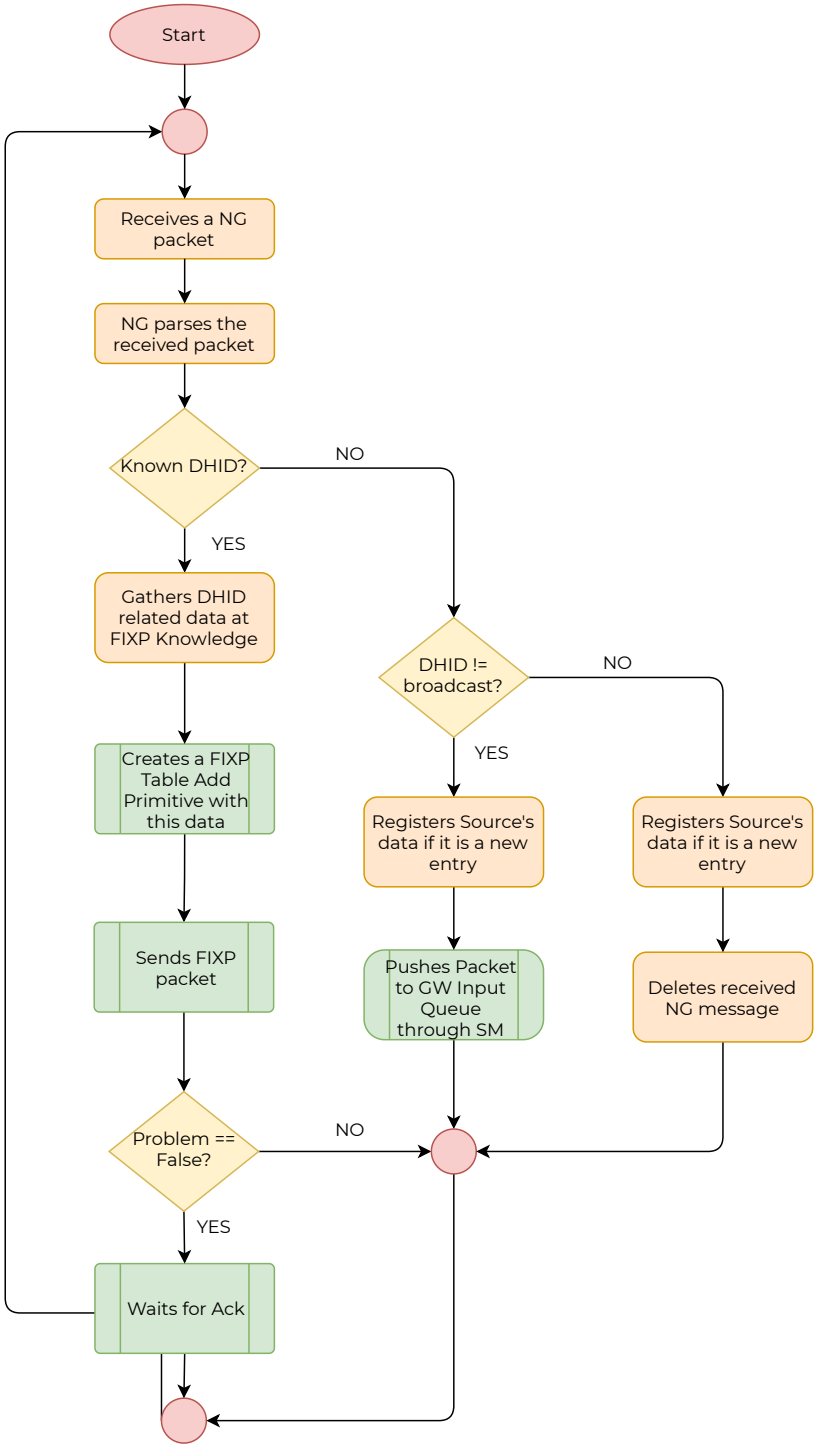

Figure 4.9: *FIXP Table Add-Modify primitive conceiving flowchart.*

relates to a NG hello broadcast or not. If it is not a hello, it registers the packet's source data at FIXP Knowledge if it is related to a new entry and puts this packet on the GW Input Queue through the SHM to be further reinserted into the network. This process is common for NG message processing that pushes the packet into a queue, so PGCS can deal with these packets at a more convenient moment. In other words, a packet enters the

PGCS processing packets until NGCA learns how to reach its destination and reinsert it into the network. Otherwise, NGCA registers the packet's source data at FIXP Knowledge if it is related to a new entry and discards it. Meanwhile, it is important to highlight that this happens because FIXPSWes have already forwarded the hellos based on the DP based on NG broadcast rule, i.e. DHID = "FFFFFFFF." In this way, NGCA optimizes GW queues by entering only packets with host specific DHIDs to be further delivered.

Alternatively, NGCA will learn how to reach a destination in a good time. In this case, the packet's DHID and FIXPSW fields matches the FIXP Knowledge's HID and SWID columns in an **n** line index. When this happens, it gathers the IngPort and its built-in knowledge of P4 structures to build the FIXP Table Add primitive. This data has is converted into a JSON pattern and its construction is exemplified on Listing 4.9.

Listing 4.9: *Table Add-Modify Primitive on JSON format.*

```
{
  "Ethertype": 0x1234,
  "Seq ID": FIXPKnowledge[n][4],
  "SWID": "FIXPKnowledge[n][2]",
  "Thrift": 9090,
  "Command": 1,
  "Forwarding": ["FIXP_Switch_Ingress.novagenesis_forward"],
  "Key List": ["FIXPKnowledge[n][0]"],
  "Action List": ["FIXP_Switch_Ingress.novagenesis_SetSpec"],
  "Action Param": ["FIXPKnowledge[n][3]"],
  "Primitive Option": 0,
}
```

PGCS employs RapidJSON API [101] to create the JSON structure. This conveys the FIXP standard to generate the Table Add-Modify primitive, in which the fields either conveys NG, HID, or P4 related data to configure the underlying FIXPSW. As a rule, JSON delineates that string type variable must be represented between quotation marks. As an example, the key "SWID" has the FIXP Knowledge value from the third column, e.g. "s01." The other fields that does not present these marking signals are integers, such as Ethertype, Thrift, or Primitive Option values.

Upon creating this JSON data structure, PGCS is ready to craft the FIXP packet to configure the FIXPSW. Algorithm 1 explains this process. Firstly, NGCA creates the FIXP Table Add primitive, encapsulating the JSON into a Ethernet payload with FIXP Ethertype, i.e. 0x0900. After this, it sends this setting through the FIXP Client Raw Socket Identifier (FCSID) raw socket. If the packet is sent successfully within two minutes, NGCA moves to other processes. If not, it tries to resend.

Algorithm 1 FIXP Primitive Packet Generator

```
Creates the FIXP Table Add packet
Selects the FCSID raw socket to send the FIXP Table Add
while Try to send the packet for 2 minutes do
    Sends the packet to FIXP Data Plane
    if Packet sent successfully then
        Problem = false
        Break
    else
        Problem = true
Return Problem variable;
```

4.3.4 FIXP Acknowledgement Primitive

After sending any setting FIXP primitive, NGCA expects a FIXP Ack packet.Figure 4.10 illustrates the flowchart of receiving a FIXP Ack, while Listing 4.10 exemplifies the JSON data structure for a FIXP primitive Ack. This algorithm validates the Ack Status field. For a valid Ack (“0”), NGCA updates its FIXP Knowledge sixth column with the Ack’s “Seq ID,” writing the “OK” string value that points that this entry has been successfully added into the desired FIXPSW. After this, NGCA is ready to initiate the FIXP reinsertion process. In case of a received SeqID or SWID value that does not point to a valid entry, PGCS disregard the Ack packet.

Listing 4.10: *FIXP Ack Packet on JSON format.*

```
{
  "Ethertype": 0x1234,
  "Seq ID": 1,
  "SWID": "s01",
  "Thrift": 9090,
  "Command": 1,
  "Handles": ["..."],
  "Status": 0 or 1,
}
```

Notice that a FIXP Ack packet presents control information that, at this moment, are not exploited by NGCA. These are necessary for the underlying FIXP structure to forward the packet. Nonetheless, the Handles field could be employed to further delete a specific entry on FIXP’s NG RT without the key that generated it. On the other hand, NGCA takes into account the SeqID, SWID, and Status fields to verify on FIXP Knowledge.

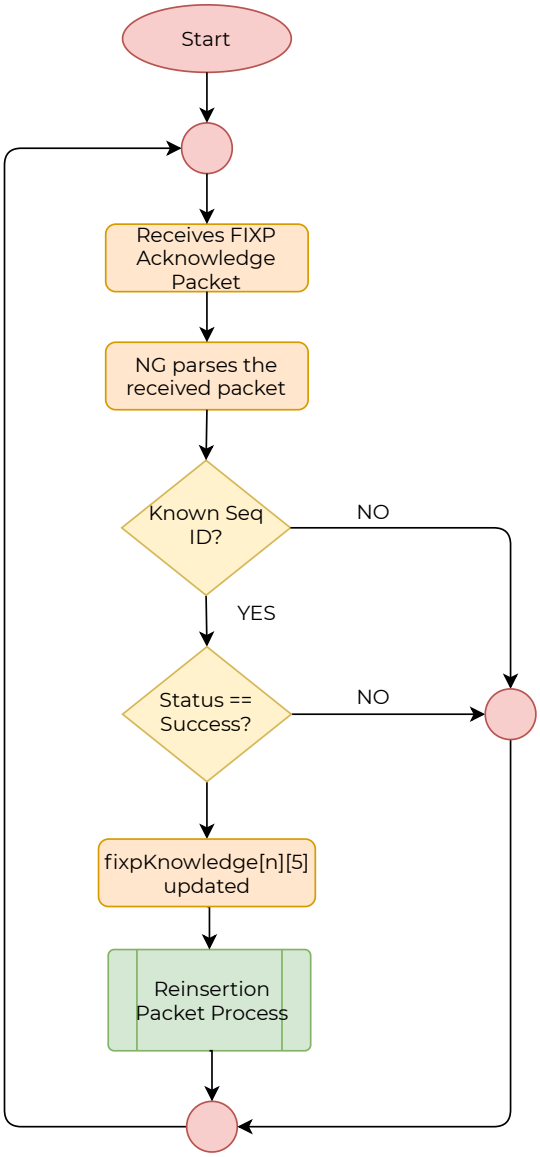

Figure 4.10: *NG FIXP Ack wait flowchart.*

4.3.5 Reinserting Original Packet

After validating the FIXP Ack packet, NGCA reinserts the NG messages with set DHID that are in its GW Input Queue into the network. Figure 4.11 illustrates the Reinsertion Process flowchart that initiates right after receiving a successful FIXP Ack. This process takes the full NG message, adds the NG Adaptation Layer control headers, fragments it into smaller packets if needed, and generates the FIXP Reinsertion JSON. Listing 4.11 exemplifies this Reinsertion JSON model. It is important to highlight that JSON encapsulates the full NG packet as a string value of the “Packet” key, i.e. the sequence of bytes encoded as a string type variable.

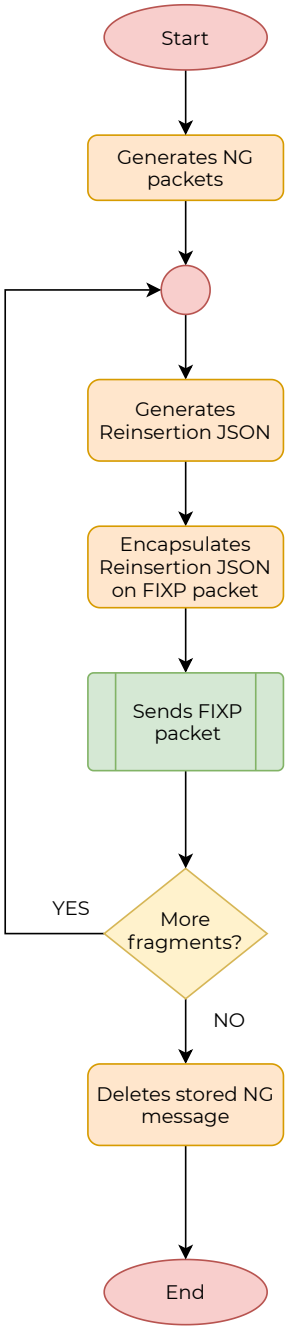

Figure 4.11: *NG Reinsertion flowchart.*

Listing 4.11: *FIXP Reinsertion Packet on JSON format.*

```
{
  "Ethertype": 0x1234,
  "SWID": "s01",
  "Command": 5,
  "Packet": String content with full NG packet,
}
```

Following the FIXP primitive conceiving, NGCA creates the FIXP Reinsertion packet to be sent through FCSID. This packet takes up the same structure as a regular FIXP primitive, i.e. it is an Ethernet packet with the MAC address from the NG controller and the desired FIXPSW, FIXP Ethertype (0x0900), and the JSON as its payload. As PGCS concludes sending all the possible fragments that compose the original NG message, this related information is discarded, i.e. deleted from its memory. Figure 4.12 demonstrates how PGCS encodes a NG packet into a JSON format, which, in turn, is later the FIXP Primitive Packet payload.

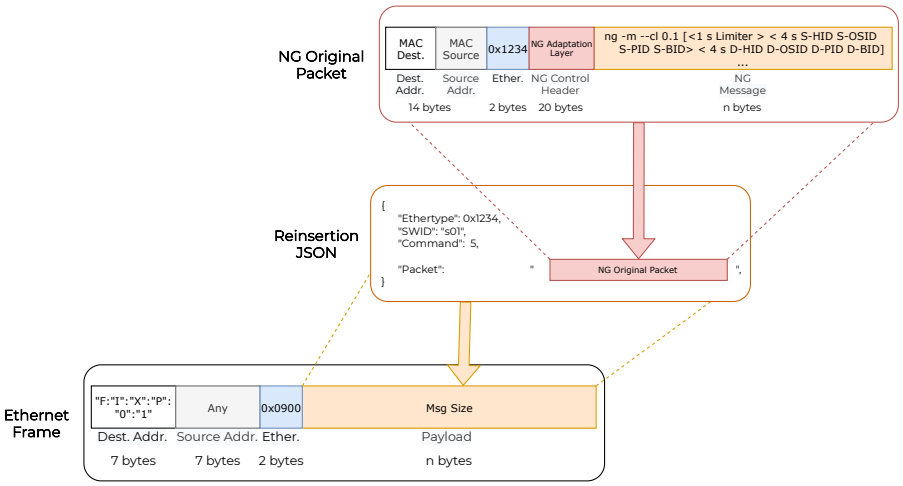

Figure 4.12: *NG Reinsertion packet.*

At last, Algorithm 2 explains how NGCA takes a NG message, generates the NG packet, encodes this data into the JSON standard, and sends this FIXP primitive down to the Data Plane through the FCSID. As the FIXPSW is set for this DHID, NGCA expects to receive these same keys again if a change happens in the network. For instance, a host can move, changing where it is connected to FIXPSW.

4.4 Closing Remarks

The presented prototype for a NovaGenesis-enabled FIXP presents a P4-based Data Plane and a native Control Agent. Therefore, it exploits the state-of-the-art of SDN concepts to leverage a clean-slate FIA with a forwarding framework that supports its novelties. Even though we had to modify the original NG protocol standard to insert new reserved bytes, this mechanism is flexible enough to be later exploited in future applications.

This proposal broadens the PGCS scope to manage a programmable Data Plane without disobeying any of its original concepts. Remember that originally PGCS is an NG core service for gateway, proxy, and controller. This

Algorithm 2 FIXP Primitive Packet Generator

```
1: Retrieves received NG message from GW Queue
2: Fragments the NG message
3: while Fragments to send do
4:     Creates the FIXP Reinsertion packet
5:     Select the FCSID raw socket to send the packet
6:     while Try to send the packet for 2 minutes do
7:         Tries to send through sendto function with FCSID raw socket
8:         Break
9:         if Packet sent successfully  then
10:            Problem = false
11:        else
12:            Break
       Deletes NG original message from its memory;
13: Return Problem;
```

fact means that it can translate NG messages to any link-layer standard, represent NG services to other domains, and, most importantly, control any programmable device. Regardless of its more than a decade of existence, NG cornerstones are still valid and relevant today.

Considering what has been proposed, this is the first step for a NG stateful routing table with programmable devices. Upon supporting NG native concepts, the Control Agent is an integral component for this architecture. For instance, it takes advantage of its naming structure to build the required knowledge to manage the underlying FIXP Data Plane.

Chapter 5

Evaluation Methodology

THIS chapter presents the evaluation methodology to assess the NovaGenesis Control Agent (NGCA) for Future Internet Internet eXchange Point (FIXP). In this way, it covers the experimentation methodology, the different virtualized network topologies and scenarios employed, the NG application applied under these scenarios, and the performance metrics overseen to validate this work.

5.1 Applied Tools

In this section, we explore and highlight every tool used or developed to acquire the data related to the execution of the FIXP test.

5.1.1 Wireshark and Zabbix

Wireshark is an open-source network packet analyzer [102]. This software captures the packet data in selected network interfaces in considerable detail. In this evaluation, Wireshark provides a network log based on the packets captured and a Python script filters and assess this data.

Zabbix is an open-source monitoring software supported by Zabbix SIA [103]. This tool oversees network and host metrics through Zabbix servers and Zabbix agents, presenting several ways to visualize the acquired data graphically. For example, this software can monitor the data traffic at a network interface, the CPU load, the CPU utilization, and the memory utilization on servers, virtual machines, applications, or websites. In this evaluation, it presents the Zabbix agents throughput.

5.1.2 Python and some related packages

In order to develop some additional tools for evaluating the FIXP environment, the Python programming language was chosen due to its massive

community support, ease of usage and remarkably customization of packages for several scenarios. The present methodology applies Python 3.8.2.

Scapy [104] focuses on the packet manipulation, allowing to create scripts that interacts directly with OS sockets to deal with network traffic. In this evaluation, Scapy 2.2.0 package version is applied.

Considering the data analysis, *pandas* [105] and NumPy [106] are Python packages focused on data manipulation. Pandas is broadly applied on several fields, such as finances, statistics, and engineering. Moreover, it provides a fast framework to deal with massive data volume, since its programming is based on the union of C and Python. Besides, it also handles diversified types of data sources and outputs handy, such as databases, Excel spreadsheets, or Comma-Separated Values formats. This work applies pandas 1.2.3 and numpy 1.20.1 for its data analysis.

Finally, *Matplotlib* [107] has been applied for the graphical representation and visualization. Besides, this is an open-source tool to develop charts, avoiding the use of proprietary software like *Matlab*. This works exploits the Matplotlib 3.4.2 version for data representation in this thesis.

5.2 Evaluation Scenarios

Overall, the NG Control Agent (NGCA) manages any number of underlying FIXPs, regardless of the NG application. The presented tests encompass the performance and functional evaluation on a controlled network topology, depicting a virtualized Internet Provider that presents a variable Autonomous System (AS) to connect NG clients.

Firstly, this sub-chapter presents a standard topology that interconnects the NG peers at the same network, namely Standard Topology. Following this, additional use cases present FIXP as an exchange point for connecting the same NG clients. Therefore, we can evaluate the impacts of exploiting FIXP against a virtualized network topology without it.

5.2.1 Standard Virtual Network Topology

The Standard Topology emulates a direct connection between the NG peers. In this, VirtualBox virtualizes the NG hosts and a single virtual network connects them. Figure 5.1 illustrates the generic standard topology.

Figure 5.1: *Standard Topology to Set the Applications Up.*

This scenario has two strategical goals. Firstly, we need to figure out the best NG parameters to enable the NG applications, presenting results with the best QoS possible, minimizing delays, and boosting NG responsiveness. Moreover, we can contrast the Standard Topology result with the FIXP experiments, validating the impact of FIXP in NG applications.

5.2.2 FIXP Topologies

The second scenario depicts an Internet Provider's AS that exploits FIXP as an interconnection point. This means that FIXP Data Plane (DP) connects the NG clients, enabling their communication. As a result, NGCA must set the underlying FIXP Switches dynamically, scattered over the network domain. Figure 5.2 illustrates a generic FIXP scenario.

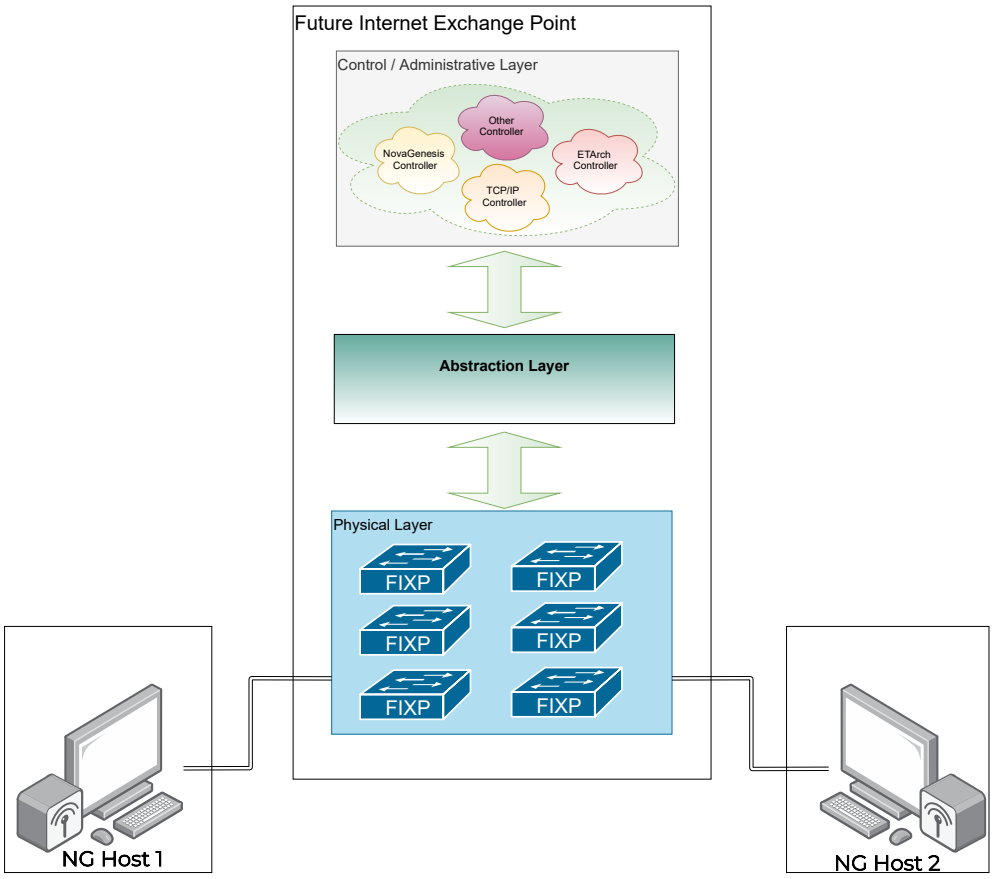

Figure 5.2: *Example of a network topology with FIXP as an exchange point.*

In this hypothetical scenario, there are two NG hosts connected by FIXP, which forwards NG packets through several FIXP switches at FIXP DP. Meanwhile, these forwarding elements request guidance from the NGCA at FIXP Control Plane (CP) for unknown packets. Based on this scenario, we can evaluate FIXP impact and the NG Control Agent performance.

5.3 NovaGenesis Content Repository and Distribution Application

This work exploits an NG application to validates the effectiveness of the NGCA and the FIXP impact. In other words, our goal is to analyze the overhead and overall costs of taking advantage of a P4-enabled forwarding device and a native NG SDN controller. In this sub-chapter, we present the NG application setup and the required settings to fulfill this experiment.

5.3.1 Setup

An ISP connects three NG hosts spread in its AS. These hosts use the Content Repository and Distribution Application (ContentApp), where one NG Source hires one NG Repository, requesting temporary storage for its content. Moreover, one NG Network Cache replicates the content closer to the NG Repository, depicting an ICN web concept through its NRNCS.

This ISP connects the three NG hosts by four different network topologies. Figures 5.3 and 5.4 depict the proposed evaluation scenarios. Notice that every NG Source is on the left-hand side of the picture, presenting the PGCS and the ContentApp with the Source role. On the other side, the NG Repository also takes advantage of the PGCS and ContentApp with the Repository role. Meanwhile, the NG Network Cache is at the bottom of each picture, exploiting the core services PGCS, HTS, GIRS, and PSS. These last three NG services are grouped under the NRNCS box.

In addition to these NG hosts, the network topologies encompass two different cases. The first is the Standard Topology, in which the peers have direct communication. Differently, the next three use cases present FIXP as a future exchange point, forwarding the received Ethernet packets to their destinations. At the FIXP DP, the number and arrangement of FIXP Switch changes. In Figure 5.3, there is only one programmable forwarding device at the left-hand side picture, while the right-hand side presents three P4-based switches. In Figure 5.4, the left-hand side picture exploits three FIXP Switches and the right-hand side presents five forwarding devices.

In every case, there is a varying number and size of locally stored photos (.jpg or .jpeg) in the Source. Considering a case that does not fragment NG messages, NG Source publishes 2500 photos of 780B are exchanged with average throughput of 33 kbps. Meanwhile, another exploits 1000 photos of 10kB with average throughput of 100kbps that fragments NG messages. It is important to highlight that these two type of photos are adequate to validate our proposal and BMv2 is not advised for greater throughput.

Focusing on the ContentApp presents some delays and hyperparameters to enable this communication. In specific, NG Source's PSS publishes the data to the NG Repository and the NG Network Cache in the form of data

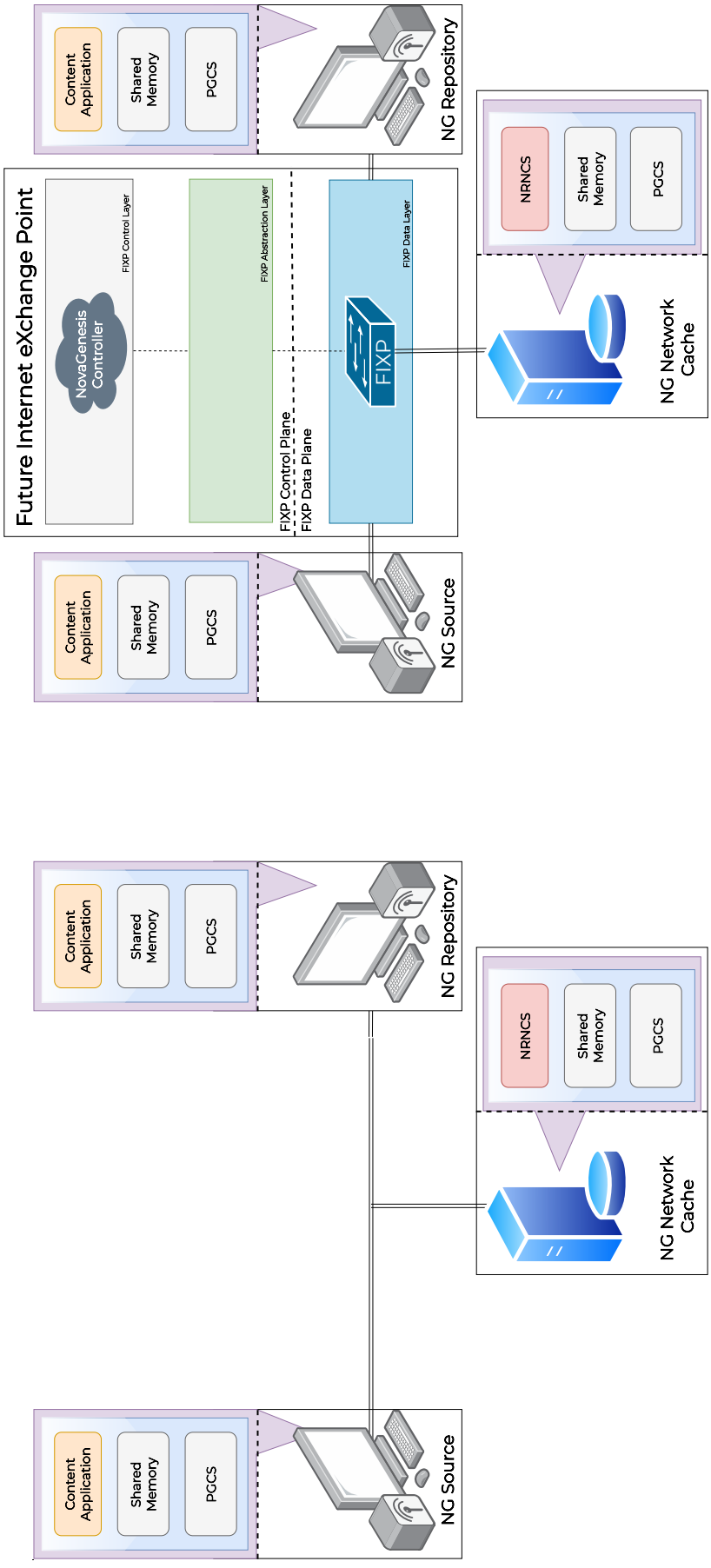

Figure 5.3: *FIXP Evaluation with the Standard Topology and another with 1 FIXP Switch.*

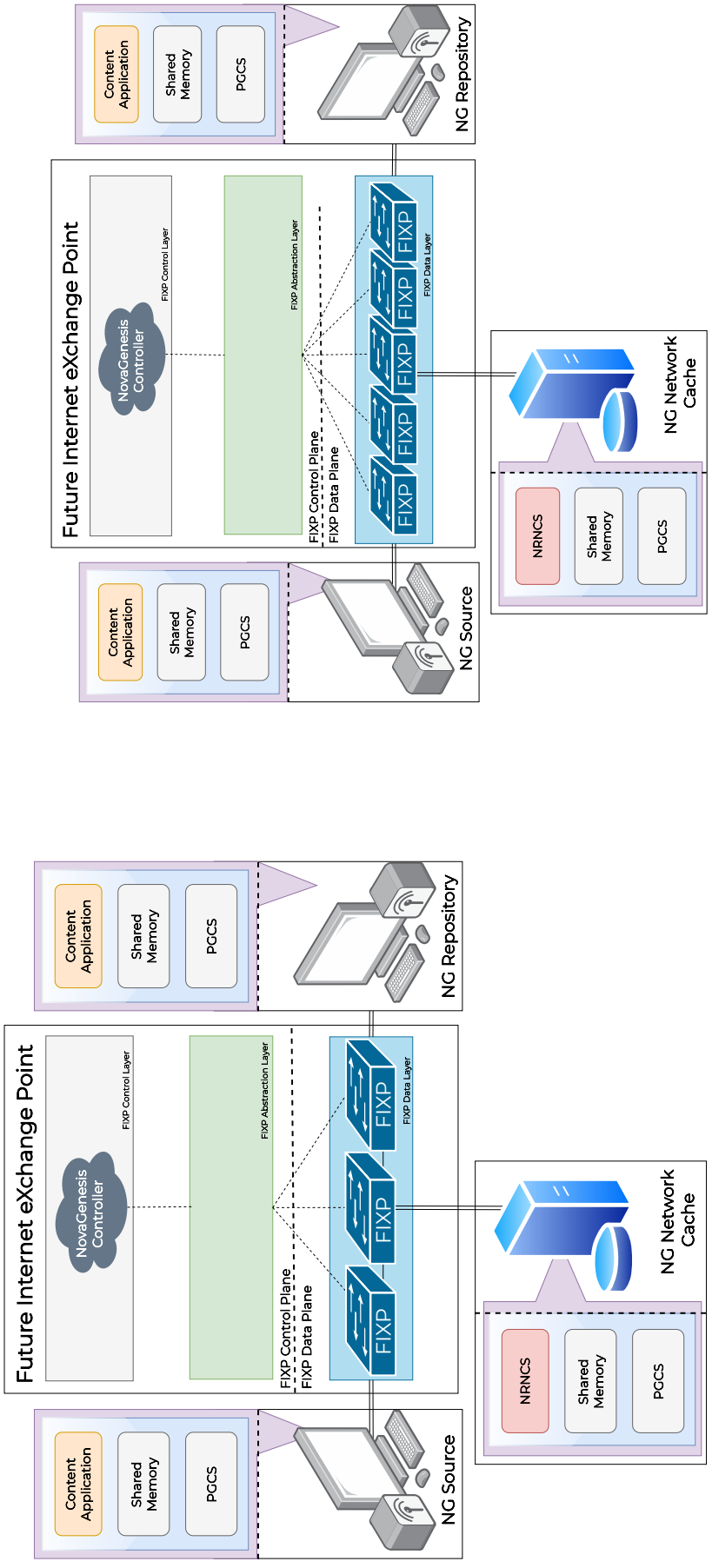

Figure 5.4: *FIXP Evaluation with 3 and 5 FIXP Switches between NG hosts.*

bursts. Before transmitting a photo, the repository sends a subscription request, which allows determining if the NB related to a file has been delivered or not. Besides, PSS provides specific time to live policies to avoid scalability issues, freeing HTS storage from unused bindings or content [108]. Also, published content can be revoked by publishers.

In summary, the goal is to exploit the NG and FIXP novelties regarding the communication among NG peers. The evaluation scenarios exchange named-content of photos between the players and the FIXP infrastructure aids in forwarding and routing this data. Nevertheless, the NG core life cycle remains intact. First, NG initializes its core services in each host. After this, it starts up the source's and repository's content distribution applications. Upon fulfilling this, the exposition of services begins, wherein it exchanges NG Hellos as broadcast messages. Following, each peer discovers possible NG peers. Afterward, the content source application offers a SLA to the discovered repository applications to establish the terms of operation. Then, the content repository application accepts the contract (in the current version, contracting clauses have not been include). At last, the source publishes and notifies the contents to the repository [108].

5.3.2 NG Parameters

Regarding the ContentApp, NG requires some hyperparameters to enable the data exchange between the NG Source and the NG Repository. This configuration is made through each *App.ini* in each peer with the following parameters. Table 5.1 delineates the chosen parameters for each photo file size following the same letter code from the list. These values are essential because they impact on the NG performance and efficiency. These hyperparameters establish two distinct test scenarios, which are not comparable. Each case presents a different throughput and dynamics. These values were chosen based on tests to optimize the NG performance in terms of packet loss and round-trip delays.

a. **Delay Before Discovery**: NG applications use this parameter to add a delay before offering a service for the application exposition, i.e. it's no use discovering peers if an application hasn't had time to finish the exposure phase. After this delay, applications start subscribing keywords to discover possible peer applications to work together. This delay is relevant for establishing contracts between applications.

b. **Delay Before Publishing a Service Offer**: This parameter introduces a delay between the moment when NGs' applications discover a possible partner and send them a service offer. In other words, the NG application continuously subscribes to several NBs and determines whether any set of NBs corresponds to a possible partner. Whenever this occurs, the application sends a service offer to a dis-

covered candidate peer in the form of an SLA. Therefore, the Delay Before Publishing a Service Offer takes place between the moment a NG source content application discovers a candidate partner (repository) and its PSS notifies the candidate about a proposal information object temporally stored in an HTS. Through this delay, NG guarantees that any application can generate the SLA invite with all the necessary data to the partner. This parameter is crucial for establishing contracts among applications.

c. **Delay Before Run Periodic**: NG infrastructure performs a periodic set of tasks at given times. For instance, the periodic discovery messages exemplify this concept, wherein NG applications send these types of messages from time to time to discover new peers. In other words, this can derive as the NG heartbeat (remember, the NG GW operates in an event-driven way), where it controls how often all its critical tasks execute. Therefore, the Delay Before Run Periodic parameter controls the execution of periodic tasks. This delay is relevant for establishing contracts between applications.

d. **Delay Before a New Peer Evaluation**: Depending on the amount of information obtained, NG can use this parameter to anticipate the evaluation of a possible partner without resorting to periodical routines. In other words, NG can hasten its processes to establish a SLA when it gathers the required NBs in its HTS. Hence, the Delay Before a New Peer Evaluation enables a new evaluation of a possible partner, regardless of the periodic delay. This time interval is also meaningful until the contracting phase.

e. **Photo Burst Size**: After establishing the SLA, the application begins to scan the directory where the source stores the data to send to the contracted repository. The Photo Burst Size value narrows the maximum number of possible photo publications per NG message. As a result, the application creates a queue of content based on the burst size from time to time. This value is critical to avoid system overload and publication delays due to the data exchange between NG processes. Upon restricting the number of content in each publication, NG guarantees the ideal flow rate for the application.

f. **Delay Before a New Photo Publish**: Alongside the Photo Burst Size parameter, this parameter establishes the time interval between each photo burst publication. Meanwhile, this parameter is crucial to evade the system's overload and a peak of internal delays.

Each PGCS requires an initialization to enable the NG communication over a legacy link-layer technology. The following Listing 5.1 exemplifies a bash script for this process, with a list below that describes the PGCS initialization parameters. This example establishes the communication port *0* for an NG intra-domain communication with the Ethernet protocol. Re-

Table 5.1: *Content Repository and Distribution Application Hyperparameters Settings for variable photos sizes.*

Scenario	Quantity	**a**	**b**	**c**	**d**	**e**	**f**
Photos of 780B	2500	10	20	40	25	0.7	3
Photos of 10KB	1000	10	20	40	25	0.3	1

garding the NG peer, "*-pc*" points that the PGCS host presents the NG Core and wants to communicate with a known peer that has the MAC address of "*00:00:00:00:00:01*" in its domain, using the *Eth0* network interface to send Ethernet packets with up to *1400* bytes of MTU.

Listing 5.1: *PGCS Initialization Example.*

```
1 ./PGCS ./ 0 Intra-Domain Ethernet -pc Eth0 00:00:00:00:00:01
    1400
```

a. **NG Service Path**: This parameter establishes the path of the executable program of the desired NG service.

b. **Path to Store Operation Logs**: This aspect points to a path where the operation logs of the NG service is going to be stored.

c. **Origin Communication Port**: This instruction delimits the communication port for the NG service.

d. **NG Role**: This field sets the NG communication domain, which can be the string value "Intra-Domain" or "Inter-Domain."

e. **Link-Layer Technology**: This specification instructs PGCS for translating the NG message for a underlying link-layer technology. For example, this can be "Ethernet," "UDP," or "LORA."

f. **Option**: This setting establishes an NG communication mode, which can be "-p" for a direct communication with known NG peers, "-c" for enabling NG core services, and "-d" for an automatic communication where a NG host discovers other NG peers.

g. **Network Interface**: This string value refers to which network interface PGCS will use for its communication, which can be a real or virtualized network interface on the host system.

h. **Destination Address**: For the "-p" option, this field determines which NG peers are joining the communication. This can be a MAC, IP, or any other link-layer technology address.

i. **MTU**: This value restricts the maximum communication packet length that PGCS can conceive in bytes.

5.4 Performance Evaluation Methodology

This sub-chapter covers the evaluation methodology for NG ContentApp scenario, FIXP infrastructure, NGCA, and the host performance.

5.4.1 NovaGenesis

Considering the ContentApp scenario, NG deploys some mechanisms to evaluate its performance. Some of the available metrics are:

- **Instantaneous RTT**: It encompasses the publication or subscription RTT of *Photo Burst Size* ammount of content in the ContentApp scenario. For example, if this parameter sets 5 photos per burst, the RTT compresses the required time to publish or subscribe 5 photos. Figure 5.5 depicts the generic network flow between the required NG hosts. Notice that the FIXP DP has been simplified to just a single *FIXP.P4*, yet this can represent more than 1 forwarding device. Moreover, it is crucial to highlight that this step happens after the complete establishment of the SLA between the NG hosts.

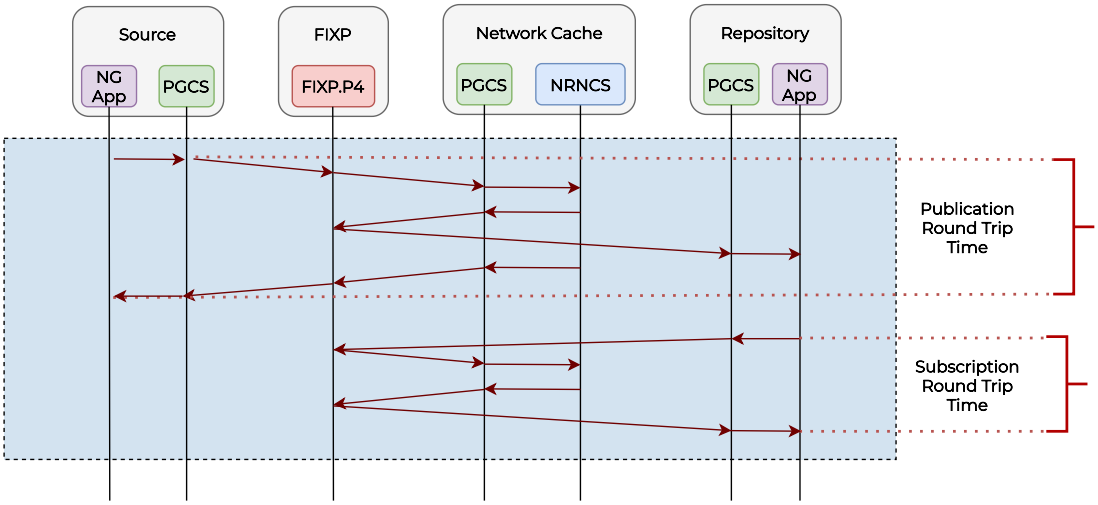

Figure 5.5: *Content Performance Metrics.*

- **Arithmetic Mean of RTT**: It is the simple average of of the RTT. Hereafter, each RTT is labeled as *x*. The arithmetic mean of x is calculated by Equation 5.1, where *x* represents the instantaneous values obtained by each sample and *N* the size of the samples population:

$$\mu = \frac{\sum x}{N} \tag{5.1}$$

- **Standard Deviation of RTT**: The standard deviation is an statistical measure that determines the expected variation or dispersion of the *x* [109]. The standard deviation is calculated by the Equation 5.2,

where x represents the instantaneous values of *RTT*, μ the samples average, and N the population of samples:

$$\sigma = \sqrt{\frac{\sum(x-\mu)^2}{N}} \tag{5.2}$$

- **Standard Error**: This statistical measure represents the deviation between the average of the sample from the sample's average population [109]. In other words, it measures the accuracy between the samples average and the overall population average. Mathematically, Equation 5.3 represents this measure taking into account the standard deviation (μ) and the population size (N):

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}} \tag{5.3}$$

- **Confidence Interval (CI)**: This statistical measure establishes a range wherein it is expected to find a certain amount of samples within these limits. Moreover, there is a confidence level that narrows or broadens the interval range by a probability [109]. For NG, the algorithm approximates the obtained samples to a Gaussian distribution. Therefore, the constant *1.96* represents this approximation, and σ_μ is the Standard Error. Mathematically, Equation 5.4 depicts this measure:

$$CI = 1,96 \cdot \sigma_\mu \tag{5.4}$$

- **Lower Limit**: This value expresses the lower limit of the CI. The Equation 5.5 delimits how to calculate this:

$$\mu_{low} = \mu - CI \tag{5.5}$$

- **Upper Limit**: This delimits the upper limit of the CI. The Equation 5.6 depicts how to calculate this:

$$\mu_{up} = \mu + CI \tag{5.6}$$

5.4.2 FIXP Delay Profile

Focusing on the FIXP infrastructure, its delay profile presents the time overhead demanded from FIXP. In this scope, Figure 5.6 aims at explaining and highlighting each specific delay covered.

Based on Figure 5.6, there are two types of delays. The first one is the propagation delay between each layer [1]. On the other hand, the processing

[1] Every time that we mention a propagation delay, throughput, and other network-related

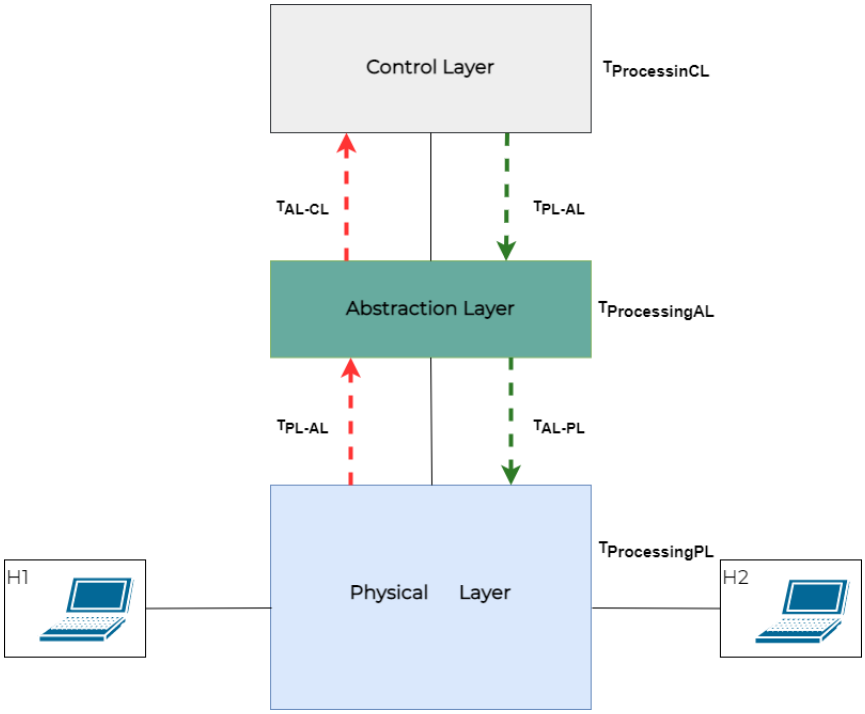

Figure 5.6: *FIXP Time Profile.*

delays are the time that it takes for one FIXP agent to process the received packet and transmit it to its destination. For example, one of the processing delays of the PL encompasses the time that the FIXP Rule Handler Service takes to set its FIXP SW and generate the FIXP Acknowledgment Packet. Bellow, there is a list of the considered delays to trace a FIXP delay profile:

- **Propagation Delay between the PL and AL**: This metric focuses on the propagation time for one packet from the Physical Layer to the Abstraction Layer and vice-versa. As the scenarios exploits virtualized environments, this result covers the propagation between VMs.
- **Propagation Delay between the AL and CL**: This metric covers the propagation time that a packet from the Abstraction Layer to the Control Layer and vice-versa. As the scenarios exploits virtualized environments, this result covers the propagation between VMs.
- **Processing Time per FIXP Switch**: This is the average time to forward a packet for a FIXPSW. Equation 5.7 outlines this metric.

$$FIXP_{ProcessingTime_x} = \frac{\sum(ProcessingTime)}{Total_{Packets}} \tag{5.7}$$

metrics in this work, we refer to what happens between VMs. Therefore, any delay is not relative to the electromagnetic waves in the air or the fiber. In other words, it refers to the processing and propagation of packets inside the O.S., VirtualBox, and processes. On the other hand, future works can present this conventional delay by exploring P4-enabled forwarding devices, as from Edgecore, NetFPGA, and SUME.

- **Processing Time per Scenario**: This metric encompass the average processing time per scenario. Equation 5.8 highlights this metric.

$$FIXP_{ProcessingTime_Scenario} = \frac{\sum(ProcessingTime_x)}{Total_{FIXPSW}} \quad (5.8)$$

- **FIXP Abstraction Layer Processing Time**: This metric comprises the processing time that the FCPH or FSPH takes to forward a packet either to FIXP DP or Control Layer.
- **FIXP Table-Add Delay**: This metric comprises the delay that a FIXP Table-Add Modify primitive takes to set a FIXP SW and receive its Ack.

5.4.3 Data Assessment Methodology

A Python script automates the data assessment. Beforehand, the user must convert Wireshark logs as JSON for each experiment. In other words, the user must select every network interface for each distinct VM in FIXP environment . As a convention, the name structure follows: ”<Day of Experiment>-<Type of Experiment>-<VM name>.json.´´

After assembling this crucial data in a single directory, the ”sumPackets.py´´ script analyzes the collected data. The box below outlines the command to execute this, taking the folder directory as an input:

```
python3 ./sumPackets.py <Wireshark logs directory>
```

This software summarizes every distinct log into a single spreadsheet file with the extension ”.xlsx´´ that contains eight sheets. These sheets represent each aspect of the FIXP packet flow and results, namely:

- **Filtered Packets**: It outlines every packet chronologically, considering every VM and the FIXP and NG architectures.
- **Filtered Table Adds**: This sheet filters the FIXP control packets by their flow from the NGCA, organizing them by their SeqID.
- **Filtered Acks**: This sheet summarizes the FIXP packets, considering their flow from the PL to the controllers, organizing them by SeqID.
- **Filtered Reinsertion**: This sheet compiles the FIXP reinsertion packets, gathering only the FIXP Reinsertion packets.
- **Table Add-Modify Primitive**: This sheet sums up the FIXP Table Add-Modify Primitive, organizing them by their SeqID.
- **Reinsertion Primitive**: This sheet arranges the FIXP Acknowledge and FIXP Reinsertion packets, organizing them by their SeqIDs.
- **Architectures and Reinsertion Packets**: This sheet organizes the FIXP Reinsertion primitive and the original packets from the AL to the PL, organizing them chronologically.

- **FIXP Delay Profile**: This sheet summarizes the FIXP Delay profile.

To elucidate the ”sumPackets.py´´ script, Figure 5.7 illustrates its simplified flowchart. Notice that it takes the logs directory as an input to read, filter, and summarize the FIXP Delay Profile. This script considers only the FIXP and NG. After doing this process for every log, the algorithm calculates the considered delays based on the already explained sheets. In the end, it generates the results spreadsheet containing the FIXP Delay Profile.

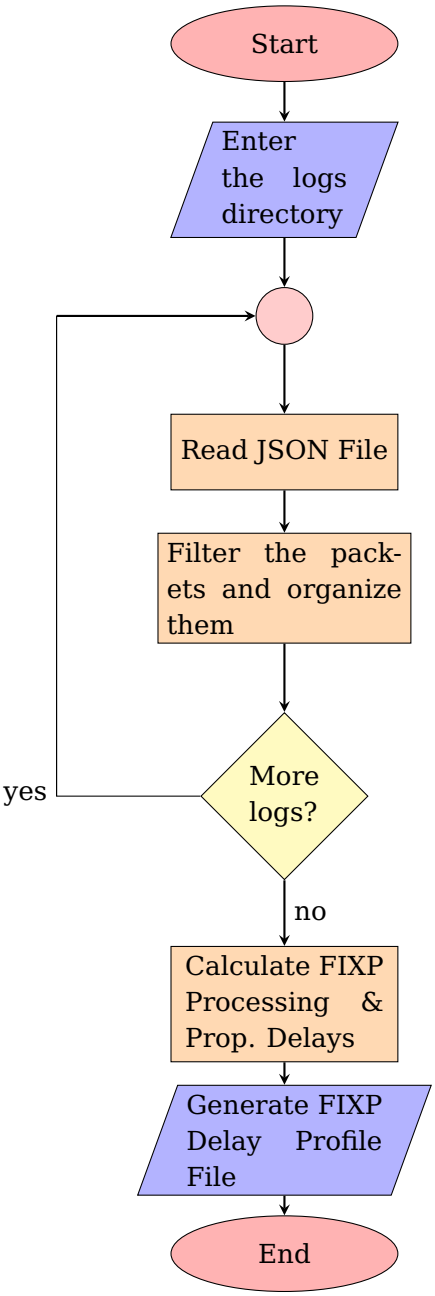

Figure 5.7: *FIXP Delays Profile Flowchart.*

5.4.4 Host Machine Evaluation Methodology

Another investigation concerns the host performance to virtualize the network environment. For these results, a bash script oversees the CPU and memory usage over the varied network topologies. Algorithm 3 highlights the infinite loop to measure the CPU and memory usage from the host machine and store these data into a text file every 10 seconds. Notice that *top* is a Linux command that summarizes the real-time information from the running processes and *free* relates to the memory usage.

Algorithm 3 CPU Usage Algorithm

```
1: while True do
2:     CPU = 'top -bn 2 | grep 'Cpu(s)' | tail -n 1 |awk 'print $2+$4+$6''
3:     MEMORY = free -t | awk 'NR == 2 print 3/2*100'
4:     Stores CPU and Memory variable into text file
5:     Sleep for 10 seconds
```

In addition, Zabbix retrieves the network throughtput graphically. Through this software, it is possible to observe the incoming and outgoing data in a network link/network interface. For the FIXP scenarios, the graphs present the FIXP Data Plane and FIXP Control Plane throughput considering the network traffic between VMs on VirtualBox's virtual networks.

5.5 Functional Evaluation Methodology

Increasing the complexity of the AS, a multi-path between the NG hosts is proposed in Figure 5.8. The goal is to observe how NGCA behaves with the current solution in a more complex environment, wherein it might lead to a non-optimal path between the NG hosts. This last case is just for curiosity's sake, evaluating the NG Control Agent performance in a mesh network.

5.6 Computational Resources

Throughout the experiments, we have selected the available ICT Lab server as the Host Machine. This server virtualizes the proposed FIXP network topologies. Table 5.2 briefs the Host Machine hardware.

Table 5.2: *Host Computational Resources.*

Role	Description	CPU	Hard Disk	RAM Memory
Host	Dell PowerEdge 7640	2x 2.2GHz Intel R Xeon TM Silver 4114 32 Core	2x SSD SATA 480GB 6 Gbps; 3x HDD SATA 4TB, 7.2K RPM	256GB (8x32GB) RDIMM DDR4 2667 MT/s

VirtualBox provides the required virtualization tools to conceive an emulated network topology with VMs and virtual networks. Considering NG and FIXP resources, Table 5.3 outlines the computational settings:

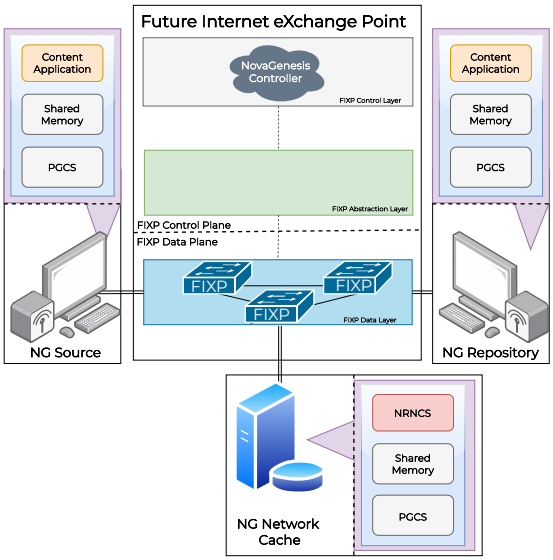

Figure 5.8: *FIXP Evaluation with a multi-path between NG hosts.*

Table 5.3: *Virtual Machines Computational Resources.*

Role	**CPU**	**Hard Disk**	**RAM Memory**
NG Hosts	3 cores of the 2.2GHz Intel Xeon Silver 4114 processor	10GB	32GB
FIXP Switches	3 cores of the 2.2GHz Intel Xeon Silver 4114 processor	20GB	32GB
FIXP Abstraction Layer	3 cores of the 2.2GHz Intel Xeon Silver 4114 processor	10GB	32GB
NG Control Agent	3 cores of the 2.2GHz Intel Xeon Silver 4114 processor	10GB	32GB

5.7 Closing Remarks

The presented methodology combines custom software and existing frameworks to evaluate the FIXP, NG, and host performance over the proposed virtualized network topologies. In this way, this assessment provides a broader panorama of how NG behaves on a virtualized environment exploiting FIXP as an exchange point. As we are handling a data analysis of a massive database of Wireshark and NG network logs, the automation of this process through Python scripts is crucial to filter, process, and present the results. For example, the Wireshark network logs have files with more than 1GB of size. As the network topology grows, it becomes unbearable for a human to examine this massive database handly and promptly.

At first, the FIXP scalability is investigated as the virtualized network topology grows. Nevertheless, this is just a partial analysis once a definitive conclusion requires a much more substantial environment. Secondly, the NG analysis contemplates the overhead impact on the ContentApp scenario. Upon comparing virtualized network topologies that present the NG hosts connect through and without FIXP, we can contrast both scenarios. Finally, the influence of such tests is analyzed in the host.

Chapter 6

Results

GIVEN the proposed methodology, it exploits a virtualized network topology to evaluate FIXP. Therefore, this chapter the obtained performance and functional results, analyzing the findings.

6.1 Performance Evaluation

This sub-chapter presents the obtained performance results from NG, FIXP, and the Host machine. Every graph but Zabbix's represents an average of 10 samplings, summarizing around 1GB of data for each plot.

6.1.1 NG Performance

Focusing on the NG performance, the first results encompass the Pub RTT from the NG Source to the NG Network Cache and the Sub RTT from the NG Network Cachee to the NG Repository. As seen in Figures 6.1-a) and 6.1-b), the FIXP impact is minimal from the application point of view.

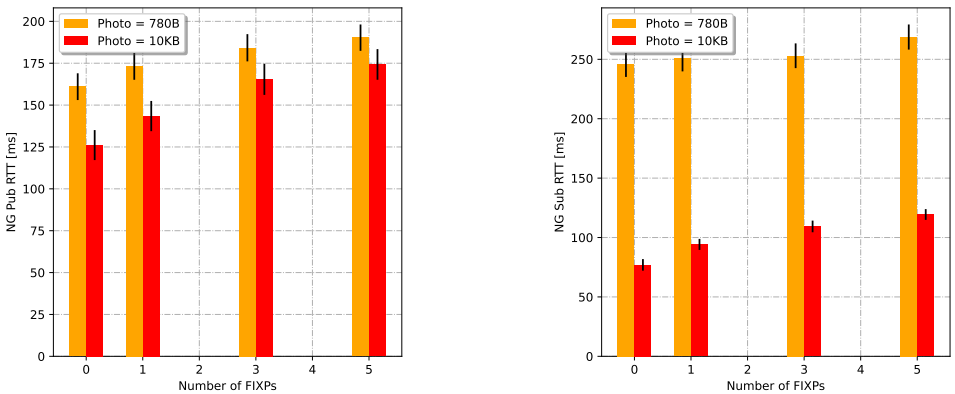

Figure 6.1: *NG Performance Evaluation considering: a) Pub RTT and b) Sub RTT.*

Considering the Standard Scenario, i.e. without FIXP, NG experiences a 161 ± 7.988 [ms] for publishing and 246.11 ± 10.93 for subscribing photos of

780B. Meanwhile, the publishing values are 173.25 ± 8.14, 184.24 ± 8.13, and 190.26 ± 7.87 [ms] for 1, 3, and 5 FIXP Switches, respectively. On the other hand, these same values with a fragmentation scenario are 126.12 ± 8.959, 143.46 ± 9.02, 165.41 ± 9.31, and 174.27 ± 9.147 [ms]. Notice that the RTTs have almost a linear behavior as the FIXP Data Plane increases its number of FIXP switches. Surprisingly, the Sub RTTs presents a bigger variation when we compare the subscribing of 780B and 10KB photos and largest RTT values for subscribing 780B. This trend might be due to the choice of applying three separated NG services (PSS, HTS, and GIRS) to create the NG Network Cache. Therefore, NG intra-communication process may not be optimized. Moreover, these results might reveal that the ContentApp settings (Table 5.1) are not the best as we thought. Meanwhile, VirtualBox's processes and virtual networks can also impact these values once it exploits its hypervisor to manage the communication between VMs.

Figures 6.2, 6.3, and 6.4 illustrate the NG hosts throughput for the scenario with photos of 780B. NG has a gradual increase of traffic while exposing, discovering, and establishing a contract with the NG peers. After this, NG Source starts to send the photo content, increasing the throughput until a peak. This value decreases over time until the moment when NG Source publishes all of its content to the NG Network Cache. The remaining network traffic derives from the periodic NG messages, like NG Hello. For photos with 10kB, the throughput follows the same pattern.

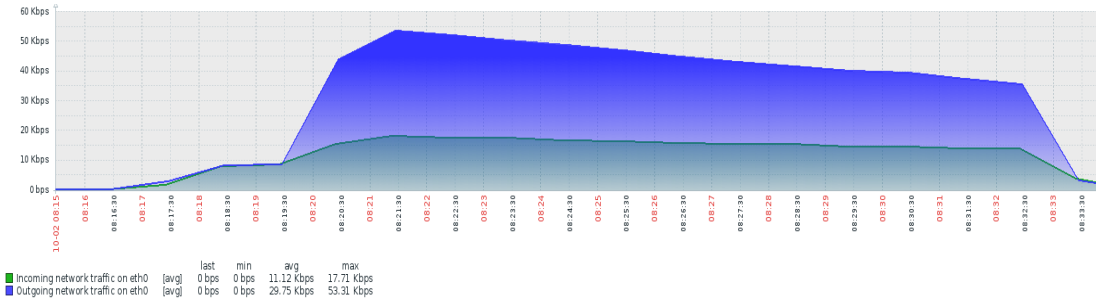

Figure 6.2: *NG Source Throughput with photos of 780B.*

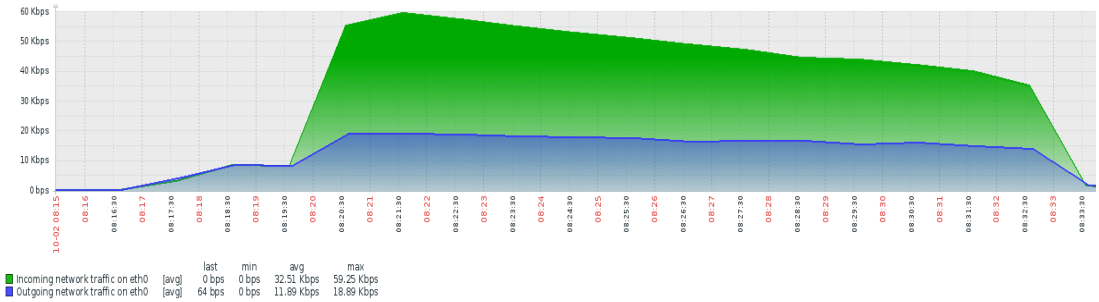

Figure 6.3: *NG Repository Throughput with photos of 780B.*

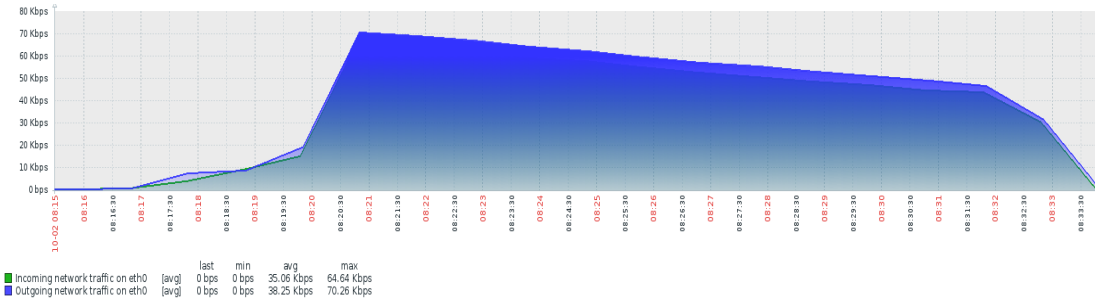

Figure 6.4: *NG Cache Throughput with photos of 780B.*

6.1.2 FIXP Switches

Considering the FIXP DP performance, Figure 6.5-a) outlines the average of FIXP Switch Processing Time per scenario. This graph summarizes the amount of time required to forward an NG considering the whole FIXP DP. This means that the x-axis represents the total number of FIXP Switches at the FIXP DP. Apparently, the scenario with photos of 780B performs worse than 10kB in any case. Moreover, the outcomes decrease over the increase of FIXP Switches, which may indicate a poor sampling for the smaller cases once the greatest scenario has more data.

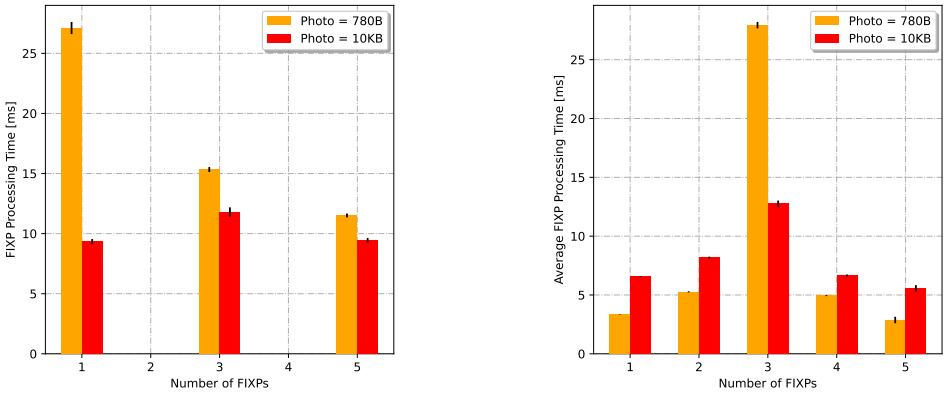

Figure 6.5: *FIXP Switch Processing Time considering: a) Average per Scenario and b) Average per FIXP Switch.*

Figure 6.5-b) contrasts the average FIXP Processing Time per FIXP Switch. This graph outlines the expected overhead inserted by each forwarding element at FIXP Data Plane, once it groups every network log per FIXP Switch, generating an x-axis that represents each FIXP Switch at FIXP DP. Opposing the last graph, this result points that in most cases the 780B photos performance is better than 10KB. This is because this scenario without fragment does not require to read the hash function to retrieve the DHID based on each NG message MsgID. Therefore, FIXP Switches forward this type of packet directly. Every FIXP Switch but FIXP Switch 3 presents a delay of less than 10 [ms]. Nevertheless, FIXP Switch 3 presents a higher overhead and shows that the 780B photo forwarding performs worse than 10KB. This

might explain why the last graph was biased towards the smaller content. As this forwarding element is at the center of the network in every scenario, connecting three NG hosts and FIXP Control Plane, this might be a network bottleneck. As the network traffic is substantial in this forwarding element, the current solution may burden its forwarding. As pointed in p4language Github issues [110–112], several factors impact the expected BMv2 throughput. As the current solution presents the JSON source code and VMs to conceive the network topology, this justifies how FIXP 3 struggles with higher network traffic than the others.

Figure 6.6 covers the required time that FRHS needs to translate the FIXP Table Add primitive to a BMv2 standard, set the NG Routing Table, and generate a FIXP Ack primitive. The first Figure 6.6-a) presents the average per scenario, while Figure 6.6-b) outlines the individual performance per forwarding element. Both results highlights an almost constant overhead regardless of the network topology size.

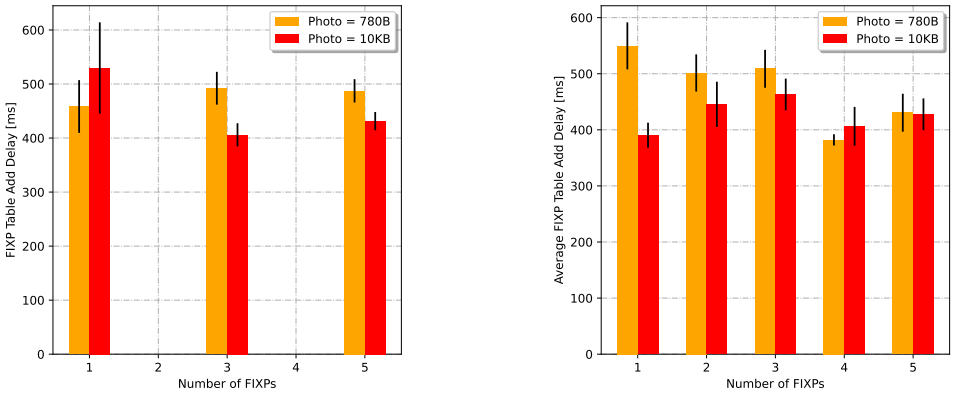

Figure 6.6: *FIXP Rule Handler Service overhead to insert a new rule considering: a) Average per Scenario and b) Average per FIXP Switch.*

Figures 6.7, 6.8, and 6.9 presents the throughput on FIXP 1. Meanwhile, Figures 6.10, 6.11, 6.12, and 6.13 highlights the FIXP 3 throughput. As expected, FIXP Switch 3 receives/forwards more data than any other device. Focusing at FIXP Data Plane, FIXP Switch 1 receives/forwards an average of 29.74/11.58 [kbps] to/from NG Source and 11.59/29.42 [kbps] to/from FIXP Switch 2. At its turn, FIXP Switch receives/forwards an average of 29.35/11.65 [kbps] to/from FIXP Switch 2, 11.5/33.05 [kbps] to/from FIXP Switch 4, and 38.2/35.5 [kbps] to/from NG Cache. Nonetheless, these values should not stress the forwarding device.

6.1.3 FIXP Abstraction Layer

Analyzing the FIXP Abstraction Layer, Figure 6.14 highlights the expected time to forward a FIXP Data Plane packet to the FIXP Control Plane and vice versa upon varying the underlying network topology. As the num-

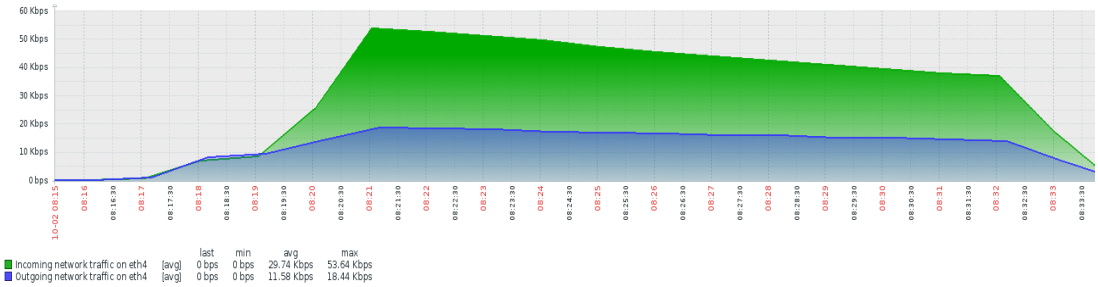

Figure 6.7: *FIXP Switch 1 Throughput from/to NG Source with photos of 780B.*

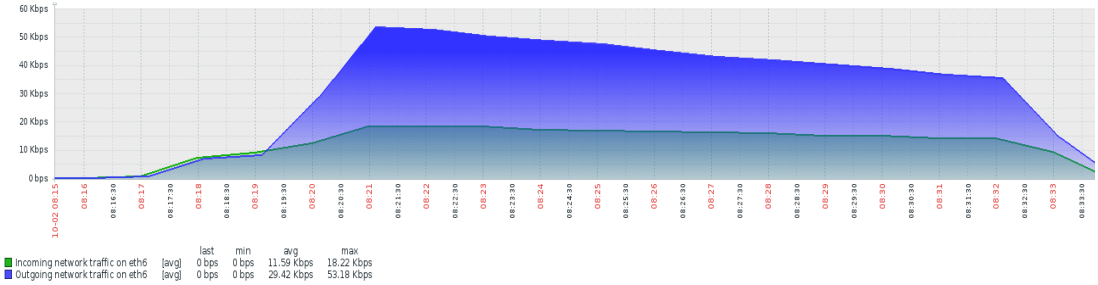

Figure 6.8: *FIXP Switch 1 Throughput from/to FIXP 2 with photos of 780B.*

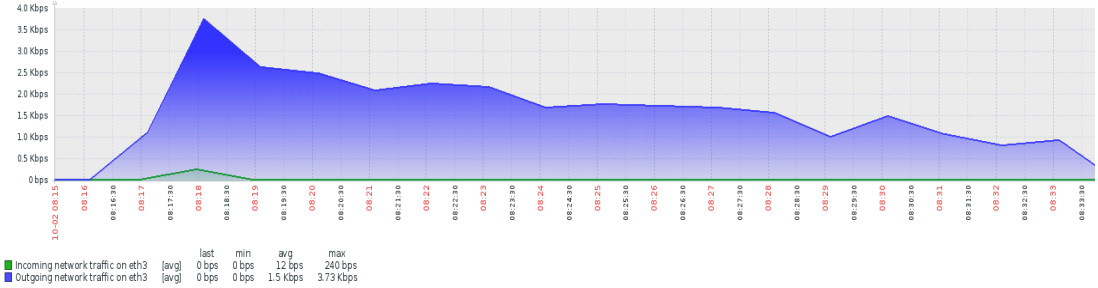

Figure 6.9: *FIXP Switch 1 throughput from/to FIXP Control Plane.*

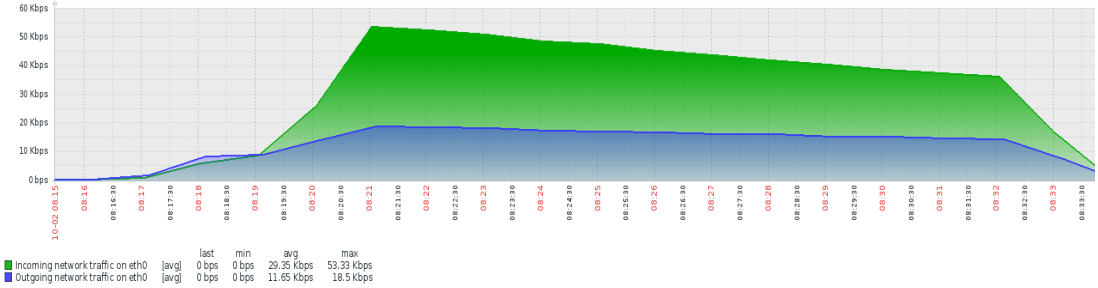

Figure 6.10: *FIXP Switch 3 Throughput from/to FIXP Switch 2 with photos of 780B.*

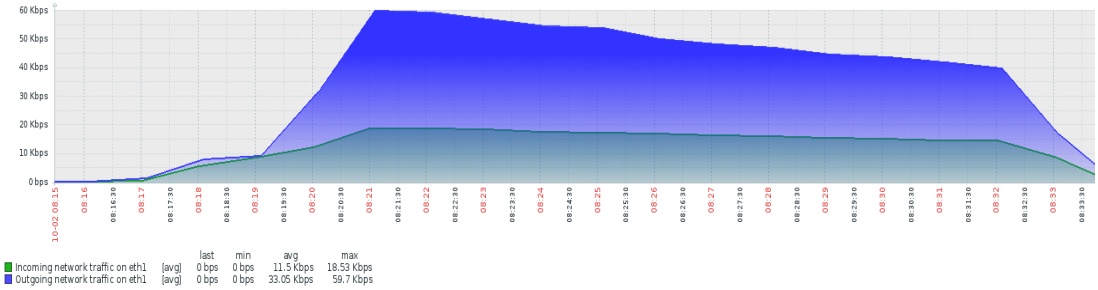

Figure 6.11: *FIXP Switch 3 Throughput from/to FIXP Switch 4 with photos of 780B.*

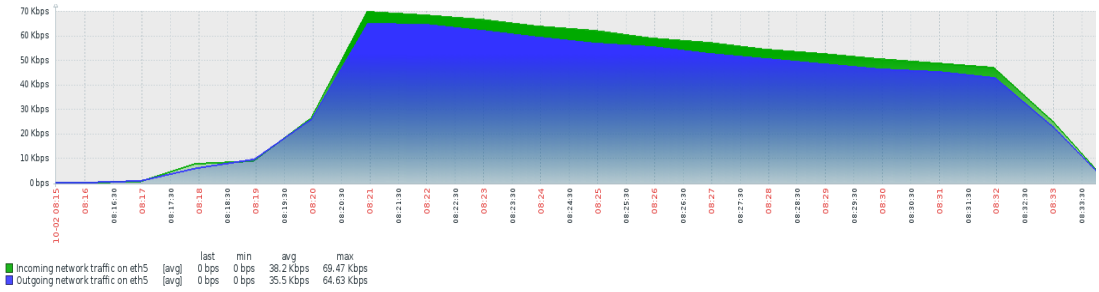

Figure 6.12: *FIXP Switch 3 Throughput from/to NG Cache with photos of 780B.*

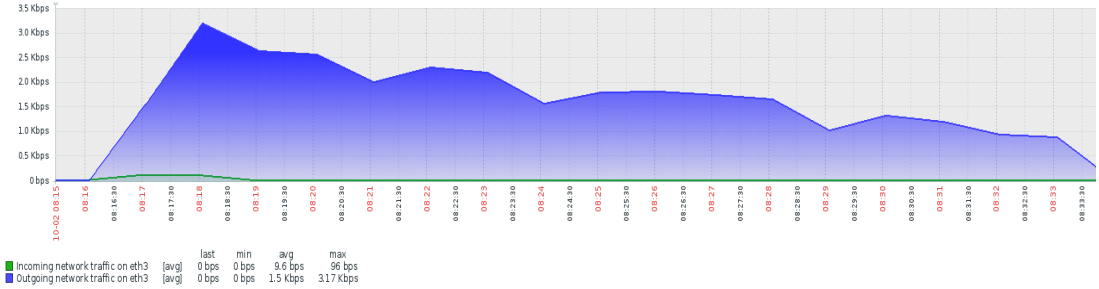

Figure 6.13: *FIXP3 Throughput from/to FIXP Control Plane with photos of 780B.*

ber of FIXP Switches increases on the FIXP Data Plane (and so the management traffic), the overhead in this layer also intensifies. This is probably due to the single thread solution of forwarding packets between the FIXP layers. Nonetheless, this raise seems linear over the scenarios and only impacts during the FIXP Data Plane setting. A possible solution can be a multi-thread algorithm with a buffer to mitigate this possible bottleneck in the FIXP Abstraction Layer. Considering the average for every sampled packet for the 780B and 10KB scenarios, the expected time to forward a packet between the FIXP Data Plane and FIXP Control Layer are 129.9568 ± 0.78254 and 163.0163 ± 1.5557 [ms], respectively. Unsurprisingly, these values does not change significantly because they present the same PGCS MTU limit.

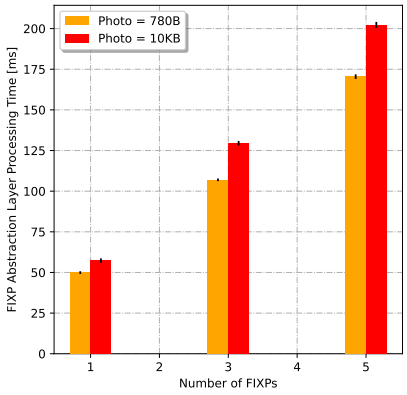

Figure 6.14: *FIXP Abstraction Layer Processing Time.*

Figure 6.15 presents the throughput from/to the FIXP Switch 3, while Figure 6.16 outlines the throughput from/to NGCA. It is important to high-

light that management data from NGCA accounts for an incoming/outgoing average of 46 / 7440 [bps], while FIXP Switch 3 is 1510 / 9.6 [bps]. Moreover, the number of unknown packets to the FIXP Control Plane decreases over time, as the FIXP Data Plane is set. The remaining traffic only encompasses the NG Hellos that are multicasted to the known peers.

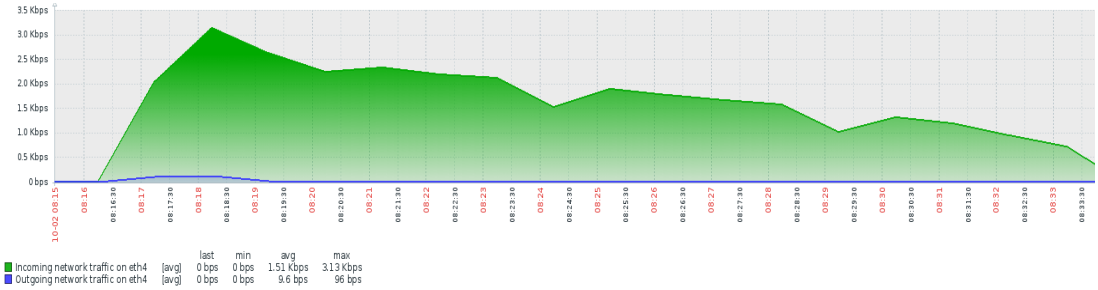

Figure 6.15: *FIXP Abstraction Layer Throughput from/to FIXP Switch 3 with photos of 780B.*

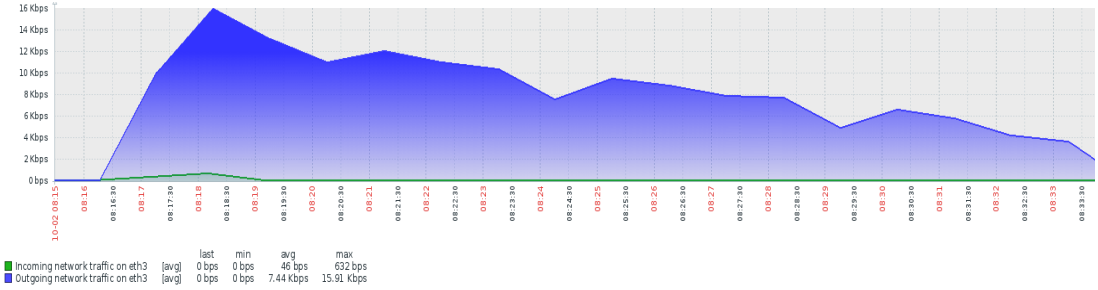

Figure 6.16: *FIXP Abstraction Layer Throughput from/to NG Controller with photos of 780B.*

6.1.4 NGCA

Finally, Figure 6.17 depicts the required time to insert a new forwarding rule at the FIXP Data Plane and receive its acknowledgment from FRHS. Essentially, the average NG Controller Table Add values do not change significantly as the underlying topology increases and require more primitives to set it. For example, it takes 555.104 ± 114.06, 469.906 ± 51.697, and 588.981 ± 50.517 [ms] to insert one rule and receive its Ack from the FIXP Data Plane with one, three, and five FIXP Switches on average. Just as a matter of curiosity, the total of rules are 3, 9, and 15, respectively.

Figure 6.18 depicts the network throughput at the FIXP Control Plane. There is a 2.91 [kbps] of average incoming traffic and 46.4 [bps] of outgoing traffic. Once again, it follows the same trend seen before. As the underlying network is set, the requesting packets decrease over time. The remaining data represents the multicasted NG Hellos.

Concerning the Reinsertion Primitive, it has not been observed once NGCA only has received NG Hellos to set the underlying FIXP Data Plane.

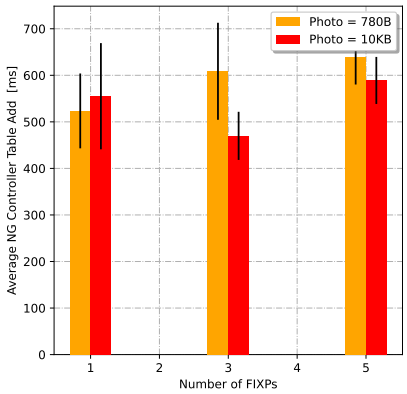

Figure 6.17: *NGCA Table Add.*

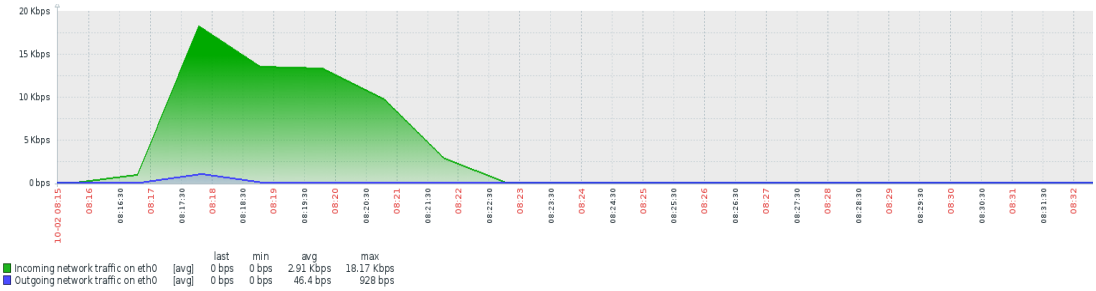

Figure 6.18: *NG Controller Throughput from/to FIXP Abstraction Layer with photos of 780B.*

Therefore, it has not stored in its queue essential data packets to reinsert. NGCA discards the received NG Hello packets because they have had already been multicasted by the FIXP Switches.

6.1.5 Host

At last, this sub-chapter presents the obtained performance related to the Host Machine. Figure 6.19-a) outlines the CPU Usage variation over the increase of FIXP Switches, while Figure 6.19-b) the Memory Usage over the same conditions. As highlighted in these figures, the required computational resources do not change significantly over the proposed topologies. Nonetheless, the bottleneck lies in the virtualization technique through VMs that is not optimal. For each VM, at least three cores of the processor and 10GB of Hard Disk were reserved from the host machine, which might restrain more substantial topologies for future works.

6.2 Functional Evaluation

There are no graphics to shorten this analysis. Even though NGCA establishes the communication between the peers in a multi-path network topology, the current design does not set the best path to connect the NG

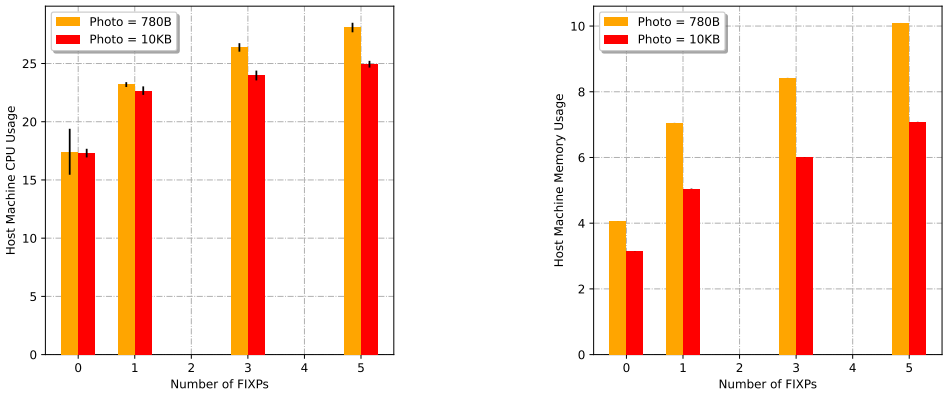

Figure 6.19: *Host Performance concerning: a) CPU Usage and b) Memory Usage.*

hosts every time. These results point that NGCA still requires more intelligence and probably a upper layer for future Internet routing algorithms with network applications to manage the controller's knowledge over time.

6.3 Concluding Remarks

This chapter has presented the obtained results for the NGCA prototype. As seen, the FIXP impact is minimal for the NG application point of view. Considering the overhead experienced by the application, the increase seems linear and in the order of 10 ms. This inserted delay is acceptable for photo exchange. Nonetheless, the current solution of virtualizing the network topology can be a bottleneck that restrains the P4 full potential, in terms of applying BMv2 and VMs.

Even though it fulfills its premise, NGCA requires further intelligence to retrieve the best path for multi-path topologies. Nevertheless, this does not burden the present work since the premise was to develop a bottom-up controller. In other words, this native solution can become flexible enough to encompass higher intelligence from a FIXP Network Application Plane, which might focus on managing the underlying FIXP SDN Controllers knowledge over time. Furthermore, NGCA can accommodate other SDN technologies. This might even lead to the management of network topologies with heterogeneous routing and SDN technologies.

Unfortunately, the complete delay profile could not be presented due to the synchronization precision between VMs. As the current solution has a clock synchronization precision of up to 1 [ms], this fact has compromised the measurements that involved the propagation delays between VMs.

Another fact that narrows the overall topology and FIXP scalability analysis is the current VirtualBox hypervisor. This hypervisor limits the total number of network interfaces connected on each Virtual Machine (VM) to eight. Moreover, it also presents some issues with the conflict of resources

in OSs that has VMware. Taking everything into account, this VM virtualization technique also narrows the full potential of FIXP to create a more substantial network topology. Although it might not strain the CPU and memory usage, it consumes a meaningful amount of hard disk storage and it is not so handy as docker containers to be automated.

Contrasting the proposed solution with the related works in Chapter 3, we can notice that NGCA can be the first native FIA SDN controller for a FIXP. Most of the covered works present an external and top-down controller to address their proposal. In other words, these controllers are specific for their application alone, while NGCA is scalable for any NG use case that exploits the Ethernet legacy link layer. Nevertheless, NGCA is not bound to Ethernet alone once PGCS can translate any other link-layer technology so this controller can manage a P4 forwarding element modeled with another protocol header. Meanwhile, this new stack must comply with the proposed design choices (4 reserved bytes in NG adaptation layer, a Data Plane that fills these bytes with meaningful data for the NGCA, and a middleware that follows the FIXP primitives).

In addition, it is challenging to present a fair comparison between NGCA and FIXP with the related works. First, they are not open-source proposals and it compromises a qualitative analysis. Second, most works are not compatible with our solution, exploiting different SDN technologies and custom software than P4. For instance, [89] does not clarify their architecture and [91, 93] apply a JAVA framework in their projects. Considering P4 proposals, Feng et al. [83] investigate the interoperation between ICN and HTTP through a custom proxy to translate these protocols, analyzing the transmission efficiency and redundant traffic. Signorello et al. [84] replicate NDN forwarding structure and validate their proposal with two native NDN applications, yet there are no qualitative results. Karrakchou et al. [85] focus on NDN architecture and evaluate their scheme with two different network topologies, supervising their latency and throughput. Gimenez et al. [86] propose an internal P4 router for RINA and investigate its throughput and packet loss over a Mininet simulation and Amazon WS VMs. Finally, Baktir et al. [88] propose a P4 forwarding element for TCP/IP, supervising the response load from the server.

In summary, the proposed NGCA for a NG-ready FIXP is appealing because it focuses on a native controller that is scalable for the NG architecture. Considering its computational overhead, it seems competitive with related works. Nevertheless, we still have to evaluate our proposal performance in more substantial environments.

Chapter 7

Final Remarks

FINALLY, this chapter concludes this master's thesis. Therefore, it outlines the main conclusion and contributions, the lessons learned, and possible future works of this work.

7.1 Conclusions and Contributions

This work directly has presented a native NovaGenesis Controller Agent (NGCA) for a programmable Future Internet eXchange Point (FIXP) that supports all NG novelties, including self-organizing scheme, fragmentation, and name-based routing. This proposal combines the trends of Future Internet Architecture (FIA) with Software Defined Networks (SDN). Moreover, it is innovative, being the first in the literature to design a native FIA SDN controller during the writing period of this dissertation.

Taking advantage of a native NG controller to manage a programmable Data Plane, this approach has proven to be efficient and competitive through an evaluation methodology that contemplates performance and functional metrics. Moreover, it has contributed to the NovaGenesis (NG) architecture development over programmable networks.

Technologies will continue to evolve continuously and will possibly present disjoint demands, which may not be solved by a single communication architecture, as we see today. This change is a debate for society, governments, and companies, once the Internet has become a public utility service. The possibility of any multi-architecture Internet must support technological growth without restricting already established services.

7.2 Lessons Learned

Through this work, we proved the feasibility of a native Control Plane for FIA, efficiently managing a programmable network dynamically.

Furthermore, we have highlighted the relevance of softwarization, virtualization, simulation, automation, and hardware programmability trends in communication networks. By proposing this experimental research that has started to solve a specific problem (i.e. developing a NG controller for a FIXP with a P4-based Data Plane), some synergy with current scientific discussions has been found. Taking into account the discussion in Chapters 1 and 2, this subject is relevant for 5G and post 5G mobile network technologies, network slicing, and multi-architecture Internet.

Finally, we also prove the relevance of the NG architecture. As much as this architecture is about ten years old, its ingredients design a flexible communication architecture project for future convergence. As seen in Chapters 2 and 3, NG has novelties that are common in other FIAs proposals and it harmoniously accommodates contemporary trends as IoT, Industry 4.0, Software Defined Radio, and Software Defined Networks.

7.3 Future Works

During the development of this work, new opportunities for future work emerged through the research, taking into account similar proposals in the literature, our efforts, and new technologies. In this way, some possible fronts for the continuity of this work are presented.

Future studies may focus on a methodology that enables the automation of more significant topologies and other demands. First, some can explore different platforms and virtualization techniques such as containers. Second, it can exploit cloud environments and testbeds for experimentation such as Future Internet Brazilian Environment for Experimentation (FIBRE). At last, it can expand the FIXP evaluation for other network parameters such as throughput, scalability, and packet loss. In such optimized opportunities, it can also investigate the NG performance with constant hyperparameters over contrasting test cases.

Taking everything into account from this dissertation, we present some NG future research. First, some future efforts can carry out a study of algorithms for Future Internet routing. Second, it can propose decentralized knowledge using technologies such as Artificial Intelligence and Blockchain for inter-domain communication. Lastly, it can consider expanding NG Control Agent to manage heterogeneous networks with disjoint and complementary Software Defined Networks technologies besides from P4.

—- That’s all folks

Bibliography

[1] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th ed. USA: Prentice Hall Press, 2010.

[2] J. Day, *Patterns in Network Architecture: A Return to Fundamentals*, 2008.

[3] J. Kurose, K. Ross, A. Marques, and W. Zucchi, *Redes de computadores e a Internet: uma abordagem top-down*. Pearson Addison Wesley, 2015.

[4] W. Ding, Z. Yan, and R. H. Deng, "A Survey on Future Internet Security Architectures," *IEEE Access*, vol. 4, pp. 4374–4393, 2016.

[5] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff, "A brief history of the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 5, p. 22–31, Oct. 2009. [Online]. Available: https://doi.org/10.1145/1629607.1629613

[6] J. Pyfer, "Sketchpad." [Online]. Available: https://www.britannica.com/technology/Sketchpad

[7] A. M. Alberti, "A Conceptual-Driven Survey on Future Internet Requirements, Technologies, and Challenges," *Journal of the Brazilian Computer Society*, vol. 19, p. 291–311, 2013. [Online]. Available: https://doi.org/10.1007/s13173-013-0101-2

[8] S. Y. Bao and H. K. Wu, "Future Internet Trends Research," in *Sensors, Measurement and Intelligent Materials II*, ser. Applied Mechanics and Materials, vol. 475. Trans Tech Publications Ltd, 3 2014, pp. 1211–1214.

[9] A. Manuel Vaz, B. Magalhaes Martins, R. Carneiro Brandao, and A. Marcos Alberti, "Internet of Information and Services: A Conceptual Architecture for Integrating Services and Contents on the Future Internet," *IEEE Latin America Transactions*, vol. 10, no. 6, pp. 2292–2300, 2012.

[10] S. Deering and R. Hinden, "Internet protocol, version 6 (ipv6) specification," Internet Requests for Comments, The Internet Society, RFC 2460, December 1998. [Online]. Available: https:

//tools.ietf.org/html/rfc2460

[11] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[12] J. Pan, S. Paul, and R. Jain, "A survey of the research on future internet architectures," *Comm. Magazine, IEEE*, vol. 49, no. 7, pp. 26 –36, 2011.

[13] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[14] T. B. da Silva, E. S. de Morais, L. F. F. d. Almeida, R. d. R. Righi, and A. M. Alberti, *Blockchain and Industry 4.0: Overview, Convergence, and Analysis*. Singapore: Springer Singapore, 2020, pp. 27–58.

[15] Y. A. Qadri, A. Nauman, Y. B. Zikria, A. V. Vasilakos, and S. W. Kim, "The future of healthcare internet of things: A survey of emerging technologies," *IEEE Communications Surveys Tutorials*, vol. 22, no. 2, pp. 1121–1167, 2020.

[16] F. Idzikowski, L. Chiaraviglio, W. Liu, and J. van de Beek, "Future Internet Architectures and Sustainability: An Overview," in *2018 IEEE International Conference on Environmental Engineering (EE)*, 2018, pp. 1–5.

[17] T. Qiu, N. Chen, K. Li, M. Atiquzzaman, and W. Zhao, "How Can Heterogeneous Internet of Things Build Our Future: A Survey," *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 2011–2027, 2018.

[18] J. Ni, X. Lin, and X. S. Shen, "Efficient and secure service-oriented authentication supporting network slicing for 5g-enabled iot," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 644–657, 2018.

[19] C. Mei, J. Liu, J. Li, L. Zhang, and M. Shao, "5g network slices embedding with sharable virtual network functions," *Journal of Communications and Networks*, vol. 22, no. 5, pp. 415–427, 2020.

[20] A. Stamou, N. Dimitriou, K. Kontovasilis, and S. Papavassiliou, "Autonomic Handover Management for Heterogeneous Networks in a Future Internet Context: A Survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3274–3297, 2019.

[21] Comitê Gestor da Internet no Brasil (CGIbr), "IX.br bate recorde histórico ao atingir 16 Tbit/s de pico de tráfego Internet." [Online]. Available: https://ix.br/noticia/releases/

ix-br-bate-recorde-historico-ao-atingir-16-tbit-s-de-pico-de-trafego-internet/

[22] K. M. Alam and A. El Saddik, "C2PS: A Digital Twin Architecture Reference Model for the Cloud-Based Cyber-Physical Systems," *IEEE Access*, vol. 5, pp. 2050–2062, 2017.

[23] P. Zanna, P. Radcliffe, and K. G. Chavez, "A Method for Comparing OpenFlow and P4," in *2019 29th International Telecommunication Networks and Applications Conference (ITNAC)*, 2019, pp. 1–3.

[24] F. Cirillo, D. Gómez, L. Diez, I. Elicegui Maestro, T. B. J. Gilbert, and R. Akhavan, "Smart city iot services creation through large-scale collaboration," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 5267–5275, 2020.

[25] G. Aceto, V. Persico, and A. Pescapé, "A survey on information and communication technologies for industry 4.0: State-of-the-art, taxonomies, perspectives, and challenges," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3467–3501, 2019.

[26] I. Yaqoob, L. U. Khan, S. M. A. Kazmi, M. Imran, N. Guizani, and C. S. Hong, "Autonomous driving cars in smart cities: Recent advances, requirements, and challenges," *IEEE Network*, vol. 34, no. 1, pp. 174–181, 2020.

[27] M. Chernyshev, Z. Baig, O. Bello, and S. Zeadally, "Internet of things (iot): Research, simulators, and testbeds," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1637–1647, 2018.

[28] Z. Yan, P. Zhang, and A. V. Vasilakos, "A survey on trust management for internet of things," *Journal of Network and Computer Applications*, vol. 42, pp. 120–134, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804514000575

[29] C. Bormann, A. P. Castellani, and Z. Shelby, "Coap: An application protocol for billions of tiny internet nodes," *IEEE Internet Computing*, vol. 16, no. 2, pp. 62–67, 2012.

[30] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-Independent Packet Processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656890

[31] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv, "Control plane of software defined networks: A survey," *Computer Communications*, vol. 67, pp. 1–10, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366415002200

[32] F. Esposito, I. Matta, and V. Ishakian, "Slice embedding solutions for distributed service architectures," *ACM Comput. Surv.*, vol. 46, no. 1, Jul. 2013. [Online]. Available: https://doi.org/10.1145/2522968.

2522974

[33] P. G. K. Patra, F. E. R. Cesen, J. S. Mejia, D. L. Feferman, L. Csikor, C. E. Rothenberg, and G. Pongracz, “Toward a Sweet Spot of Data Plane Programmability, Portability, and Performance: On the Scalability of Multi-Architecture P4 Pipelines,” *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2603–2611, 2018.

[34] A. Yazdinejad, A. Bohlooli, and K. Jamshidi, “P4 to sdnet: Automatic generation of an efficient protocol-independent packet parser on reconfigurable hardware,” in *2018 8th International Conference on Computer and Knowledge Engineering (ICCKE)*, 2018, pp. 159–164.

[35] The P4 Language Consortium, “P4 16 language specification version 1.0.0.” [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.pdf

[36] The P4 Language Consortium, “P4 contributors.” [Online]. Available: https://p4.org/contributors/

[37] B. O’Connor, Y. Tseng, M. Pudelko, C. Cascone, A. Endurthi, Y. Wang, A. Ghaffarkhah, D. Gopalpur, T. Everman, T. Madejski, J. Wanderer, and A. Vahdat, “Using p4 on fixed-pipeline and programmable stratum switches,” in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2019, pp. 1–2.

[38] The P4 Language Consortium, “P4runtime specification.” [Online]. Available: https://p4.org/p4runtime/spec/v1.0.0/P4Runtime-Spec.html

[39] The P4 Language Consortium, “BEHAVIORAL MODEL (bmv2).” [Online]. Available: https://github.com/p4lang/behavioral-model

[40] Linux Foundation, “P4-OvS - Bringing the power of P4 to OvS!” [Online]. Available: https://github.com/osinstom/P4-OvS

[41] Linux Foundation, “P4rt-OVS: Programming Protocol-Independent, Runtime Extensions for Open vSwitch using P4.” [Online]. Available: https://github.com/Orange-OpenSource/p4rt-ovs

[42] J. A. T. Gavazza, J. C. Melo, T. B. da Silva, A. M. Alberti, P. F. Rosa, F. de Oliveira Silva, F. L. Verdi, and J. A. Suruagy, “Future internet exchange point (fixp): Enabling future internet architectures interconnection,” in *Advanced Information Networking and Applications*, L. Barolli, F. Amato, F. Moscato, T. Enokido, and M. Takizawa, Eds. Cham: Springer International Publishing, 2020, pp. 703–714.

[43] J. A. T. Gavazza, “Uma proposta para a coexistência de múltiplas Arquiteturas de Internet do Futuro,” Master’s thesis, Universidade Federal de São Carlos (UFSCar), Sorocaba, SP, 2020. [Online]. Available: https://repositorio.ufscar.br/handle/ufscar/13176

[44] A. M. Alberti, *Future Network Architectures: Technological Chal-*

lenges and Trends. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 79–120. [Online]. Available: https://doi.org/10.1007/978-3-642-13247-6_5

[45] A. Alberti, M. Casaroli, R. Righi, and D. Singh, "Introducing novagenesis as a novel distributed system-based convergent information architecture," in *Nature-Inspired Networking Theory and Applications*. Boca Raton: CRC Press, 03 2018, ch. 4, pp. 88–144. [Online]. Available: https://doi.org/10.1201/9781351182089

[46] A. M. Alberti, G. D. Scarpioni, V. J. Magalhães, A. Cerqueira S., J. J. P. C. Rodrigues, and R. da Rosa Righi, "Advancing novagenesis architecture towards future internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 215–229, 2019.

[47] A. Alberti, M. Casaroli, D. Singh, and R. Righi, "Naming and name resolution in the future internet: Introducing the novagenesis approach," *Future Generation Computer Systems*, vol. 67, 09 2016.

[48] A. Alberti, M. Bontempo, J. Dos Santos, A. Sodré Jr., and R. Righi, "Novagenesis applied to information-centric, service-defined, trustable iot/wsan control plane and spectrum management," *Sensors*, vol. 18(9), September 2018.

[49] A. Alberti, J. Santos, E. Morais, R. Hilário Santos, and V. Magalhaes, "Forwarding/Routing with Dual Names: The NovaGenesis Approach," in *IX Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF)*, Campos do Jordão, Brazil, May 2018. [Online]. Available: https://sol.sbc.org.br/index.php/wpeif/article/view/2323/2287

[50] R. Garner, "Observatory - Optics," Dec. 2017. [Online]. Available: http://www.nasa.gov/content/goddard/hubble-space-telescope-optics-system

[51] "Hubble's Instruments Including Control and Support Systems (Cutaway)." [Online]. Available: http://hubblesite.org/contents/media/images/4521-Image

[52] A. M. Alberti, D. Mazzer, M. Bontempo, L. H. de Oliveira, R. da Rosa Righi, and A. Cerqueira Sodré, "Cognitive radio in the context of internet of things using a novel future internet architecture called novagenesis," *Computers & Electrical Engineering*, vol. 57, pp. 147 – 161, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S004579061630180X

[53] S. Saxena, "A guide to using raw sockets," 2015. [Online]. Available: https://www.opensourceforu.com/2015/03/a-guide-to-using-raw-sockets/

[54] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking,"

SIGCOMM Comput. Commun. Rev., vol. 44, no. 3, p. 66–73, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656887

[55] Z. Zhang, Y. Yu, H. Zhang, E. Newberry, S. Mastorakis, Y. Li, A. Afanasyev, and L. Zhang, "An Overview of Security Support in Named Data Networking," *IEEE Communications Magazine*, vol. 56, no. 11, pp. 62–68, 2018.

[56] Z. Zhang, E. Lu, Y. Guan, T. Li, X. Ma, Z. Kong, and L. Zhang, "Evolving Intelligent Devices for the Future via Named Data Networking," *XRDS*, vol. 26, no. 1, p. 36–39, Sep. 2019. [Online]. Available: https://doi.org/10.1145/3351482

[57] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A case for stateful forwarding plane," *Computer Communications*, vol. 36, no. 7, pp. 779–791, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366413000236

[58] A. Afanasyev, J. Burke, T. Refaei, L. Wang, B. Zhang, and L. Zhang, "A brief introduction to named data networking," in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, 2018, pp. 1–6.

[59] N. D. Networking, "Interest packet – ndn." [Online]. Available: https://named-data.net/doc/NDN-packet-spec/current/interest.html

[60] Z. Zhang, V. Vasavada, X. Ma, and L. Zhang, "DLedger: An IoT-Friendly Private Distributed Ledger System Based on DAG," 2019.

[61] T. Liang, J. Pan, M. A. Rahman, J. Shi, D. Pesavento, A. Afanasyev, and B. Zhang, "Enabling Named Data Networking Forwarder to Work Out-of-the-Box at Edge Networks," in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2020, pp. 1–6.

[62] S. Han and H. Woo, "NDN-Based Pub/Sub System for Scalable IoT Cloud," in *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2016, pp. 488–491.

[63] G. Grassi, D. Pesavento, G. Pau, L. Zhang, and S. Fdida, "Navigo: Interest forwarding by geolocations in vehicular Named Data Networking," in *2015 IEEE 16th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2015, pp. 1–10.

[64] M. Chowdhury, A. Gawande, and L. Wang, "Secure Information Sharing among Autonomous Vehicles in NDN," *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, 2017.

[65] G. Grassi, D. Pesavento, G. Pau, R. Vuyyuru, R. Wakikawa, and L. Zhang, "VANET via Named Data Networking," in *2014 IEEE Con-*

ference on Computer Communications Workshops (INFOCOM WKSHPS), 2014, pp. 410–415.

[66] E. Newberry and B. Zhang, "On the Power of In-Network Caching in the Hadoop Distributed File System," in *Proceedings of the 6th ACM Conference on Information-Centric Networking*, ser. ICN '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 89–99. [Online]. Available: https://doi.org/10.1145/3357150.3357392

[67] H. Lim, A. Ni, D. Kim, Y. Ko, S. Shannigrahi, and C. Papadopoulos, "NDN Construction for Big Science: Lessons Learned from Establishing a Testbed," *IEEE Network*, vol. 32, no. 6, pp. 124–136, 2018.

[68] S. Shannigrahi, C. Fan, and C. Papadopoulos, "Named Data Networking Strategies for Improving Large Scientific Data Transfers," in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2018, pp. 1–6.

[69] A. Baid and D. Raychaudhuri, "Wireless access considerations for the mobilityfirst future internet architecture," in *2012 35th IEEE Sarnoff Symposium*, 2012, pp. 1–5.

[70] D. Raychaudhuri, K. Nagaraja, and A. Venkataramani, "Mobilityfirst: A robust and trustworthy mobility-centric architecture for the future internet," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 16, no. 3, p. 2–13, Dec. 2012. [Online]. Available: https://doi.org/10.1145/2412096.2412098

[71] I. Seskar, K. Nagaraja, S. Nelson, and D. Raychaudhuri, "Mobilityfirst future internet architecture project," in *Proceedings of the 7th Asian Internet Engineering Conference*, ser. AINTEC '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 1–3. [Online]. Available: https://doi.org/10.1145/2089016.2089017

[72] S. Li, J. Chen, H. Yu, Y. Zhang, D. Raychaudhuri, R. Ravindran, H. Gao, L. Dong, G. Wang, and H. Liu, "Mf-iot: A mobilityfirst-based internet of things architecture with global reach-ability and communication diversity," in *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2016, pp. 129–140.

[73] A. Baid, T. Vu, and D. Raychaudhuri, "Comparing alternative approaches for networking of named objects in the future internet," in *2012 Proceedings IEEE INFOCOM Workshops*, 2012, pp. 298–303.

[74] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing information networking further: From psirp to pursuit," in *Broadband Communications, Networks, and Systems*, I. Tomkos, C. J. Bouras, G. Ellinas, P. Demestichas, and P. Sinha, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–13.

[75] G. Xylomenos, X. Vasilakos, C. Tsilopoulos, V. A. Siris, and G. C. Poly-

zos, “Caching and mobility support in a publish-subscribe internet architecture,” *IEEE Communications Magazine*, vol. 50, no. 7, pp. 52–58, 2012.

[76] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, “A survey of information-centric networking research,” *IEEE Communications Surveys Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

[77] J. Day, *Patterns in Network Architecture: A Return to Fundamentals (paperback): A Return to Fundamentals*. Pearson Education, 2007. [Online]. Available: https://books.google.com.br/books?id=k9sFgIM-z6UC

[78] V. Ishakian, J. Akinwumi, F. Esposito, and I. Matta, “On supporting mobility and multihoming in recursive internet architectures,” *Computer Communications*, vol. 35, no. 13, pp. 1561–1573, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366412001521

[79] E. Trouva, E. Grasa, J. Day, and S. Bunch, “Layer discovery in rina networks,” in *2012 IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2012, pp. 368–372.

[80] G. Boddapati, J. Day, I. Matta, and L. Chitkushev, “Assessing the security of a clean-slate internet architecture,” in *2012 20th IEEE International Conference on Network Protocols (ICNP)*, 2012, pp. 1–6.

[81] E. Trouva, E. Grasa, J. Day, I. Matta, L. T. Chitkushev, S. Bunch, M. Leon, P. Phelan, and X. Hesselbach-Serra, “Transport over Heterogeneous Networks Using the RINA Architecture,” in *9th Wired/Wireless Internet Communications (WWIC)*, ser. Wired/Wireless Internet Communications, X. Masip-Bruin, D. Verchere, V. Tsaoussidis, and M. Yannuzzi, Eds., vol. LNCS-6649. Vilanova i la Geltrú, Spain: Springer, Jun. 2011, pp. 297–308, part 6: Quality through Routing, Naming and Control. [Online]. Available: https://hal.inria.fr/hal-01583665

[82] J. Day, E. Trouva, E. Grasa, P. Phelan, M. P. de Leon, S. Bunch, I. Matta, L. T. Chitkushev, and L. Pouzin, “Bounding the router table size in an isp network using rina,” in *2011 International Conference on the Network of the Future*, 2011, pp. 57–61.

[83] W. Feng, X. Tan, and Y. Jin, “Implementing ICN over P4 in HTTP scenario,” in *2019 2nd International Conference on Hot Information-Centric Networking (HotICN)*, 2019, pp. 37–43.

[84] S. Signorello, R. State, J. François, and O. Festor, “NDN.p4: Programming Information-Centric data-planes,” in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 384–389.

[85] O. Karrakchou, N. Samaan, and A. Karmouch, "Endn: An enhanced ndn architecture with a p4-programmabie data plane," in *Proceedings of the 7th ACM Conference on Information-Centric Networking*, ser. ICN '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–11. [Online]. Available: https://doi.org/10.1145/3405656.3418720

[86] S. Gimenez, E. Grasa, and S. Bunch, "A Proof of Concept implementation of a RINA interior router using P4-enabled software targets," in *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, 2020, pp. 57–62.

[87] E. Grasa, M. Tarzan, L. Bergesio, B. Gastón, V. Maffione, F. Salvestrini, S. Vrijders, and D. Staessens, "Irati: Open source rina implementation for linux," *Software Impacts*, vol. 1, p. 100003, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S266596381930003X

[88] A. C. Baktir, A. Ozgovde, and C. Ersoy, "Implementing service-centric model with P4: A fully-programmable approach," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–6.

[89] C. Guimarães, J. Quevedo, R. Ferreira, D. Corujo, and R. L. Aguiar, "Exploring interoperability assessment for future internet architectures roll out," *Journal of Network and Computer Applications*, vol. 136, pp. 38–56, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804519301201

[90] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifier (uri): Generic syntax," Internet Requests for Comments, IETF, RFC 3986, 1 2005. [Online]. Available: https://tools.ietf.org/html/rfc3986

[91] M. Jahanian, J. Chen, and K. Ramakrishnan, *Managing the Evolution to Future Internet Architectures and Seamless Interoperation*, 2020, pp. 1–11.

[92] "Open-Access Research Testbed for Next-Generation Wireless Networks (ORBIT)." [Online]. Available: https://www.orbit-lab.org/

[93] M. Jahanian, J. Chen, and K. Ramakrishnan, *Formal Verification of Interoperability Between Future Network Architectures Using Alloy*, 05 2020, pp. 44–60.

[94] A. M. Alberti, K. Costa, and T. T. Rezende, "Virtualização em redes terrestre-satélite 5g," in *Anais do I Workshop de Teoria, Tecnologias e Aplicações de Slicing para Infraestruturas Softwarizadas*. Porto Alegre, RS, Brasil: SBC, 2019, pp. 69–82. [Online]. Available: https://sol.sbc.org.br/index.php/wslice/article/view/7723

[95] A. M. Alberti, V. H. d. O. Fernandes, M. A. F. Casaroli, L. H. d. Oliveira, F. M. P. Júnior, and D. Singh, "A novagenesis proxy/gateway/controller for openflow software defined networks," in *10th International Conference on Network and Service Management (CNSM) and Workshop*, 2014, pp. 394–399.

[96] M. C. Silva, P. M. R. Pereira, and P. H. de Paula Viana Dias, "Projeto e Desenvolvimento de um Comutador para a NovaGenesis usando o Protocolo KeyFlow," 2019, bachelor's dissertation at INATEL (Instituto Nacional de Telecomunicacoes), Santa Rita do Sapucai, Brazil.

[97] M. C. Silva, A. C. Dominiano, and A. M. Alberti, "Encaminhador de pacotes utilizando dois comutadores com protocolo keyflow," in *Anais do Congresso de Iniciacao Cientifica do INATEL - INCITEL 2019*. Santa Rita do Sapucai, MG, Brasil: INATEL, 2019, pp. 37–40.

[98] "Internet protocol version 4 (ipv4) parameters." [Online]. Available: https://www.iana.org/assignments/ip-parameters/ip-parameters.xhtml#ip-parameters-1

[99] "Internet protocol, version 6 (ipv6) specification." [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2460#page-6

[100] "The p4 language specification version 1.0.4," May 2017. [Online]. Available: https://p4.org/p4-spec/p4-14/v1.0.4/tex/p4.pdf

[101] "Main page," 2015. [Online]. Available: https://rapidjson.org/

[102] Wireshark, "Chapter 1. introduction." [Online]. Available: https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html#ChIntroWhatIs

[103] Zabbix SIA, "Zabbix documentation 4.0." [Online]. Available: https://www.zabbix.com/documentation/4.0/manual/introduction/about

[104] NumPy community, "Scapy." [Online]. Available: https://scapy.net

[105] Pandas development team, "Package overview." [Online]. Available: https://pandas.pydata.org/docs/getting_started/overview.html

[106] Philippe Biondi and the Scapy community, "Numpy." [Online]. Available: https://numpy.org

[107] John Hunter and Darren Dale and Eric Firing and Michael Droettboom and the Matplotlib development team, "Matplotlib: Visualization with python." [Online]. Available: https://matplotlib.org/stable/index.html

[108] A. M. Alberti, E. C. do Rosário, G. Cassiano, J. R. dos Santos, V. H. D'Ávila, and J. R. Carneiro, "Performance evaluation of novagenesis information-centric network," in *2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech)*, 2017, pp. 1–6.

[109] A. Leon-Garcia, *Probability, Statistics, and Random Processes for Electrical Engineering*. Pearson/Prentice Hall, 2008. [Online]. Available: https://books.google.com.br/books?id=GUJosCkbBywC

[110] "P4 broadcasting rules issue." [Online]. Available: https://github.com/p4lang/tutorials/issues/398

[111] "P4 switch performance." [Online]. Available: https://github.com/p4lang/behavioral-model/issues/301

[112] "bmv2 low throughtput." [Online]. Available: https://github.com/p4lang/behavioral-model/issues/537

Appendix I

NG Flow

Figure I.1 outlines a simplified flowchart between NovaGenesis (NG) processes that enables the Proxy/Gateway Controller Service (PGCS), setting this service in a perennial execution of receiving and sending packets from/through communicating sockets or shared memory. Meanwhile, this simplified version only addresses the main C++ source codes that impact more in this scenario. This means that there are several more instances, classes, and interactions between other NG processes that build and collaborate in a fully communicating PGCS, yet they are omitted in this document. In other words, the focus here lies only on providing the crucial points to understand how a NG communication develops for a generic scenario.

Input Data:

First of all, it is important to highlight the main NG command line to initialize its operation and explain its 10 parameters.

```
sudo PGCS_executable_path Path_to_store_logs Origin_Port
PGCS_Communicating_Role NG_Option Communication_Stack
PGCS_Communicating_Role Communicating_Interface Peer_Address MTU
```

a. **sudo**: It usually concedes root privileges to the application. As PGCS exploits raw sockets to communicate, it is a good practice to initiate NG with *sudo*.

b. **PGCS_executable_path**: It points to the **path** where *PGCS* application directory is located in the host machine.

c. **Path_to_store_logs**: It points to the **path** where PGCS logs must be stored in the host machine.

d. **Origin_Port**: It specifies the communication port.

e. **PGCS_Communicating_Role**: It specifies the **role** in which the PGCS acts in the application communication in terms of domains. As a matter of fact, this field can be either “intra-domain,” which con-

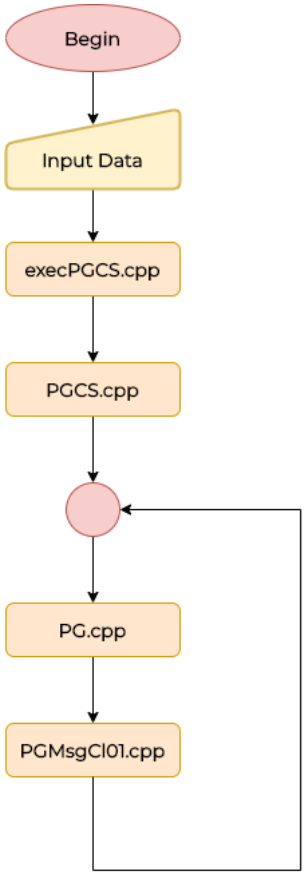

Figure I.1: *Simplified Flowchart for PGCS main processes.*

siders that the communication happens inside the same domain, or "inter-domain," which specifies the communication between distinct domains.

f. **NG_Option**: This field specifies a functional NG scenario to deploy a given application. For example, this parameter can receive the value of "-p," which points to direct communication with determined peers, or "-pc," which indicates a direct communication with a NG defined core service.

g. **Communication_Stack**: It specifies the communication protocol **Stack** in which PGCS must adapt NG messages to communicate with other hosts. Besides of "Ethernet," this field can receive ...

h. **PGCS_Communicating_Role**: It specifies the **role**.

i. **Communicating_Interface**: It specifies the communicating **interface** that PGCS manages to communicate with other peers. This can be either a physical network interface on the host machine, i.e. physical hardware that grants access to an external communication through cabled or wireless means, or an emulated network interface on a Virtual Machine (VM).

j. **Peer_Address**: It specifies the desired peer address to communicate. For example, this parameter can receive either Layer-2, i.e. Media Access Control (MAC) addresses, or Layer-3, i.e. Internet Protocol (IP), values from IEEE 802.3 stack. In special cases, this field can be omitted.

k. Maximum Transmission Unit (MTU): It specifies the maximum transmission unit for each packet sent or received to/from the network. This parameter is crucial to the NG Adaptation Layer, once it fragment or remount packets of a given NG based on this value. This is an integer value that conveys the maximum protocol communicating packet size or the desired packet length.

For example, someone can initialize NG with the following command line:

```
sudo ./PGCS ./ 0 Intra_Domain -p Ethernet Intra_Domain eth0 AB:BC:CD:EF:GH:HI 1500
```

This command line sets a direct Ethernet intra-domain communication with a peer that has the *AB:BC:CD:EF:GH:HI* Mac Address, transmitting packets with up to 1500 bytes of MTU.

After executing this command line, the PGCS begins to initiate a desired scenario.

execPGCS.cpp:

execPGCS.cpp is a C++ source code that acts as a switcher, retrieving the Input Data based on the command line that starts the NG up. This manipulates the given arguments to enable the desired application scenario, making this data available to every required NG process.

In the given example, **execPGCS.cpp** selects the scenario where NG must communicate directly with the *AB:BC:CD:EF:GH:HI* peer within an intra-domain. Therefore, it must prepare other processes to create mechanisms that enable this case. In other words, PGCS must exchange NG messages with an external PGCS, format the data per the Ethernet protocol standard, and ensure that the fragments of the packet take up to 1500 bytes in an Ethernet payload.

PGCS.cpp:

PGCS.cpp is a C++ source code that focuses on the high-level tasks of PGCS process. In its scope, it manages the required computational threads to deal with the Server Raw Socket (SRS) that reads incoming packets.

Moreover, it allocates blocks that accumulate Proxy/Gateway or Core functions for network tasks.

PG.cpp:

PG.cpp is a source code that manages the required sockets to establish a communication, deals with processing and preparing NG messages to several protocol stacks, and exchanges data between other NG processes through the shared memory.

At a first moment, *PG.cpp* initializes the required sockets to accomplish the selected communication in NG startup. For instance, it creates a pair of Client Raw Socket (CRS) and a SRS to deal with incoming packets from the Ethernet stack. At this configuration, the CRS binds to a communication port and takes up the communication, sending the data to a peer SRS. On the other hand, SRS commences to bind and only listen to a given port.

Besides that, it deals with the incoming packets received from the thread that reads the NG server raw socket. As this kind of socket acquires the packets directly from the communication interface, incorporating mechanisms to manipulate its control data in every packet to determine whether the packet belongs to a fragmented NG message, its message Identifier (ID), and which fragment it is. For instance, the receiving function discounts the Ethernet header, extracts the NG control information for its Adaptation Layer, mounts the NG message by each NG payload received, and feeds the intact NG message to the shared memory for other processes.

Whenever a NG message is ready to be sent to a peer, PGCS receives this data from the shared memory or the output queue and handles it to send through a proper communication socket. For example, *PG.cpp* must ensure that the NG message fits within a packet before effectively sending this packet through a CRS in an Ethernet frame protocol. Thus, it must fragment this message if necessary, insert the sequence control data in the adaptation field, and generate a message ID for each distinct NG message. After this, the NG packet is ready to be sent through the CRS.

PGMsgCl01.cpp:

PGMsgCl01.cpp is a C++ source code that deals with incoming NG messages derived from received NG command lines. These data can come from other NG processes from the shared memory or the output queue.

In its scope, it determines it the received NG command line is valid, if it targets a intra- or inter- process of its hosts/domain, and how this information shall be routed. At this moment, NG just considered software routing mechanisms.

For example...

However, Future Internet Exchange Point (FIXP) requires a new way to design Hardware (HW) enabled routing structure to configure a Programming Protocol-Independent Packet Processors (P4) modeled switch.

Appendix II

Setting the VirtualBox Environment

This appendix serves as a guide to prepare the virtual machines for FIXP operation in the VirtualBox virtualized environment with NovaGenesis architecture.

II.1 NovaGenesis Hosts

To use NovaGenesis applications, you must obtain an image of the Ubuntu 16.04 operating system, either server or desktop. For this work, we have opted for the server distribution as it is compacter. The next steps guide this setting:

a. Obtain Linux Ubuntu 16.04 Server;

b. Import the O.S. into VirtualBox and perform the necessary settings to install it;

c. After completing the Linux Ubuntu 16.04 Server installation, start the virtual machine and install all the necessary NovaGenesis architecture dependencies;

d. To use FIXP as presented, Linux Ubuntu must be prepared to accept interface names other than the default. Therefore, modify the ***GRUB_CMDLINE_LINUX*** line in the ***/etc/default/grub*** file as follows:

```
GRUB_CMDLINE_LINUX="net.ifnames=0 biosdevname=0"
```

e. Shut the Virtual Machine down.

II.2 FIXP Switch

Regarding the FIXP Switches, Ubuntu Linux version 16.04 Server distribution is recommended. After obtaining such an image, perform the following steps:

a. Import the O.S. into VirtualBox and perform the necessary settings to install it;

b. For these Virtual Machines, it is required to prepare it with the P4 language, the P4 compiler, the Behavioral Model version 2, and all its dependencies. To summarize these steps, see the tutorial available on the website:

```
https://p4.org/p4/getting-started-with-p4.html
```

c. Install Python 3.8.2 and Scapy 2.2.0 dependencies;

d. To use FIXP as presented, Linux Ubuntu must be prepared to accept interface names other than the default. Therefore, modify the ***GRUB_CMDLINE_LINUX*** line in the ***/etc/default/grub*** file as follows:

```
GRUB_CMDLINE_LINUX="net.ifnames=0 biosdevname=0"
```

e. Shut the Virtual Machine down.

II.3 FIXP Abstraction Layer

Regarding the FIXP Abstraction Layer, Ubuntu Linux version 16.04 Server distribution is recommended. After obtaining such an image, perform the following steps:

a. Import the O.S. into VirtualBox and perform the necessary settings to install it;

b. Install Python 3.8.2 and Scapy 2.2.0 dependencies;

c. To use FIXP as presented, Linux Ubuntu must be prepared to accept interface names other than the default. Therefore, modify the ***GRUB_CMDLINE_LINUX*** line in the ***/etc/default/grub*** file as follows:

```
GRUB_CMDLINE_LINUX="net.ifnames=0 biosdevname=0"
```

d. Shut the Virtual Machine down.

II.4 NovaGenesis Control Agent

Finally, Ubuntu Linux version 16.04 Server distribution is recommended for the NovaGenesis Control Agent. After obtaining such an image, perform the following steps:

a. Obtain Linux Ubuntu 16.04 Server;

b. Import the O.S. into VirtualBox and perform the necessary settings to install it;

c. After completing the Linux Ubuntu 16.04 Server installation, start the virtual machine and install all the necessary NovaGenesis architecture dependencies;

d. To use FIXP as presented, Linux Ubuntu must be prepared to accept interface names other than the default. Therefore, modify the ***GRUB_CMDLINE_LINUX*** line in the ***/etc/default/grub*** file as follows:

```
GRUB_CMDLINE_LINUX="net.ifnames=0 biosdevname=0"
```

e. Shut the Virtual Machine down.

II.5 Virtual Machines Interconnection and Network Setting

The VirtualBox's Virtual machines must be interconnected through virtual networks. To explain this step, Figure II.1 presents a standard topology that contains 1 NovaGenesis Control Agent, 1 FIXP Abstraction Layer, 1 FIXP Switch, and 2 NovaGenesis hosts. Notice the Network Interface Controllers (NICs) shown in the drawing. They are required for the configuration of the VirtualBox hypervisor.

Each network interface is in VirtualBox's promiscuous mode. In the host machine, run the following commands in the terminal to set each Virtual Machine for the FIXP environement:

Internal Networks:
vboxmamage modifyvm <vm> -nic<1-8> intnet_x
vboxmanage modifyvm <vm> –nicpromisc<1-8> allow_all
Bridge Adapters:
vboxmamage modifyvm <vm> -nic<1-8> bridged
vboxmamage modifyvm <vm> –bridgeadapter<1-N> <bridge name>

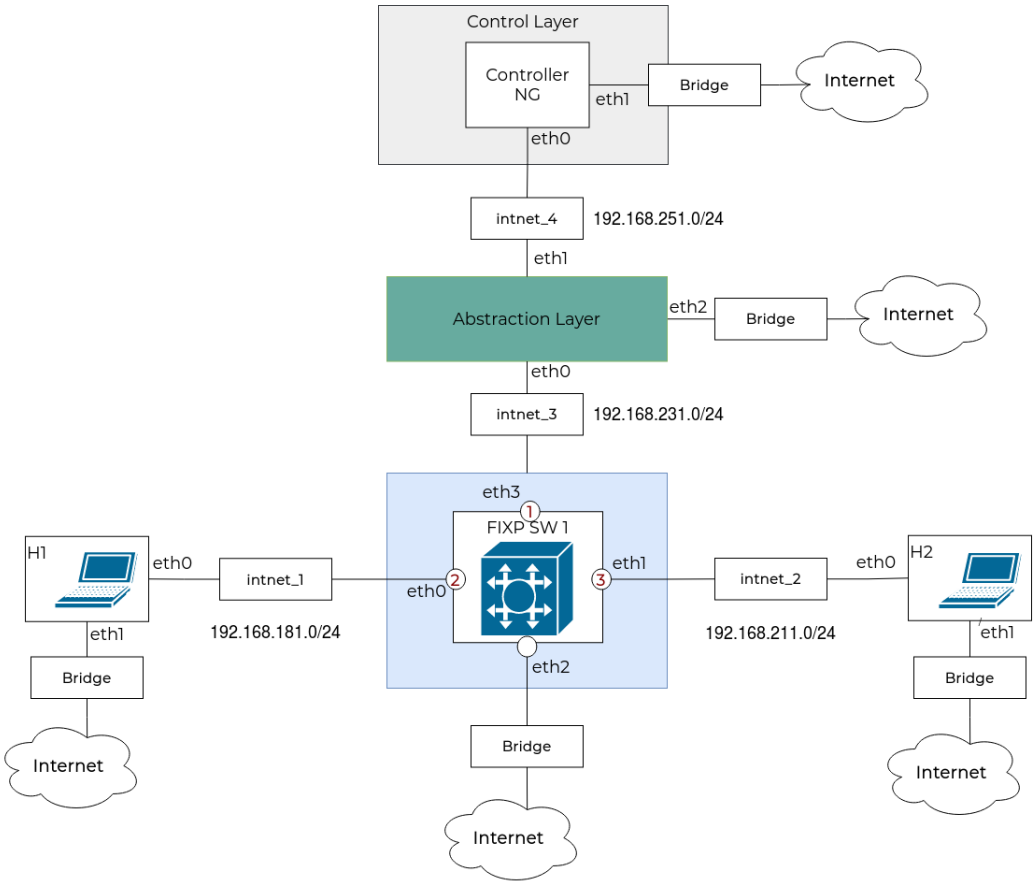

Figure II.1: *Standard Network Topology.*

Appendix III

Network Topology

III.1 Network Topology for 5 FIXP SWes

This section exposes the VirtualBox virtual networks. Table III.1 describes the virtual networks, while Figure III.1 illustrates these connections between VMs.

Table III.1: *FIXP Virtual Network Topology.*

Index	Internal Network	Network Address	Description
1	intnet_4	192.168.191.0/24	NG Source to FIXP SW 1.
2	intnet_1	192.168.181.0/24	NG Cache to FIXP 3.
3	intnet_13	192.168.151.0/24	NG Repo to FIXP 5.
4	intnet_6	192.168.211.0/24	FIXP 1 to FIXP 2.
5	intnet_11	192.168.121.0/24	FIXP 2 to FIXP 3.
6	intnet_12	192.168.131.0/24	FIXP 3 to FIXP 4.
7	intnet_7	192.168.221.0/24	FIXP 4 to FIXP 5.
8	intnet_3	192.168.231.0/24	FIXP 1 to Interface.
9	intnet_10	192.168.241.0/24	FIXP 2 to Interface.
10	intnet_8	192.168.111.0/24	FIXP 3 to Interface.
11	intnet_2	192.168.141.0/24	FIXP 4 to Interface.
12	intnet_5	192.168.201.0/24	FIXP 5 to Interface.
13	intnet_9	192.168.251.0/24	Interface to NG Controller.

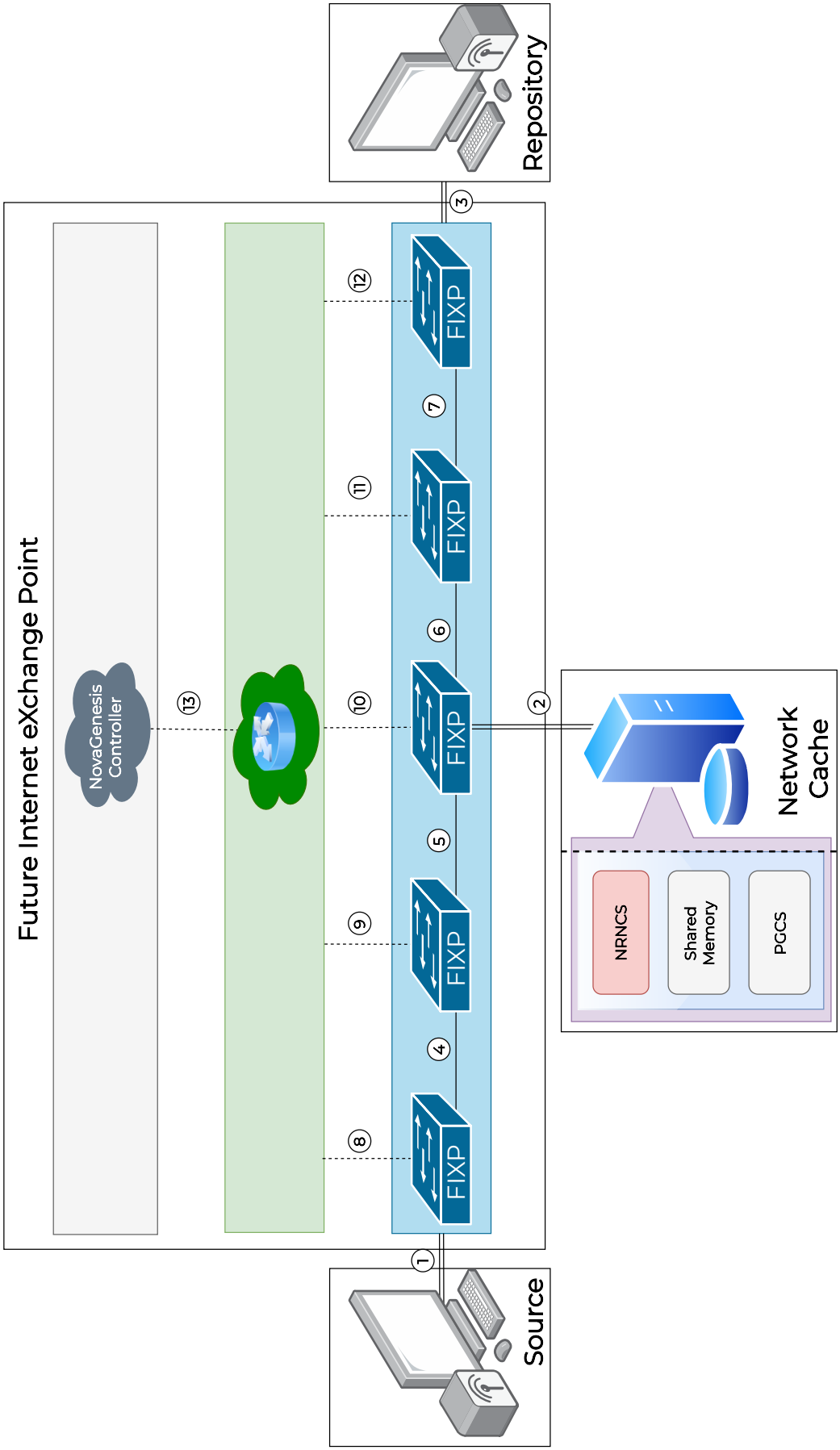

Figure III.1.: *FIXP Virtual Network Topology*