09-09-2025

Programmable Networks: Current State and Trends

Grant Agreement No.:	101194278
Work Package:	WP6
Task Item:	T1
Nature of Document:	White Paper
Dissemination Level:	PU
Lead Partner:	FAU/DFN
Document ID:	GN5-2-25-111DCD
Authors:	Asier Atutxa (EHU/RedIRIS); David Franco (EHU/RedIRIS); Eduardo Jacob (EHU/RedIRIS); Frederic Loui (RENATER); Ivana Golub (PCSS); Susanne Naegele-Jackson (FAU/DFN); Pavle Vuletić (UoB/AMRES); Simon Leinen (Switch)

Abstract

The document analyses the current status of network programmability and programmable networks and suggests possible future work in this area within the GÉANT project.

Co-funded by the European Union

© GÉANT Association on behalf of the GN5-2 project. The research leading to these results has received funding from the European Union's Horizon Europe research and innovation programme under Grant Agreement No. 101194278 (GN5-2).
Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. The European Union cannot be held responsible for them.

Contents

Figures

Executive Summary

This white paper provides a comprehensive overview of the current state of network programmability and programmable networks. It explores their historical evolution, technological background, current implementations and possible future directions and opportunities for research and experimentation, particularly within the GÉANT project.

Network programmability has evolved significantly during the last decade, offering greater flexibility, efficiency, and customisation in packet processing. This evolution is traced in this document from the early days up to the development of distributed control plane and control plane programmability, data plane programmability, and data plane programmable models and languages.

The document also covers some ongoing work on these technologies being carried out in research and education institutions and National Research and Education Networks (NRENs). Examples include the use of white boxes and P4-programmable hardware. Special emphasis is given to the Router for Academia, Research, and Education (RARE) open-source operating system and the Global Platform for Labs (GP4L) – both developed over a few iterations of the GÉANT project. This white paper also underscores the importance of continued research and experimentation in network programmability to adapt to evolving technological trends.

1 Introduction

Network engineering has evolved over the last few decades, which have seen the development of several technologies and trends that have changed how communications are understood in our society. Each of these innovations has pushed towards better network performance and efficiency (lower delay and energy consumption, higher reliability and available bandwidth, etc.), adapting communications to the characteristics of different scenarios and use cases.

In this context, one of the most studied concepts is network programmability, which encompasses the data and control plane and aims to provide complete customisation of packet processing throughout the network. Programmable networks have several advantages over traditional networks, such as greater flexibility and the capacity to create custom pipelines, process packets with specific requirements, or implement protocols that might not otherwise be provided by vendors.

In view of the several benefits offered by programmable networks, academia, research and industry have focused efforts on the active development of network devices and software. Companies such as Barefoot, Intel, and Netronome have developed Application-Specific Integrated Circuits (ASICs) and Network Interface Cards (NICs) to be implemented in switches and Network Processing Units (NPUs). These, together with the associated proprietary software, have contributed to the expansion of programmable networks. On the other hand, several entities have been developing open-source alternatives, so that general-purpose equipment can be used to deploy programmable networks.

In this context, the GÉANT project Network Development Work Package (WP6) Technology Task (Task 1) has developed an open-source multifunctional routing software platform called RARE (Router for Academia, Research and Education). RARE has been designed to provide a cost-effective, high-performance Network Operating System (NOS) that has the particularity of combining one control plane with multiple data planes. RARE can be deployed on several targets, including various P4-programmable hardware and software switches as well as DPDK- and XDP-based targets. The flagship product of this project combines RARE and the Tofino chipset, merging the flexibility of open-source routing software with the terabits per second (Tbps) forwarding capability of the Tofino chip family.

Due to recent changes brought in by the main network programming technology promoters and hardware manufacturers, such as Intel stopping the development of Tofino, the current roadmap for RARE and other similar alternatives and efforts remains uncertain. The objective of this white paper is to examine the state of the art and the current trends and initiatives in the development of network programmability within this context.

Section 2 of the document provides a background and an overview of existing network programmability solutions, including control and data plane programmability, network programmability devices, targets, models and languages. Section 3 provides an insight into the adoption of network programmability examples and use cases. Examples from NRENs are provided in section 4. Special focus is given to RARE and the GP4L and their use cases (section 5). Finally, the white paper draws some main conclusions and highlights possible steps for the future of network programmability and RARE (section 6).

2 Network Programmability – Background and Solutions

Over the past decade, network programmability has been extensively studied by both researchers [1] and manufacturers. Several initiatives have emerged in academia and industry that have advanced the technology to its current state in networking and communications [2] [3] [4].

The evolution of network programmability began with centralisation of control in network devices in the control and data plane. This in turn enabled programming of the control and data plane of network devices, and soon resulted in a proliferation of hardware-, Linux- and software-based solutions, programmability models and languages, all of which are described in more detail in the rest of this chapter.

2.1 First Steps in Network Programmability: Towards Control Centralisation

Communication networks were traditionally based on commercial, off-the-shelf network devices, which were designed to provide specific services that are aligned with the requirements of user applications. These devices were customised by manufacturers to operate at high speeds and to accommodate a range of traffic types. Such traditional devices relied on distributed routing protocols that involved cooperation between network nodes to obtain a unified network view and apply the required routing policies. This cooperation required the implementation of sophisticated routing algorithms. This increased the convergence time and complexity for identifying an optimal routing. This is of particular importance in networks that are subject to constant changes.

Moreover, although these devices were able to work at high speeds and support typical network traffic, in some cases they were not flexible enough to be adapted to specific scenarios. Users relied on software updates for the implementation of new functions, which – as they used proprietary and closed software – was subject to limitations due to the vendor interests or the development cycle of networking equipment manufacturers, among other reasons. Also, being proprietary, they were usually not fully open or could not be programmed by external contributors. Additionally, network researchers could not implement newly proposed protocols due to the lack of evaluation architectures or devices in such a vendor-locked environment and therefore could not experiment with novel networking mechanisms.

Besides, depending on their priorities and roadmaps, networking equipment manufacturers can require long periods to design, implement, test, polish and commercially deploy new features – unless these happen to be their primary focus and interest, in which case development may be faster. In some cases, vendor development dynamics can also depend on the development cycle of chip vendors.

Subsequently, during the 2000s, the concept of separation between the control and data planes began to gain traction, driven by the objective of simplifying networking platforms. These new devices were designed with a focus on hardware packet forwarding, rather than software forwarding. The control plane could subsequently be centralised and offloaded to computational hardware located remotely. This is the origin of Software-Defined Networking (SDN), which was formalised in 2008 with the definition of OpenFlow. OpenFlow was introduced as the standard protocol for centralising control and deploying networks formed by devices that simply forward packets according to a set of flow-based match-action rules. With OpenFlow, all the routing decisions were moved to the centralised controller, providing interfaces for programming custom control-plane applications that improved the flexibility of network solutions.

2.2 Control Plane Programmability

Traditionally, the control and data planes have been tightly coupled in communication networks, meaning that network devices, such as routers and switches, manage both the control and forwarding functions internally. This traditional approach, though reliable, limits the flexibility and adaptability of network infrastructure, making it difficult to scale, innovate, or manage efficiently.

With the rise of cloud computing and mobile applications, and the increasing demand for dynamic and scalable network environments, traditional network infrastructure faces limitations in meeting modern requirements. Control plane programmability emerged as a solution to decouple the control plane from the data plane, enabling centralised and programmable management of network resources.

The main benefits of control plane programmability are:

- **Centralised control**: routing and forwarding decisions are made by a centralised entity that has a unique and global view of the network's topology and status. This prevents loops and other problems associated with distributed network control. Operators can manage the network as a whole rather than dealing with individual devices. They manage large, complex networks from a central point, reducing complexity and improving response times. This simplifies troubleshooting, traffic engineering, and security enforcement.
- **Flexibility**: administrators can easily modify network policies, routing paths, and overall network behaviour in real time.
- **Scalability**: as networks grow in complexity, programmable control planes can simplify management and reduce the risk of errors during configuration. Network configurations can be applied across hundreds or thousands of devices without manual intervention. Instead of manually configuring each network device, the control plane pushes configurations and policies across the network.
- **Automation**: the northbound interface is defined to program the control plane. Automation tools and scripts can be used to perform routine tasks like network provisioning, traffic engineering, or fault management. Traffic conditions, security threats, or service requests can trigger real-time adjustments. This reduces operational costs and human error.
- **Security**: with a global view of the network, more consistent and granular security policies can be enforced.
- **Innovation and experimentation**: new services, protocols, or network optimisations can be tested and deployed with minimal impact on the existing infrastructure.

The aforementioned characteristics serve as key motivators for the growing interest in control plane programmability within modern networks. Consequently, the following subsections will delineate the pivotal aspects of control plane programmability, encompassing the overall communication and management framework. This includes discussion of SDN controllers, southbound interfaces, and the OpenFlow protocol.

2.2.1 SDN Controllers

Software-Defined Networking (SDN) is one of the most important concepts in control plane programmability. SDN facilitates the separation of the control plane from the data plane, giving rise to a more flexible and centralised network management model.

The SDN controller, which resides in the control plane, is the centralised entity responsible for managing the control plane across the network. It has a global view of the network topology and can control the behaviour of network devices, such as switches and routers, through programmable interfaces. It oversees making routing decisions. It essentially decides how data packets should be forwarded based on network policies and conditions.

The main functions of an SDN controller include:

- **Centralised view**: the SDN controller maintains a centralised view of the network's topology, current state, and traffic conditions, allowing it to make informed routing decisions.
- **Policy Enforcement**: network policies related to security, Quality of Service (QoS), and traffic engineering can be centrally enforced through the controller.
- **Resource Management**: the controller optimises the use of network resources by allocating bandwidth, rerouting traffic in case of congestion or failure, and adjusting network paths based on changing conditions.
- **Programmability**: the SDN controller enables network operators and applications to interact with the network infrastructure through Application Programming Interfaces (APIs). The northbound interface allows network operators to define applications that control the behaviour of the network. The southbound interface provides a mechanism to communicate with data plane devices and configure them. In addition, east/west-bound interfaces are defined to synchronise with other SDN controllers.

The most popular SDN controllers include:

- **OpenDaylight** [5]: An open-source SDN controller (Linux Foundation) that provides a modular platform that supports a variety of southbound protocols, including OpenFlow, NETCONF, and BGP. The latest version (Scandium-SR1) was released in November 2024.
- **ONOS (Open Network Operating System)** [6]: ONOS is designed with a focus on scalability and reliability. It supports large-scale SDN deployments, particularly in service provider networks. It is oriented to production networks. The latest ONOS LTS version (X-Wing 2.7) was released in July 2021 and it has been three years since ONOS received its last update on GitHub's official repository. TeraFlowSDN appears to be the replacement for ONOS.
- **Ryu** [7]: an open-source SDN framework that provides a set of APIs to manage switches using protocols like OpenFlow. It provides great flexibility and compatibility thanks to the Python-based programming interface, which makes it suitable for research and experimentation. According to its official repository, it is not currently maintained. There is an alternative, os-ken [8], which provides the Ryu library tailored for OpenStack.
- **Cisco ACI (Application Centric Infrastructure)** [9]: This commercial SDN controller is designed to provide automated management of both physical and virtual network environments. At the time of writing, it is still maintained by Cisco, which provides subscription-based access to different features.
- **TeraFlowSDN** (TFS) [10] is an open-source, micro-service-based and carrier-grade SDN controller capable of integrating with current NFV and MEC frameworks. It was developed in the context of an H2020 project, and there is now an ETSI Software Development Group (SDG) dedicated to TFS.

2.2.2 Southbound Interfaces

In traditional networks, control and data planes are tightly integrated within devices, but southbound interfaces are key in SDN architectures for decoupling these planes and enabling centralised control. Southbound interfaces serve as the communication channel between the SDN controller and the data plane devices, such as routers, switches or firewalls. These interfaces allow the controller to populate the forwarding tables of the data plane devices.

Southbound interfaces define how an SDN controller communicates with data plane devices. They allow the controller to query the network topology, gather statistics, configure devices, and enforce policies.

Southbound interfaces enable the SDN controller to dynamically modify how devices forward traffic, thereby enabling real-time adjustments to network conditions. By supporting multiple southbound protocols, SDN controllers can interact with various types of devices from different vendors, reducing vendor lock-in and improving interoperability. Southbound interfaces also allow for granular control over packet handling, ensuring precise network traffic management and optimised performance.

The most common southbound interfaces are:

- **OpenFlow** [11]: This is the most well-known and widely used southbound protocol in SDN networks. It allows the controller to interact directly with the forwarding plane of network devices, specifying how packets should be handled.
- **NETCONF (Network Configuration Protocol)** [12]: This is primarily used for configuration management and monitoring. It uses XML-based data to configure network devices.
- **BGP (Border Gateway Protocol)** [13]: BGP can also be used as a southbound protocol, enabling controllers to make decisions on inter-domain routing. This means the SDN controller acts as a BGP speaker and communicates with the network devices using BGP messages.
- **OVSDB (Open vSwitch Database Management Protocol)** [14]: OVSDB is used to manage and control Open vSwitch instances, a popular virtual switch for network virtualisation.

2.2.3 OpenFlow Protocol

As previously stated, the OpenFlow protocol is the de facto standard for the implementation of the SDN southbound interface. It is a foundational element of SDN architectures and is widely considered the first standard interface for SDN. It facilitates communication between the SDN controller and the forwarding devices in the data plane.

OpenFlow enables the SDN controller to program network switches by providing a set of forwarding or flow rules that define how traffic should be handled. The controller installs these flow rules in the switch's flow table, allowing the switch to forward packets accordingly without needing further intervention from the controller unless conditions change. Each flow rule in the OpenFlow protocol consists of three main components:

- **Match Fields**: Each flow rule represents a group of packets, also known as flow, which is identified by the match fields. In other words, it is the criteria for identifying traffic flows. The matching fields are based on packet headers such as source/destination IP addresses, port numbers, protocol type, etc.
- **Actions**: Instructions for the data plane device on how to handle the packets that match the criteria, such as dropping, forwarding them to a specific port, or modifying packet headers.
- **Counters**: Statistics on how many packets or bytes match a flow rule, allowing the controller to monitor network conditions.

One of the main benefits of OpenFlow is that it abstracts control logic from the physical devices, which simplifies the overall network architecture and the procedures to manage and control it. OpenFlow also provides a standard interface that applies to different manufacturers, thus reducing vendor lock-in.

OpenFlow is commonly used in data centres to optimise traffic flows, balance loads, and enable virtualisation. It can also be employed in Wide Area Networks (WANs) to ensure efficient traffic routing and management across multiple sites. OpenFlow is also used for enhancing the security of the network by implementing centralised security policies.

The OpenFlow protocol has evolved through various versions, with each iteration introducing more complex features such as support for multiple flow tables, group tables for multipath routing, and enhanced match fields for more granular packet inspection. The latest version of OpenFlow is 1.5.1. Even though it was released in 2015, OpenFlow is still seen as a pivotal protocol in the technological background of network programmability, as it reflects several key aspects in the development of this technology. After this, the release of new versions was stopped due to several factors:

- **Maturity of the protocol**: OpenFlow had a rapid evolution in its early years, as it was attempting to gain wider adoption in SDN environments. By the time it reached version 1.5.1, the protocol had become relatively stable, and the focus shifted from making rapid changes to refining and integrating OpenFlow into practical, real-world deployments.
- **Shift to other SDN solutions**: While OpenFlow was initially central to the SDN movement, other protocols and technologies (such as P4, gRPC-based models, and proprietary APIs) have emerged, leading some organisations to move away from OpenFlow as the primary means of controlling networks.
- **Network control ecosystem**: Another reason that no new OpenFlow versions are being released is that the network control ecosystem has itself evolved. Much of the development in SDN has moved to other layers or frameworks, such as NFV platforms (e.g., OpenStack and Kubernetes) or intent-based networking, which do not necessarily rely on OpenFlow.
- **Industry adoption**: The adoption of OpenFlow in production networks has been limited. Many hardware vendors and software solutions shifted to proprietary solutions for better compatibility, which also contributed to reducing the development of OpenFlow.

Therefore, OpenFlow 1.5.1 remains the latest stable version, the industry has shifted focus to newer solutions, and the role of OpenFlow is now more that of a foundational component rather than a leading-edge protocol.

In conclusion, control plane programmability represents a shift towards more dynamic, scalable, and flexible network management. SDN controllers, southbound interfaces, and protocols like OpenFlow are critical components in achieving this vision, allowing network operators to take full control of their infrastructure with unprecedented levels of automation and precision. These technologies provide the foundation for the future of networking, offering significant benefits in terms of agility, innovation and efficiency.

2.2.4 Open-Source Control Plane Landscape

Open-source control plane software development has intensified since 2005, some examples being VyOS [15], OpenBGPD [16], BIRD [17], ExaBGP [18], GoBGP [19] and FRR [20]. FRR is more versatile and supports protocols such as BGP and MPLS. It is used by former Cumulus Networks (acquired by Mellanox/NVIDIA) and SONIC (Rest of the World).

However, the previously mentioned control plane software projects have their specific use cases, and some come with their own specific history and technical debt. There have been several attempts to create new open-source control plane software written in a secure programming language able to leverage new CPU architectures, such as GoBGP and Holo [21]. Both these projects seem promising and are starting to gain attention from the industry and standardisation bodies.

2.3 Data Plane Programmability

Data plane programmability enables the implementation of custom packet processing algorithms. The data plane algorithms are responsible for processing all the packets that pass through a telecommunication system. Thus, they ultimately define the functionality, performance, and scalability of such systems. Any attempt to implement data plane functionality in the control plane typically leads to significant performance degradation. However, with data plane programming, users can build custom network equipment without any compromise in performance, scalability, speed or energy consumption.

For custom networks, new control planes and SDN applications can be designed, and users can design data plane algorithms that fit them ideally. Data plane programming does not necessarily imply any provision of APIs for users, nor does it require support for outside control planes as in OpenFlow. Device vendors might still decide to develop a proprietary control plane and use data plane programming only for their own benefit without necessarily making their systems more open (although many do open their systems now).

The following sections take an in-depth look at three central aspects that are shaping the implementation and effectiveness of data plane programmability. First, data plane programming models, which offer frameworks and abstractions to simplify the development and deployment of data plane functionalities, are discussed. Next, the various types of devices and targets, focusing on software and hardware platforms, are described. Finally, data plane programming languages, which provide the syntax and semantics for defining the behaviour of network devices, are explored.

2.3.1 Data Plane Programming Models

Data plane algorithms are expressed using standard programming languages, including P4, NPL, C++, and others. However, they do not map particularly effectively onto specialised hardware such as high-speed ASICs. Consequently, data plane models are proposed as abstractions of the hardware. Data plane programming languages are tailored to those data plane models and provide mechanisms to express algorithms for them in an abstract way. The resulting code is then compiled for execution on a specific packet processing node supporting the respective data plane programming model. Protocol-Independent Switch Architecture (PISA), depicted in Figure 3.4 is an example of a P4-based data plane programming model.

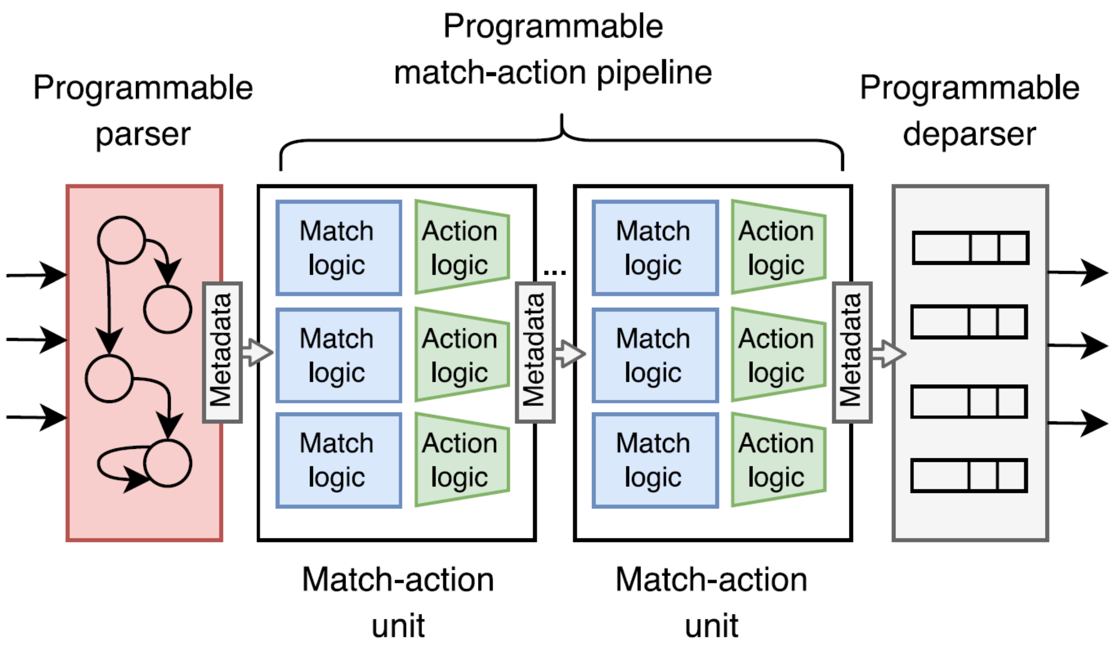

Figure 2.1: PISA data plane programming model - Image Source: [22]

The PISA model is based on the concept of a programmable match-action, which is perfectly aligned with modern switching hardware architectures. The PISA model comprises a programmable parser, a programmable deparser, and a programmable match-action pipeline formed by multiple stages. The programmable parser allows for the declaration of arbitrary headers together in a finite state machine that specifies the order of the headers within packets. In other words, it serialises packet headers into a well-structured format. In contrast, the deparser performs the opposite function, serialising packets so that they can be transmitted through links.

The programmable match-action pipeline is the central element of the model. It comprises multiple match-action units, each including one or more Match-Action Tables (MATs) for matching packets and performing specific actions with the supplied action data. The packet processing algorithm is defined in the match-action pipeline. Each MAT includes matching logic coupled with the memory (Static Random-Access Memory – SRAM or Ternary Content-Addressable Memory – TCAM) to store lookup keys and the corresponding action data. The action logic, which may include arithmetic operations or header modifications, is implemented using Arithmetic Logic Units (ALUs). Additional action logic can be implemented using stateful objects, such as counters, meters, or registers, that are stored in the SRAM. Registers provide a mechanism for maintaining the state between consecutive packets, thereby enabling state-based functions, such as alternate path selection [23] or advanced firewalling [24], to be entirely offloaded to the data plane. A control plane manages the matching logic by populating the MATs to modify the runtime behaviour.

2.4 Devices and Targets

In the context of data plane programmability, understanding the various devices and targets is crucial for effective implementation. This subsection explores the diverse platforms available, including hardware-based, Linux-based and software-based solutions, highlighting the standout systems in each category.

To systematically present different P4-related solutions, the P4 organisation has structured a repository [25] as an ecosystem for experimenting with programmable networks. This ecosystem includes several open-source programs, projects, testbeds, and targets that enable researchers to develop their interest in this field.

2.4.1 Hardware-Based

Hardware data plane programmable targets are specialised networking devices that allow the custom processing of packets directly in the hardware. In contrast with traditional networking devices, these programmable targets provide improved flexibility, allowing network operators to define protocols and mechanisms for specific conditions or scenarios while supporting high-performance packet forwarding.

Depending on the architecture and the form factor, targets can be of different types: programmable ASICs, Smart NICs, Field-Programmable Gate Arrays (FPGAs), or Data Processing Units (DPUs).

2.4.1.1 High-Performance Programmable ASICs

Data Plane Programmability (DPP) allows for the use of custom algorithms that define sets of match-action rules to control the forwarding of data packets. However, the data plane remains static, which means that packets are forwarded according to the same match-action tables. DPP means that the network users, not just vendors, have the capacity to define custom data plane algorithms.

DPP was a feature during most of the networking industry's history because data plane algorithms were typically executed on general-purpose Central Processing Units (CPUs) programmed by the vendors. With the advent of high-speed links, the processing capabilities of CPUs were exceeded, and fixed packet processing ASICs were introduced. At this point, data plane programmability presented an issue because general-purpose CPUs were not capable of processing traffic coming from high-speed links and fixed ASICs slowed down the development cycle of new networking functionalities. High-performance programmable ASICs were introduced as a solution to implement custom data plane algorithms capable of processing packets at line-rate.

Programmable ASICs offer a middle ground between traditional fixed-function ASICs and fully programmable general-purpose processors, allowing for specialised, high-performance processing while also enabling flexibility and adaptability. Unlike fixed-function ASICs that are designed to perform a specific task with limited options for modification, programmable ASICs can be dynamically configured to support a range of functions or protocols. This ability to tailor functionality post-deployment makes programmable ASICs highly valuable in applications where requirements evolve rapidly, particularly in networking and data centre environments.

One of the core benefits of programmable ASICs is to propose a programmable SDK that exposes the API via an open language such as P4. This results in an ability to implement new protocols, policies, and network standards in a flexible way at a fast pace. ASIC programmability allows it to adjust to new packet-processing rules as they evolve, which is particularly important in environments where network functionality needs frequent updates. Network operators can, on their own, implement and test new protocols or features without requiring new hardware and most importantly potentially assistance from the vendor.

Another interesting aspect of programmable ASICs is the combination of the flexibility of software-defined capabilities with the performance efficiency of hardware-specific designs. For example, Tofino ASICs can process packets at terabit speeds while offering precise control over how each packet is handled. This high level of customisability ensures that network operators can optimise performance for specific tasks, such as load balancing or traffic shaping.

Deploying fixed-function ASICs can become expensive if the hardware quickly becomes obsolete due to new standards or evolving needs. Programmable ASICs extend the useful life of network hardware by enabling software-driven updates. This adaptability means that organisations can use the same hardware for a more extended period, saving on upgrade and replacement costs. This also applies to security, as programmable ASICs allow for faster and more granular updates to security policies. They enable custom traffic filtering, real-time threat mitigation, and the implementation of specialised security policies. Then, with enough knowledge of programming tools and languages and enough resources, this ability to adapt to emerging threats helps organisations keep their network infrastructure secure while reducing downtime and maintenance efforts.

Programmable ASICs provide the ability to quickly deploy new services or applications, shortening the time-to-market for innovative services, and giving providers the flexibility to adapt to customer demands more rapidly. Additionally, they provide a bridge between hardware efficiency and software flexibility, offering substantial advantages in adaptability, performance, security, and cost-efficiency. This combination of programmability with the speed of dedicated hardware makes programmable ASICs powerful tools in the hands of network operators and data centre managers.

There are numerous examples where ASICs are used. To mention just a few:

- ASICs specialised hardware chips designed for high-performance packet processing are usually mounted in commercial switches (e.g. Intel Tofino [26], or Broadcom Trident [27]). The traditional reliance on in-house ASIC designs by manufacturers like Cisco and Juniper is shifting towards programmable silicon designs from companies like Broadcom, Marvell, and Intel. Products such as Broadcom's Tomahawk [28] and Trident [29] series, along with Marvell's Teralynx [30], Prestera [31], are commonly used in core and edge networking equipment due to their optimal balance of performance and cost.
- National Research and Education Networks (NRENs) are adopting open networking models through the deployment of White box solutions and open-source platforms [32][33]. These models often utilise ASICs compatible with the Open Compute Project (OCP) or Software for Open Networking in the Cloud (SONiC). Typically based on Broadcom or Marvell silicon, these ASICs are vital for disaggregated networking environments where hardware is decoupled from software, promoting greater flexibility and innovation.
- Networking equipment vendors leverage programmable data planes as a tool to reduce the time-to-market of their solutions. They use programmable ASICs to provide high-performance devices with the latest protocol features and releases. For example, Cisco produces the Silicon One, a P4-programmable ASIC that can process up to 25Tbps, offering 400Gbps interfaces.

- Some of the ASICs have a common interface which is the Switch Abstraction Interface (SAI) used in SONiC. SAI is a switch specification that each ASIC must comply with to work with SONiC and other proprietary NOSs such as the Facebook Open Switching System (FBOSS) from Meta.
- A new P4 ASIC replacement contender from Xsight [34].

2.4.1.2 Intel Tofino

One prominent example of a programmable ASIC is the Intel Tofino [35]. Designed by Barefoot Networks, which Intel acquired in 2019, Tofino is a state-of-the-art P4-programmable Ethernet switch ASIC, specifically tailored for packet processing. The primary advantage of Tofino lies in its utilisation of an open language like P4 while using an ASIC that comes with its own SDK, enabling programmability within high-performance hardware. This programmability allows developers to specify how packets should be processed without requiring hardware changes, making Tofino versatile in handling a wide array of networking protocols, formats, and new service capabilities. In an era of expanding data volumes and ever-changing network requirements, Tofino's flexibility stands out as a significant advantage.

However, when Intel officially discontinued the Tofino roadmap in 2023 [36] and announced End of Sale (EOS) and End of Life (EoL) for their product lines in Q4 2024, this looked to be a major setback for network programmable solutions. Even after these announcements, at the 2024 P4 Workshop [37], Intel presented its most relevant P4 initiatives, which included P4-enable IPSec, P4-enable Kubernetes, P4-enables vSwitch, P4 support for Infrastructure Programmer Development Kit (IPDK), and P4 support in the Linux Kernel (P4-TC) [38]. In addition, Intel shared their thoughts on the future of the P4 language, highlighting its integration with AI/ML and the open sourcing of Tofino as a benefit for the community.

Since the beginning of 2025, Tofino has made their software open "for anyone developing Tofino-specific P4 data plane programs (both P414 and P416 versions) and related control plane applications to openly publish their data plane and control plane code" [Open-Tofino]. The Tofino compiler (or some part of it) is also open-sourced in the p4lang GitHub repository [39], which promises future development of network programmable solutions based on the Tofino platform.

2.4.1.3 Smart Network Interface Cards (NICs) / Network Processing Units (NPUs)

Smart Network Interface Cards (NICs) / Network Processing Units (NPUs) are advanced networking interface card modules that can be programmed to perform custom mechanisms directly in the card itself, which are mainly used to offload network-oriented functions from the CPU (e.g., AMD Pensando [40] or Netronome [41]). Such cards are considered for processing large amounts of complex data, including but not limited to multimedia data such as images and videos and in scenarios where machine learning, artificial intelligence or neural network algorithms and techniques are required.

2.4.1.4 Field-Programmable Gate Arrays (FPGAs)

FPGAs are reconfigurable hardware devices that can be programmed to perform network-specific functions and offload work from the main computational and storage units. FPGAs can be programmed to execute complex, parallel processing workflows efficiently, which is particularly advantageous in network environments that demand rapid packet handling, filtering, and forwarding. They also excel in high-speed data processing tasks, including packet processing in networking applications. Unlike general-purpose processors, FPGAs can process packets at wire speed and with minimal latency, making them essential for high-performance network functions such as routing, firewalling, telemetry, and QoS management [42] [43]. AMD provides the Vitis Networking P4 [44] environment to design packet-processing data planes that target FPGA hardware. There are several device families, such as Zynq [45], Virtex [46], and Versal [47], that support this IP module.

2.4.1.5 Data Processing Units (DPUs)

While the DPU concept is rather old, it continues to evolve over time. Normally, DPUs involve a tight integration between a general CPU and a specific purpose core. This concept is different from the Smart NICs, which are network interface cards that provide additional programmability to offload specific tasks from host systems. Smart NICs alleviate the burden from the host CPU as part of the overall server, while DPUs are more focused on being independent infrastructure endpoints [48].

One of the more successful DPUs was the Cavium Octeon II [49], which provided hardware acceleration for tasks such as packet processing, deep packet inspection with History-based Finite Automaton (HFA) and comprehensive crypto algorithms.

These chips were part of several appliances, including routers, security appliances, and storage coprocessors. The SDK comprised SMP LINUX support, a Simple Executive for data plane applications, the complete GNU toolchain, optimized C libraries for security, and support for run-to-completion or pipelined software models.

The emergence of SDN has brought a renaissance in specialised hardware designed to accelerate and offload workloads from general-purpose CPUs. Therefore, DPUs have attracted the attention of many vendors, and there are now a large number of alternatives, as described in [50]. As an example, in 2018 Cavium was acquired by Marvell, which has been working on the evolution of the DPU technology. A representative of this evolution is the Marvell Octeon 10 [51]. Marvell defines a DPU as a computing entity that moves, processes, secures, and manages data to make it available and optimise it for applications.

Marvell's Octeon 10 comprises compute and accelerator cores along with high-speed interfaces. As shown in the Octeon 10 Block Diagram depicted in Figure 5.2, it offers not only ARM cores (from 8 to 24 64-bit ARM Neoverse N2 cores) but also support for Vector Packet Processing (VPP) units, Scalable Vector Extension V2 (SVE2), enhanced cryptography instructions and hardware-based AI/ML acceleration.

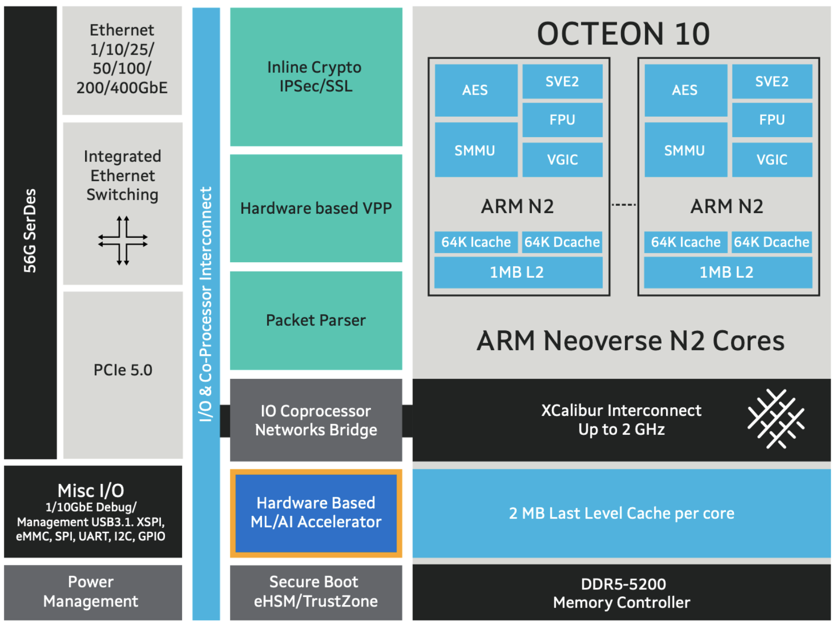
56G SerDes
Ethernet 1/10/25/50/100/200/400GbE
Integrated Ethernet Switching
PCIe 5.0
I/O & Co-Processor Interconnect
Inline Crypto IPSec/SSL
Hardware based VPP
Packet Parser
IO Coprocessor Networks Bridge
OCTEON 10
AES
SVE2
FPU
SMMU
VGIC
ARM N2
64K Icache
64K Dcache
1MB L2
ARM Neoverse N2 Cores
XCalibur Interconnect Up to 2 GHz
Misc I/O 1/10GbE Debug/ Management USB3.1. XSPI, eMMC, SPI, UART, I2C, GPIO
Hardware Based ML/AI Accelerator
2 MB Last Level Cache per core
Power Management
Secure Boot eHSM/TrustZone
DDR5-5200 Memory Controller

Figure 2.2: The Marvell Octeon 10 block diagram [52].

While PCIe boards exist that can be used to improve network capability, this architecture is designed to be integrated into switching equipment. In fact, there are several manufacturers, such as CloudSwitch [53], that are starting to offer modular gateway platforms with these DPUs with different switching capacities. Perhaps more importantly, well-known programming paradigms such as VPP and DPDK are available, enabling integration with a complete ecosystem of applications.

2.4.1.6 White Boxes

In the networking world, a “white box” typically refers to a generic, low-cost networking device from an original design manufacturer that can be tailored to specific requirements and offer an alternative to proprietary solutions.

Manufacturers such as Edgecore, Delta, and Celestica are producing high-performance generic hardware capable of running various software stacks, including Software for Open Networking in the Cloud (SONiC) [54], Cumulus Linux [55], or proprietary systems such as RTBricks [56], IP infusion [57] or Arrcus [58]. This trend towards white-box networking is complemented by a disaggregated approach, where the network operating system (NOS) is chosen independently of the hardware. This approach is popular with large hyperscalers such as Google and Facebook, leading to the emergence of platforms such as Pica8 and Cumulus (now part of NVIDIA), which provide highly customisable networking software [59]. By adopting these disaggregated solutions, NRENs can tailor their networks to specific needs, driving innovation and efficiency in network management. In addition, this approach supports the integration of cutting-edge technologies and facilitates rapid adaptation to evolving network requirements, ensuring robust and scalable network infrastructures.

It is likely that, with time, more solutions and platforms will emerge, and there are some examples of new collaborations in the area of P4 programmable devices being announced, such as between Oxide and Xsight Labs to build the next generation of P4-programmable networks on the Oxide Cloud Computer [60].

2.4.2 Linux-Based Packet Processing Solutions

There are two linux-based packet processing solutions – Data Plane Development Kit (DPDK) and extended Berkeley Packet Filter (eBPF) / eXpress Data Path (XDP).

2.4.2.1 Data Plane Development Kit – DPDK

Traditionally, networking equipment provided by vendors such as Cisco, Ericsson, Huawei, Juniper, Nokia, and ZTE used specific ASICs to perform low-level data plane functions like packet processing. The main constraint for these ASICs was the long schedule for introducing new products, which was limited by the silicon development/debug cycles. In this context, DPDK [61] was created as a solution to perform efficient data plane functions using general-purpose CPUs.

DPDK is a set of software libraries and drivers, running in user space, that accelerate packet-processing workloads running on all major CPU architectures. DPDK was created by Intel but now is housed as a project under the Linux Foundation. DPDK allows the use of general-purpose CPUs in high-performance environments, from enterprise data centres to public clouds and particularly in communication networks.

To accelerate the processing of packets, DPDK allows incoming network packets to transition to user space with no overhead for memory copying, where they are rapidly processed without the expense of context switching between user space and kernel space. In other words, DPDK bypasses the Linux kernel, performing packet processing in user space to maximise networking performance. DPDK achieves this using a Poll-Mode Driver (PMD) running in user space that continually checks incoming packet queues to see if new data has arrived, achieving both high throughput and low latency.

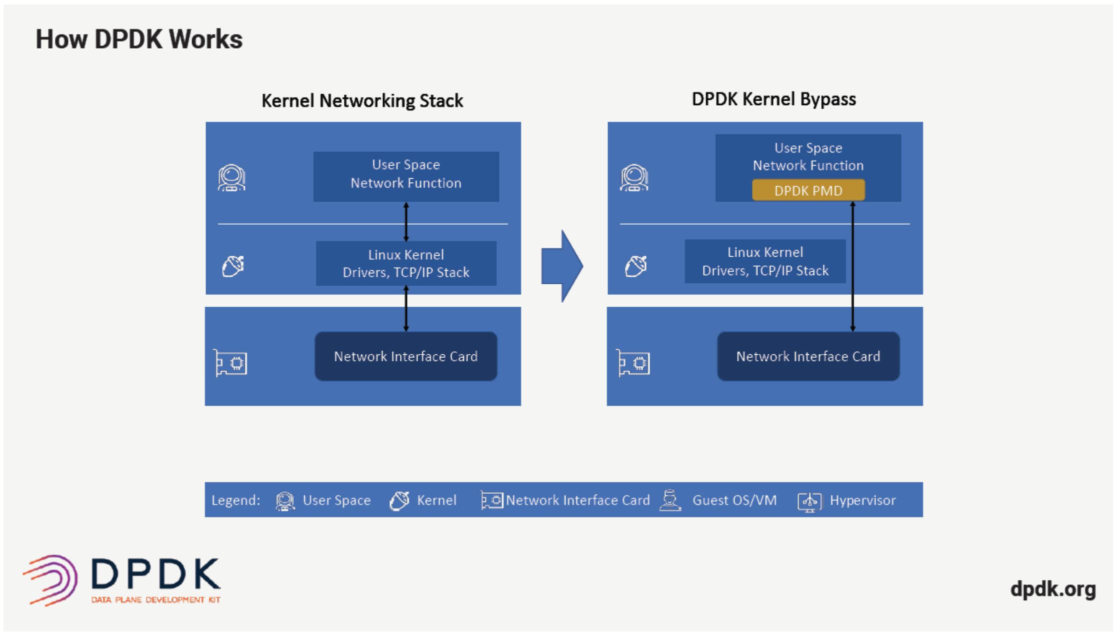
How DPDK Works
Kernel Networking Stack
User Space Network Function
Linux Kernel Drivers, TCP/IP Stack
Network Interface Card
DPDK Kernel Bypass
User Space Network Function
DPDK PMD
Linux Kernel Drivers, TCP/IP Stack
Network Interface Card
Legend: User Space Kernel Network Interface Card Guest OS/VM Hypervisor
DPDK DATA PLANE DEVELOPMENT KIT
dpdk.org

Figure 2.3: DPDK kernel bypass compared to the standard Linux kernel stack [3]

Since 2019, there has also been a surge in the adoption of kernel by-pass mechanisms. DPDK was very popular, however, this was a very low-level C library dedicated to developing hardware-accelerated packet forwarding by NIC chipsets and there was no high-level library available that would abstract DPDK C programming at that time. Vector Packet Processor – VPP [67], in combination with the Linux control plane plugin [68], are enabling DPDK hardware acceleration for all netlink-based control plane software.

2.4.2.2 Extended Berkeley Packet Filter (eBPF) / eXpress Data Path (XDP)

eBPF [62] [63] is a Linux-based technology that allows programmers to run sandboxed programs in the operating system kernel. It is used to safely and efficiently extend the capabilities of the kernel without requiring changes to the kernel source code or needing to load kernel modules.

Historically, the operating system has always been the ideal place to implement observability, security, and networking functionality due to its kernel's privileged ability to oversee and control the entire system. At the same time, an operating system kernel is hard to evolve due to its central role and high requirement for stability and security. The rate of innovation at the operating system level has thus traditionally been lower compared to functionality implemented outside of it.

XDP [64] provides a framework for BPF that enables high-performance programmable packet processing in the Linux kernel. It runs the eBPF program at the earliest possible point in the software, i.e., when the network driver receives the packet. This means the packet is intercepted before any expensive operations such as pushing the packet up the networking stack have taken place. Thus, the XDP BPF program is executed at the earliest point where it becomes available to the CPU for processing.

In contrast to DPDK, XDP works in the Linux kernel, so it does not bypass the kernel to operate in the user space. Keeping the packet in the kernel space has several major advantages:

- It can reuse all the kernel networking drivers, user space tooling, or even other available in-kernel infrastructure such as routing tables, sockets, etc. developed upstream in BPF helper calls.
- It has the same security model as the rest of the kernel for accessing hardware.

- There is no need for crossing kernel/user space boundaries since the processed packet already resides in the kernel and can therefore flexibly forward packets into other in-kernel entities like namespaces used by containers or the kernel's networking stack itself. This is particularly relevant in case of Meltdown and Spectre (security vulnerabilities recently found in Intel, AMD, Apple, and ARM processor chips), as minimising unnecessary transitions between kernel and user spaces can help mitigate performance overhead while ensuring security.
- It is possible to handle packets by the regular Linux TCP/IP stack (part of the kernel) without having a separate TCP/IP stack in user space.
- It allows for full programmability, keeping a stable application binary interface (ABI) with the same "never-break-user-space" guarantees as with the kernel.
- It allows for atomically swapping programs during runtime without any network traffic interruption or even kernel/system reboot.
- XDP allows for flexible structuring of workloads integrated into the kernel. For example, it can operate in "busy polling" or "interrupt-driven" mode. Explicitly dedicating CPUs to XDP is not required. There are no specific hardware requirements, and it does not rely on huge pages.
- It supports most major 10G or higher networking drivers.

2.4.3 Software-Based Targets

Software-based targets are packet-forwarding programs that run on a standard CPU. Software-based targets provide an abstraction layer that emulates the behaviour of a programmable data plane on a standard Linux-based host.

2.4.3.1 Behavioral Model Version 2 (bmv2)

Behavioral model version 2 (bmv2) [65] is the most widely known software-based target. It implements Portable Switch Architecture (PSA) and v1model architectures to run P4 programs. While bmv2 is easy to use and readily available, due to its software-based nature this is mainly used for training, demonstrations, and development rather than for production purposes.

2.4.3.2 Intel P4 Studio Software Development Environment (SDE)

Intel P4 Studio Software Development Environment (SDE), often referred to as Open P4 Studio [66] is a set of packages that could be used to develop software for Intel's family of programmable Ethernet Switches. This open-source version includes Barefoot Runtime Interface (BRI) with the gRPC-based protocol, called BF Runtime, and allows development using the bf_switchd virtual model. However, P4Insight GUI for visualising the hardware resources used by P4 programs, Board Support Package (BSP) and ASIC-specific drivers are not included in the open-source bundle, as explained in [67].

2.4.3.3 Switch Abstraction Interface (SAI)

Switch Abstraction Interface (SAI) is a standardised Application Programming Interface (API) which provides the opportunity to use a consistent and simple programming interface to develop diverse and innovative functionalities on various hardware platforms. Although it is not a target to deploy data plane programs, as they are bmv2 of the bf_switchd, SAI provides an agnostic and vendor-independent way of controlling switch entities like ASICs, NPUs or software switches. SAI enables open networking as it allows the use of the same application stack on different hardware platforms irrespective of vendors, executing true software-hardware decoupling. Several organisations implemented SAI for their solutions, such as Cisco in its Silicon One home-made ASIC, or Intel, who implemented SAI in 2019 as part of their Intel P4 Studio.

2.5 Data Plane Programming Languages

In data plane programmability, data plane programming languages play a crucial role in defining and controlling the behaviour of network devices. This subsection provides an in-depth view of these languages as they have been developed throughout the years, exploring their unique features, advantages, and specific characteristics.

2.5.1 P4 Language

The P4 programming language [68] was created to provide a high-level, protocol-independent way to define the behaviour of packet-processing devices. It allows the programmers to describe data plane algorithms using a combination of constructs that provide support for the typical data-plane-specific functionality (e.g., counters, meters, checksum calculations, etc.). The first version of the P4 language (P414) was released in 2014. In 2016, the next version, P416, was introduced to address several P414 limitations. The major contribution of the P416 standard is that it supports multiple different targets and pipeline architectures. This is achieved by separating the core language from the specifics of a given architecture thus making it architecture-agnostic. The structure, capabilities and interfaces of a specific pipeline are encapsulated into an architecture description, while the architecture- or target-specific functions are accessible through an architecture library, typically provided by the target vendor.

The remainder of this paper will focus on P416 and any further reference to the P4 language is implicitly intended to refer to that version. P4 programs are supplied by the user and are implemented for a particular P4 architecture model. They define algorithms that will be executed by the P4-programmable components and their interaction with those implemented in the fixed-function logic. The composition of the P4 programs and the fixed-function logic constitute the full data plane algorithm.

P4 compilers are also provided by the manufacturers, they translate P4 programs into target-specific code, which is loaded and executed by the P4 target. The P4 compiler generates a data plane API that can be used by a user-supplied control plane to manage the runtime behaviour of the P4 target. For instance, the control plane will use this API to fill the data plane tables that are defined in the P4 program.

Figure 3.1 taken from [69] depicts the information flow in a P4 processing pipeline, which is divided into different blocks that perform operations to process the packets. The packet headers and metadata are used to pass the information between them, therefore representing a uniform interface.

The parser splits up the received packet into individual headers and the remaining payload. Intrinsic metadata from the ingress block, e.g., the ingress port number or the ingress timestamp, is often provided by the hardware and can be made available for further processing. Many targets allow the user metadata to be initialised in the parser as well.

The headers and metadata are then passed to the match-action pipeline that consists of one or more match-action units. The remaining payload travels separately and cannot be directly affected by the match-action pipeline processing. While traversing the individual match-action pipeline units, the headers can be added, modified, or removed and additional metadata can be generated.

The deparser reassembles the packet by emitting the specified headers followed by the original packet payload. Packet output is configured with intrinsic metadata that includes information such as a drop flag, desired egress port, queue number, etc.

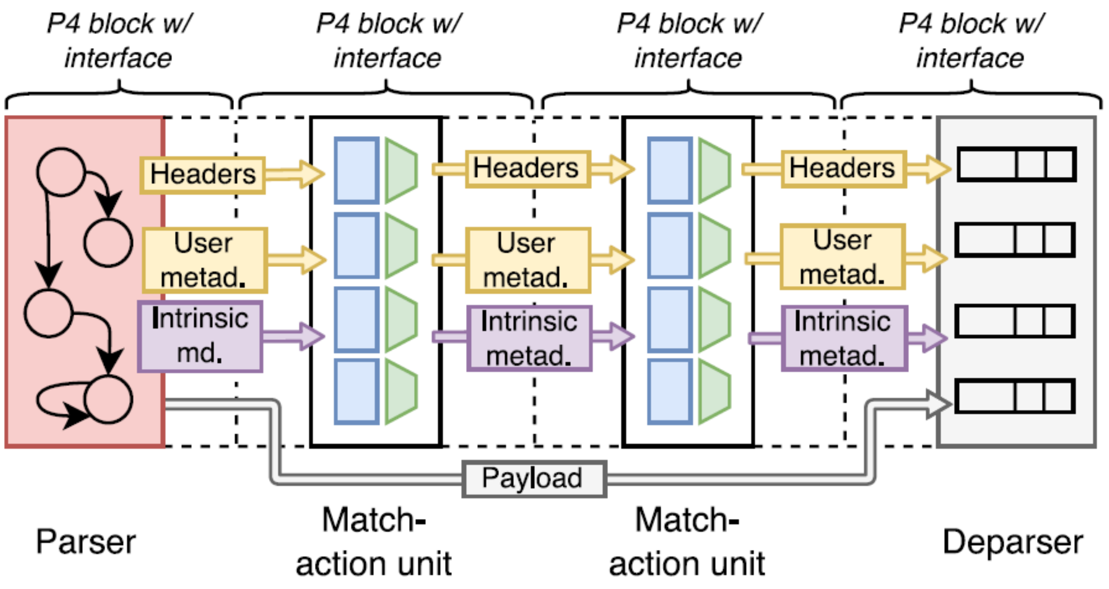

Figure 2.4: Information flow in the P4 processing pipeline, Picture Source [70]

2.5.2 Network Programming Language - NPL

Network Programming Language (NPL) [71] is an open, high-level language that also addresses the requirements of efficiently programming data planes. Like P4, NPL expresses network behaviour using constructs that take advantage of advanced features of the underlying programmable hardware. NPL building blocks, presented in Figure 3.2 from [72], range from data types that allow the specification of individual control signals to high-level constructs that allow interfacing with complex hardware blocks. NPL includes the following core abstractions:

- Data Types: specifies the basic building blocks of any object field.
- Parser: specifies the allowed headers within received packets and extracts those headers from the packets.
- Logical Bus: specifies the fields and overlays of a logical bus. The logical bus connects various other NPL objects.
- Match Action (MA) table: describes a particular table with the associated keys and actions.
- Editor: provides the ability to add, remove, or replace a header.
- Special Function: a mechanism to call a particular hardware function that may be treated as intellectual property. This provides a structured mechanism to define the interface into these functions without revealing the contents of the function.
- Function: provides programmable decision logic without the overhead of a table. For example, it can be used to resolve the results of multiple Match Actions or to resolve the Match Action key selection.
- Strength Resolution: a mechanism to resolve multiple tables updating the same object in parallel.
- Packet Drop, Packet Trace, and Packet Count: built-in functions to drop, trace, and count packets.
- Create Checksum and Update Packet Length: built-in functions to create checksum and update packet lengths.
- Metadata for MA and Parser: data not created in the NPL yet still exists at runtime with the packet and can be used by the NPL.

The NPL language is not bound to any specific hardware architecture, so it can be implemented on multiple hardware platforms such as programmable ASICs, programmable NICs, FPGAs, and software switches. However, certain language constructs are intended to optimise the use of specific hardware features on certain targets.

Like any high-level programming language, NPL requires a set of compilers and associated tools to map the programs written to target hardware objects. The front-end compiler is responsible for checking the syntax and semantics of the user-written program in the NPL language generating an Intermediate Representation (IR). The back-end compiler is responsible for mapping these intermediate representations into specific hardware objects. It also generates an API that the control plane uses to manage the behaviour of the switch.

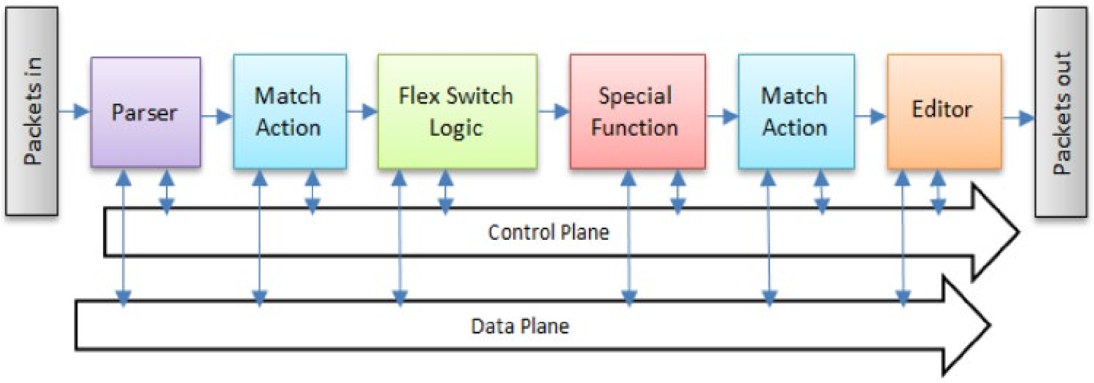

Figure 2.5: NPL architecture model, Picture source [73]

2.5.3 C/C++

C++ [74] is a powerful language choice for network programming and has been extensively used for programmable network devices such as smart NICs, FPGAs, and NPUs. One of the main advantages of this programming language is its focus on high-performance environments. It is heavily used in performance-oriented applications thanks to the direct control it provides over memory management, essential for real-time applications and latency-sensitive traffic forwarding. In this context, several libraries and frameworks use C++ for network programmability, such as:

- DPDK: an open-source set of libraries and drivers for high-performance packet processing that supports low-latency applications.
- eBPF/XDP: technology that allows the development of programs that run sandboxed in the kernel. XDP extends it allowing fast packet processing at the NIC level.

For instance, some devices like Netronome Agilio SmarNICs [75] use either eBPF or C language to efficiently program the data plane processing pipelines, providing full control over memory.

2.5.4 Very High Speed Integrated Circuit (VHSIC) Hardware Description Language - VHDL

VHDL, defined in [76], is very often used to design packet processing functionalities for FPGA cards, as it provides a way to define custom logic circuits that process network packets at various protocol layers. The flexibility of VHDL allows for designing logic that processes headers, analyses payloads, and routes or discards packets based on content. All these tasks are performed in hardware, ensuring low latency and high-speed operation.

In a typical packet processing application, FPGAs receive incoming data packets, which are streamed in as a series of bits or bytes. VHDL modules then define a series of processing stages to handle different packet components, such as headers and payloads. Each module within this flow can operate concurrently within the FPGA, leveraging its parallel processing capability. FPGAs can process multiple packets simultaneously or different portions of a single packet at once.

Development in VHDL is challenging and requires in-depth knowledge of both digital circuit design and network protocols. Moreover, FPGAs have finite resources, and implementing complex packet processing functions may require careful optimisation.

3 Network Programmability Use Cases and Examples

The last several years have seen important changes in networking architecture, hardware and software, and solutions. Organisations have by and large abandoned vertical integration and adopted a strategy that consists in ensuring a high level of independence from specific hardware vendors. Google has undertaken work on P4-based Automated Reasoning [77]. Meta has abandoned SDK-based development and embraced [78] Switch Abstraction Interface (SAI), while Alibaba has decided to use SONiC [79] and is spearheading a routing working group to ensure that SONIC does not remain limited to data centre architecture but is also deployed in a MAN and WAN context. These developments are evidence that there are numerous areas where network programmability can be of help and use.

This chapter presents some exploratory use cases, such as the possible advantages network programmability can provide for future 6G networks and support of artificial intelligence (AI) and machine learning (ML). More practical examples are also provided, such as the use of network programmable solutions in data centres, at network access level or flow monitoring. Special emphasis is given to RARE – Router for Academia, Research and Education, its deployments on the Global Platform for Labs (GP4L), and numerous use cases. These scenarios focus on the use of network programmability as a tool, to demonstrate network advancements through network automation and development and supporting the standardisation of new network protocols.

3.1 Network Programmability in 6G

New generations of mobile communication systems typically also set the trend for the upcoming technologies and procedures that are deployed in fixed communication networks. For example, the 5G standard natively integrates technologies such as NFV and SDN, which imply the use of network virtualisation and network programmability in 5G networks. Although 5G technology is not yet fully operational, research and academia are already shaping the sixth generation of communications (6G).

The key pillars of 6G technology are currently being defined based on emerging trends and research. These pillars will underpin the future development of 6G, enabling it to surpass 5G in terms of capability and innovation. Several sources in the industry have identified which of these pillars are primary along with the technological enablers required to achieve them, such as Native Artificial Intelligence (AI), Extreme Connectivity, and Native Trustworthiness and Security [80] [81].

Deep network programmability, which refers to programming the network both vertically (control and data plane) and horizontally (end-to-end from the radio edge to the core network), is expected to play a relevant role in 6G networks. In this way, 6G will enable a more flexible, dynamic, and efficient management of both the control and data planes to support extreme performance requirements and service-specific operations. This will allow networks to rapidly adapt to evolving demands and support advanced use cases, such as AI-driven automation, intelligent resource allocation, and ultra-reliable low-latency communication (URLLC). Programmable control planes contribute to 6G through:

- **Dynamic Network Management**: In 6G, the programmable control plane will allow operators to adjust network configurations in real time based on specific application requirements or environmental conditions. This includes adjusting network policies, optimising routing paths, and dynamically allocating resources such as bandwidth or power based on the real-time needs of applications like autonomous vehicles, smart cities, or telemedicine.

- **Network Slicing**: A key feature of 6G will be enhanced network slicing, where a single physical network can be partitioned into multiple virtual networks, each optimised for a specific service or application. Programmable control planes are crucial for this, enabling the definition and modification of slices based on different parameters such as latency, throughput, and security.
- **AI-Enhanced Control**: AI will be integrated into the control plane to enable predictive and self-optimising networks. By leveraging programmability, AI systems can dynamically reconfigure networks to optimise performance, energy efficiency, or cost. This will help 6G networks become more autonomous and responsive to real-time demands [82] [83].

Programmable data planes contribute to 6G through:

- **Customised Data Handling**: Programmable data planes enable more sophisticated packet processing and flow management. For example, data can be processed in-network, reducing latency for critical applications. This is especially relevant for use cases such as immersive XR (Extended Reality) or tactile internet, where real-time data processing is crucial.
- **Offloading Tasks**: A programmable data plane can offload computational tasks from end devices, allowing edge computing nodes to handle data processing directly within the network. This reduces the need for raw data transmission back to central servers, optimising bandwidth use and improving latency [84]. Additionally, the expansion of switches and devices with Data Processing Units fosters the acceleration of computational functions in network equipment. As an example, devices that combine programmable data plane technologies and DPUs to increase the variety of tasks that can be offloaded to the data plane are gaining traction [85], as described in the previous section.

6G will rely heavily on edge computing to meet latency and bandwidth requirements for next-generation applications. Programmable control and data planes allow processing to be offloaded to edge nodes. This distributed intelligence across the network ensures that performance-critical tasks are handled with minimal delay. Programmability will also be key in enhancing the security and privacy of 6G networks. It allows for the real-time implementation of security protocols, detection of malicious traffic, and isolation of affected network slices or devices. Network programmability in 6G allows network operators to quickly adapt to new service requirements and adjust their networks without needing major hardware modifications. It also enables networks to handle a wide range of services and devices, scaling resources dynamically as user demands grow.

3.2 AI/ML and Network Programmability

In recent years, various data plane programming languages (e.g. P4, eBPF/XDP, and DPDK) have been proposed to accelerate traditional networking operations. Researchers of both academic and commercial environments have widely investigated the application of the aforementioned solutions within their infrastructures. However, none of these languages have been widely adopted as a standard solution in modern networking environments.

Data plane programming is not straightforward and introduces a significant learning curve for developers. Learning novel instruction sets is usually expected, whereas developers are required to adhere to the restrictions imposed by the data plane programming languages; these restrictions (e.g. the lack of direct support for loops) safeguard the security of the systems utilised as the program targets. Therefore, the prevalence of such techniques in data centres and core networks is typically hindered by the total effort required by researchers to get accustomed to the intricacies of data plane programming.

3.2.1 AI Applications

The advent of Generative Artificial Intelligence (AI) provides novel opportunities to facilitate development in data plane programming languages. Specifically, Large Language Models (LLMs), such as ChatGPT, have been proposed as suitable candidates for generating data plane software. For example, the solution proposed in [86] leverages ChatGPT to produce valid data plane programs based on instructions provided in natural language and generate programs in Lucid (i.e. a P4 refinement) of varying complexity by selecting the appropriate ChatGPT prompts and providing the language syntax rules.

Another important aspect of data plane programming involves assessing the quality of developed software. Specifically, programs may include bugs and/or important security vulnerabilities that may be exploited by malicious Internet users to compromise critical infrastructures. Various solutions have been proposed to address the aforementioned issue. An example of this is HackP4 [87], a tool that analyses P4 programs both statically and dynamically to discover potential security vulnerabilities.

3.2.2 ML Applications

For network providers, as well as security organisations and intelligence services, the use of probes to detect network anomalies is crucial. In addition to identifying anomalies from normal traffic patterns, it is useful to detect known threat patterns. Known threats are traditionally detected using rule-based approaches like Snort [88], which is a commonly open-source Intrusion Detection (IDS) and Prevention System (IPS), where the “rule” book is regularly updated. With the relevant training sets, it is also possible to convert this rule-based approach into a statistical ML-based approach.

Unknown threats and malicious patterns including zero-day attack patterns cannot normally be detected using rule-based approaches, yet they can be indicated based on variations from the normal data pattern. ML, in contrast to rule-based approaches, is quite suitable for such anomaly detection.

High-performance P4-programmable switches, such as those using Tofino chips, can be utilised for probes if ML is integrated with the data plane. This integration allows for the detection of network intrusions at line-rate, as the traffic is processed directly within the switch, eliminating the need to offload replicated traffic to a dedicated endpoint server for ML analysis. However, implementing more advanced or complex detection models for in-network traffic analysis and anomaly detection increases hardware resource demands, which can affect the performance of the network devices.

Artificial neural networks, particularly autoencoders, are well-suited for unsupervised learning tasks. There are existing frameworks that can map a single autoencoder to the match-action pipeline of a P4-programmable switch, enabling these switches to handle the necessary ML tasks.

Since zero-day attacks are inherently unknown beforehand, unsupervised learning is often preferred. Unsupervised learning tasks typically include clustering, density estimation, or anomaly detection. In anomaly detection, the model identifies data values that deviate significantly from other observations, which may indicate attack behaviour. Anomaly detection is similar to density estimation, where the goal is to determine the likelihood of future observations based on past data, as an unlikely set of observed values can be considered an anomaly.

When an ML model has been trained using a suitable dataset, which is far from trivial, it is deployed into production on a P4-programmable switch using one of the available mapping frameworks like Planter and/or Mousika. For example, Planter has the “Common P4” module for generating use case P4 code for deploying an ML model. Training features, such as TCP source and destination ports, are extracted through the ingress parser. Planter links these features with feature fields in the "user-defined" metadata from where they can be used as keys in the corresponding feature M/A tables of the mapped ML model. Figure 5.3 shows the linkage between the normal P4 parser flow and the mapped ML model.

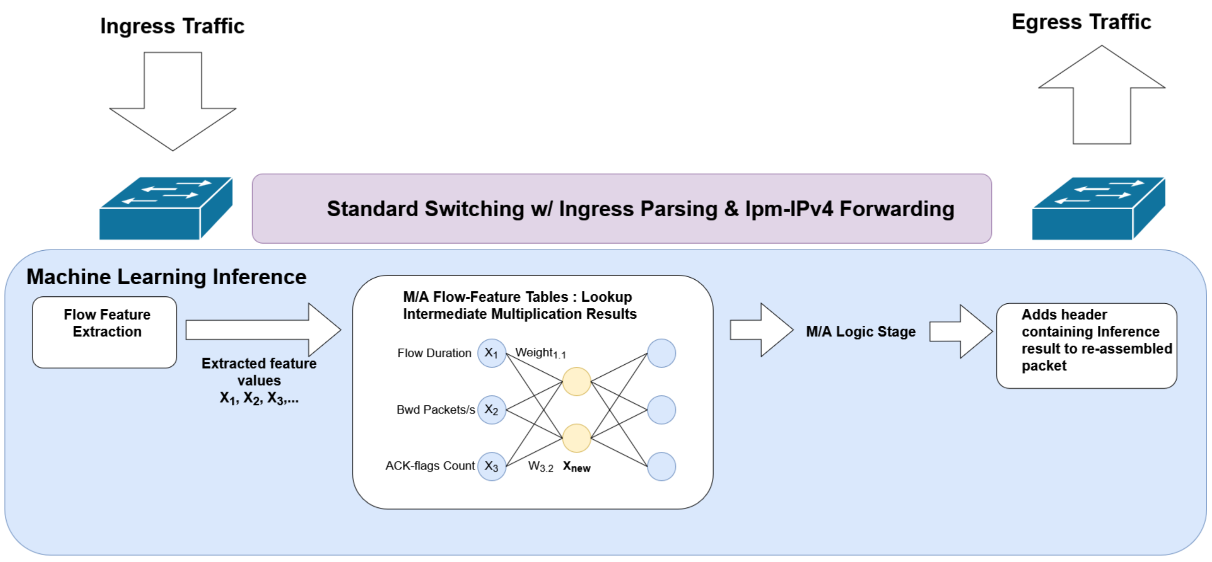
Ingress Traffic
Egress Traffic
Standard Switching w/ Ingress Parsing & Ipm-IPv4 Forwarding
Machine Learning Inference
Flow Feature Extraction
Extracted feature values $X_1, X_2, X_3, \ldots$
M/A Flow-Feature Tables : Lookup Intermediate Multiplication Results
Flow Duration
Bwd Packets/s
ACK-flags Count
X_1
X_2
X_3
$\text{Weight}_{1.1}$
$W_{3.2}$
X_{new}
M/A Logic Stage
Adds header containing Inference result to re-assembled packet

Figure 3.1: Overview of a P4 program for deploying an ML detection model to a P4-switch.

One approach is to focus on packet-level features like TCP and IP fields. Another approach, characteristically used for DDoS and Overload attacks, is to focus on anomalies in packet interarrival times. Such data can be extracted in P4 as metadata and distributed to a control or management plane. It is worth noting that, when monitoring for security threats across network infrastructures it is necessary to propagate security-related alerts throughout the network infrastructure to other systems.

3.3 Network Programmability in Data Centres

Network programmability can be useful for interconnecting physical devices and virtual machines in data centre networks (DCNs). Connectivity management in DCNs has always presented a challenge. ISPs need to maintain thousands of physical and virtual devices interconnected in a DCN. To do this, they typically use leaf-spine topologies that provide enough redundancy to guarantee L2 connectivity even in the event of link or node failures. Additional protocols, such as the Spanning Tree Protocol (STP), are required to avoid loops in such topologies. However, the performance of STP is very limited due to the time that it takes to converge after any changes in the topology. There are solutions, such as the one described in RFC 7938 [89], that try to solve this problem, while other solutions are leveraging network programmability to overcome performance limitations.

In 2015, Google presented Jupiter [90], a scalable data centre network fabric designed to handle the enormous data demands of Google's services (YouTube, Gmail, etc.) with high throughput, low latency, and fault tolerance. Today, Jupiter supports more than 6Pb/sec of data centre bandwidth. This level of performance is delivered by leveraging three concepts [91]:

- An SDN-based network architecture with a logically centralised and hierarchical control plane to program and manage the thousands of switching chips in the data centre network.
- A non-blocking multistage switching topology (Clos), built out of smaller radix switch chips, that can scale to arbitrarily large networks.
- Cost-effective, commodity general-purpose Ethernet switching components, based on merchant switch silicon, for a converged storage and data network.

While Google has not officially disclosed extensive details about using P4 and programmable data planes in Jupiter specifically, several factors suggest its likely adoption. For example, Google has invested heavily in programmable switches and NICs, being a pioneer in SDN, and P4 is a natural extension of SDN principles, focusing on the programmability of the data plane. In addition, Google actively contributes to P4.org, exhibiting its strategic interest in this technology. As an example, at the 2024 P4 Workshop [92], Google presented a new surprising use case in which they employed P4 to capture and model the requirements of a fixed-function network ASIC [93].

3.4 Network Programmability for Network Function Virtualisation

Although the trend towards SDN has not significantly changed network operations, it has impacted the perception of networking. Devices are now being designed to support centralised control, allowing for increased network programmability. This capability is particularly important in networks where dynamic provisioning of resources is essential. Alongside SDN, Network Function Virtualisation (NFV) plays a key role in minimising reliance on physical hardware by replacing traditional network appliances with virtualised counterparts. In NFV environments, ASICs that support advanced offloading for virtual network functions (VNFs) are highly valued for their efficiency and performance [94] [95] [96]. In addition, it is important for networks which are increasingly integrating real-time network telemetry into their equipment to facilitate advanced analytics and troubleshooting. The demand for programmable ASICs that can manage granular flow-level data without compromising performance is growing, as these technologies are critical to maintaining robust, scalable, and efficient network infrastructures. This integration of SDN, NFV, and advanced telemetry not only increases network flexibility and innovation but also ensures that networks can quickly adapt to evolving requirements and challenges.

4 Network Programmability in the R&E Community

Exploration and adoption of network programmability in National Research and Education Networks (NRENs) and Global Research and Education Networks (GRENs) are motivated primarily by the drive to achieve openness and flexibility. The level of involvement and interest in these technologies of these various organisations is shaped by a number of factors, among which are an interest in technological innovations, advancements in the network industry and standardisation efforts, availability of hardware and software solutions, and the respective business strategies of the main vendor companies. Several NRENs have looked at different network programmability use cases and solutions, which can broadly be categorised under network monitoring, network security, and exploration of white boxes.

4.1 Network Programmability Support for Network Monitoring

In the area of network monitoring, network programmability was explored for flow monitoring and for in-band network telemetry (INT).

4.1.1 Flow Monitoring in Switch

At Switch, the Swiss NREN, Tofino-based white box switches are used as part of an IPFIX generation platform that scales to many 100/400Gbps interfaces. The Tofino switches are used as Packet Brokers, which collect traffic from many tapped (half-) links, and send it to dedicated IPFIX exporters over a few 100GE or 400GE interfaces. The broker function is implemented as a custom P4 program developed in-house at Switch. This allows good integration with the IPFIX generator, which was also partly developed in-house based on the high-performance Snabb user-space network function platform.

4.1.2 High Performance Flow Monitoring Using Programmable NICs

As a part of the activities of the Monitoring and Management Task of the GN4-3 project Network Technologies and Services Development Work Package (WP6), the team explored how Netronome Agilio CX programmable network interface cards could be used as a low-cost open-source flow monitoring system. A system was developed that was capable of non-sampled line-rate flow monitoring at 10 Gbps [97].

4.1.3 In-Band Network Telemetry (INT)

Several activities have been recorded in the last few years in Research & Education organisations that focus on the use of network programmability for In-Band Network Telemetry (INT) [98]:

- ESnet in the US deployed FPGA-based INT nodes connected to standard ESnet switches in several locations in the US to debug its network's behaviour and traffic.
- AmLight in Brazil deployed NoviFlow switches with INT capabilities to monitor traffic for astronomy users.
- SURF in the Netherlands deployed INT nodes using Tofino-based switches and FPGA cards for research activities around IPv6 path tracking and IPv6 switch latency measurement.

- The Network Development work package of the GN4-3 project deployed INT protocol version 2.1 using P4 on FPGA cards and DPDK at three sites - the Poznan SuperComputing and Networking Centre (PSNC) in Poland, the Czech Education and Scientific NETwork (CESNET) in the Czech Republic, and Fondazione Bruno Kessler (FBK) in Italy, as explained in detail in [99].

4.2 Network Programmability Support for Network Security

Network programmability, and in particular data plane programming, is considered a useful platform to explore defending a network against Distributed Denial of Service (DDoS) attacks.

4.2.1 eBPF/XDP Anti DDoS Application in SURF

SURF, the Dutch NREN, has been investigating the usage of eBPF/XDP to filter layer 7 Distributed Denial of Service attacks [SURF-1], [SURF-2], specifically Water Torture attacks. Such attacks deplete the processing capacity of victim authoritative DNS servers by forwarding massive amounts of invalid DNS requests about names that are not present in the server zones. By randomly generating domain name prefixes, the attackers bypass the DNS caches of the intermediary recursive DNS Servers, thus all the attack traffic is forwarded to victims, and the attack impact is maximised.

According to the experimentation in [100], eBPF/XDP is considered an appropriate remedy to Water Torture attacks. Specifically, in-kernel Bloom Filters are employed to effectively map the zone contents of the available authoritative DNS servers and rapidly differentiate between legitimate and malicious DNS requests at the NIC driver level. Notably, Bloom Filters are probabilistic data structures, which store elements hashed (instead of their cleartext form), thus supporting time- and space-efficient membership lookups.

4.3 White Boxes Deployment in NRENs

The advantages of white-box switches - which allow greater flexibility, lower costs, and more control over network innovation by decoupling hardware and software - have been recognised by several European NRENs. During the GN4-3 project, which was completed at the end of 2022, the integration of white box equipment at different NREN sites was promoted. These white boxes share some common characteristics, such as the use of open-source Network Operating Systems and typically programmable chipset-based forwarding equipment.

It was then demonstrated in several NRENs that white box-based solutions could provide comparable performance to proprietary solutions by carrying out a performance evaluation in a white box-based MPLS Provider router use case [101]. The evaluation was performed on Edgecore AS5912-54X-O-AC-F, Dell EMC S4248FBL-ON, Arista 7280SR, and Juniper PTX white boxes running the OcNOS system.

As for specific implementations, RENATER, the French NREN, implemented a Global Internet Exchange point (GIX) based on white box hardware and a commercial network operating system, OcNOS [102]. The white box replaces RENATER’s SFINX Brocade MLX switches. GRNET, the Greek NREN, also used a white box switch/router to implement a production data centre network in Greece [103]. The network devices selected to implement this design were the Edgecore AS7712-32X (equipped with the chipset forwarding Broadcom Tomahawk) for the spines and the Edgecore AS5812-54T (equipped with the chipset forwarding Broadcom Trident II+) for the leaves. They used Cumulus Linux, which was already acquired by NVIDIA, as their main NOS. Funet, the Finnish NREN, used Edgecore AS7315-27X devices and the ADVA Ensemble Activator NOS to implement their customer premises equipment edge router [104].

4.4 CERN IPv6 Packet Marking

CERN is the European Organisation for Nuclear Research and operates the largest particle physics facility in the world. Its facility hosts instruments equipped with sensors able to generate a huge amount of traffic on a per-experiment basis. Most of this traffic is IPv6 traffic, and it has been determined that each experiment should have its traffic marked so that it can be identified and processed accordingly across network domains. A prototype for this function was created and tested successfully using RARE/freeRtr in GP4L. Packet marking demos via GP4L were presented during SC23 and SC24 by CERN and Northwestern University.

4.5 UFES - PolKA & MPolKA

PolKA – Polynomial Key-based Architecture for Source Routing – and its multipath form M-PolKA, is a protocol developed by UFES (Universidade Federal do Espírito Santo, Brasil) that "proposes a novel RNS-based SR scheme that explores binary polynomial arithmetic using a Galois field (GF)" [105]. It describes an entry-less forwarding paradigm resulting from a mathematical computation between a PolKA Route ID in the packet header entering the system and the Node ID receiving the packet, determining the egress port ID. PolKA received the 2021 Google Research Scholar Award.

The PolKA protocol was initially validated using the GP4L lab as the first global production platform, which then led to its implementation in the RARE/freeRtr software stack. This was officially recognised in UFES's publications and is an illustration of how RARE/GP4L [106] can help the research community.

5 RARE and GP4L

RARE, the first open-source routing operating system, and GP4L, the platform where RARE was first implemented over operational production networks, were first conceived in 2019 by the Network Technologies and Services Development work package of the GN4-3 project. Their development has continued in subsequent GÉANT projects, first in GN5-1 and now in GN5-2.

5.1 RARE – Router for Academia, Research and Education

The RARE acronym stands for 'Router for Academia, Research and Education'. As its name suggests, it provides a routing platform specifically tailored for research and education use cases. RARE shares its acronym with the «Réseaux Associés pour la Recherche Européenne», created in 1986, and which was the very first association of organisations that today represent the GÉANT community. The name RARE was therefore intentionally chosen to preserve the memory of that first association within the community.

Starting from a blank canvas, the project began its work by identifying the building blocks of a routing and switching platform: a data plane and a control plane, as well as a programming language and a programmable interface that would act as the glue between the two.

5.1.1 The RARE Control Plane – freeRtr

For its control plane, which decides how packets should be conveyed globally in the entire routing domain, several open-source candidates were initially considered, such as VyOS, BIRD, OpenBGPD, FRR and SONIC and freeRtr. Ultimately, freeRtr [107] was chosen as it originated from the NREN community and offered a rich palette of features, including BGP and MPLS support. At the time, freeRtr was also used by KIFÜ, the Hungarian NREN, as a route reflector in their core network. To emphasise the use of freeRtr for its control plane, and similarly to the GNU/Linux project, RARE software was also referred to as RARE/freeRtr.

5.1.2 Programming Language and Data Planes for RARE

Considering different domain-specific languages such as $P4_{14}$ (subsequently $P4_{16}$) and NPL that provide a simplified way to program data plane functions for multiple different vendor targets, the RARE team decided to use the P4 domain-specific language as at the time it was able to work with different targets that were easily available for the group to work on.

The first chosen target was a virtual switch from P4.org named Behavioral Model version 2 – bmv2 [108]. Being a virtual target, its purpose was to be used for developing, testing and debugging P4 data planes and control plane software, rather than to be able to serve as a production-grade software switch.

The objectives of the team were to look for a production-grade solution that could be used in production NREN networks. **Tofino ASIC** was chosen as a target as it could provide higher throughput and packet forwarding speeds and lower jitter and latency.

5.1.2.1 RARE/freeRtr and DPDK, XDP

As a part of its goal to explore different platforms for network programmability, the RARE project team looked for affordable data planes other than P4-based ones. Linux kernel-bypass mechanisms such as DPDK and XDP are examples of affordable and ubiquitous technology with the potential to reduce the digital divide. At first, it was planned to use P4ELTE/T4P4S software to translate RARE P4 programs to DPDK primitives. However, this solution was abandoned when some critical bugs were encountered, so it was decided as an alternative to create a layer that emulated P4 RARE Match Action Unit program behaviour. Instead of yielding P4 primitives, the layer would yield wrapper primitives from the packet forwarding library such as DPDK, XDP and libpcap functions.

5.1.2.2 Unexplored Data Planes (Spectrum, XGS, DNX, Teralynx, FPGA)

The next target that was explored was switchdev, which allows a standard Linux to use a specific driver model to integrate networking hardware into the Linux server environment [109]. Such a driver would then allow the Linux kernel to expose hardware networking ports as standard Linux interfaces.

For the team, using switchdev would have opened the door to Mellanox/NVIDIA Spectrum ASIC hardware offload capability for freeRtr. The use cases in focus were related to MPLS, however, this work was abandoned due to some limitations of switchdev that were recognised at the time.

Several initiatives to adapt the RARE P4 code to some platforms were closed down due to limited support and documentation from vendors and/or hardware and licensing availability, such as those involving Broadcom NPL running on top of the XGS & DNX ASIC families, Marvell Teralynx, Prestera and FPGAs.

5.1.3 RARE Use Cases

From its beginnings, the intention of the RARE team was to create software to be run in a production environment that is functional, stable and reliable. The use cases in scope were those that are typical for national research and education networks as well as for small institutions such as schools and campuses, taking also into consideration those affected by low-income and the digital divide.

5.1.3.1 RARE for Data Center Interconnect (DCI) Internet

RARE/freeRtr provides backend connectivity via MPLS and BGP labelled unicast to the Regional Computing Mesocenter in the north-east of France managed by CRIANN [110], which also runs the Normandy Regional Network. Initially, it was planned to use RARE/freeRtr on Wedge100BF-32 but, due to Intel announcing that it was dropping Tofino support, CRIANN decided to test and use RARE/freeRtr with DPDK. This connectivity is realised as an overlay on top of the RENATER core backbone network and seamlessly connects other mesocenters in Brittany in the far western part of France and the Lorraine region in the far east of the country.

5.1.3.2 RARE as an Open Source Anti-DDoS Solution

The relevance and criticality of DDoS continue to be on the rise due to increased DDoS frequency and DDoS traffic rates reaching another level of magnitude (3,8 Tbps). This means that today's increased requirements for anti-DDoS measures are typically only met by expensive, commercial software products that are often only suited to company network traffic profiles and do not target heterogeneous, open NREN network traffic. This is why, in the context of the NREN/GÉANT community, open-source software products for anti-DDoS solutions (DDoS detection and mitigation), namely NeMo [111] and Firewall-on-Demand (FoD) [112], which complement each other, are actively being developed and operated.

In line with the scope and vision of RARE/freeRtr, therefore, the team conducted investigations and tests on how to most efficiently combine RARE/freeRtr and these two GÉANT Anti-DDoS software products. The aim was to make these solutions usable with RARE/freeRtr in particular and easier to use for NRENs and other research organisations in general. These efforts were successful and RARE/freerRtr can now be leveraged to yield auto-installable, integrated demonstration and testing container solutions for both products. In the case of FoD, this was used as a basis for a virtual anti-DDoS demonstration [113] combining RARE/freeRtr, FoD and containerlab [114].

5.1.3.3 RARE 5G UPF Implementation

5G technology is designed for heterogeneous scenarios, improving QoS and performance where huge numbers of users are connected, or high data rates are required. 5G is also designed for private networks, also known as Non-Public Networks (NPN). This technology helps private small-to-large corporations to deploy and manage their own networks, as they currently do with other standards like IEEE 802.11, but with noticeable improvements in QoS and performance. However, a significant drawback of 5G remains the dependency on vendor-specific equipment, which forces network administrators to stick to specific and usually expensive hardware devices.

The GÉANT project's WP6 Task 1 group has proposed the implementation of a 5G User Plane Function (UPF) in RARE/freertr, an open-source multifunctional router. The UPF routes user traffic between the Radio Access Network (RAN) and external networks, enhancing performance and giving network operators full control over user traffic. This approach separates the UPF from the 5G core implementation and leverages general-purpose hardware, eliminating the need for expensive and specific equipment.

The proposal aims to serve as a baseline for deploying a more complex entity that could provide additional features defined in the 3GPP architecture. Future work could involve deploying several UPFs to leverage network slicing and provide appropriate QoS for different communications. Such work could be particularly relevant towards promoting open-source developments and reducing dependency on vendor-specific hardware.

5.1.3.4 RARE/freeRtr Network Management Innovation

The programmable RARE/freeRtr platform can be used for network protocol development. However, RARE/freeRtr programmability is not restricted to pure control plane functionality development and the RARE team realised that, having the possibility to run multiple IGPs or BGP, they can provide network operators with a convenient way to monitor RARE/freeRtr nodes. In this context:

- A Prometheus agent was created powered by a sensor concept. In a nutshell, a sensor can be seen as a CLI parser that is able to return key/value pairs as metrics ready to be scraped by a Prometheus server.
- Streaming telemetry was also implemented. Coupled with InfluxDb and Telegraf in nmaas, the network operator can receive granular information from the network element.
- BGP-LS was also implemented so that a northbound server can retrieve IGP topology information from an external domain. Various tests have been undertaken with HEAnet and GP4L.
- A set of RARE/freeRtr Grafana dashboards have been implemented to provide a full suite of network monitoring dashboards [64].
- An innovative alarm system that monitors IGP topology was also developed using icinga2 and Nagios NRPE transport. This was presented during the 17th SIG-NOC in Paris [115].

5.1.4 RARE Packaging

During the initial phase of the project, creating a P4 environment with bmv2 was an hour-long compilation process. It then became obvious that creating packages would help project team members to quickly start P4 development and the team has now started to prepare RARE software packages for different platforms.

P4 and RARE/freeRtr Ubuntu and Debian packages were the first to be created. These packages helped the RARE and GP4L project participants to effectively start P4 development but were not popular with the P4 community outside of the RARE project as they were not official p4lang packages. External developers were reluctant to install non-certified packages signed by an unknown organisation. In addition, not having a close interaction with the p4.org project made maintenance of the packages extremely difficult as it was not possible to anticipate any dependency changes. Nevertheless, these packages contributed to the elaboration of a RARE/freeRtr ONIE image via a Jenkins CI/CD pipeline.

Additionally, the Nix package manager and language were used to create Nix SDE packages, which were also officially accepted by Intel. All the Nix packages can be integrated into an ONIE image ready to be installed in one line. Nix’s inherent properties make sure that the installation is completely reproducible. The time to deploy a P4 node in GP4L was reduced from a few hours, sometimes days, to about five minutes in a predictable way.

5.1.5 RARE Documentation

More information and documentation on RARE are available at [116].

5.2 GP4L

During RARE/freeRtr code development it was necessary to test the code in a representative environment. For this purpose, the GÉANT project first deployed four P4 nodes in Amsterdam, Frankfurt, Budapest and Poznan, followed a few years later by five nodes in Geneva. This infrastructure, which is maintained and operated by the Network Development work package of the GÉANT project, was initially meant to be used to test the code developed by the RARE team. However, unexpectedly the RARE team received multiple requests from research organisations wanting to connect their P4 switch to this core infrastructure in Europe, thus leading to the creation of the GÉANT P4 lab, also known as GP4L.

Strengthened by this expansion of its European footprint, GP4L also began to attract the attention of research organisations from other countries, such as the USA, Brazil, Japan and Korea. This eventually led to the GÉANT P4 Lab’s expanding into what became the Global P4 lab, a worldwide networking platform available for researchers to experiment with various networking ideas. The map in Figure 4.1 shows the locations at which nodes with RARE deployments are in place (with over 30 nodes being deployed overall) at the time of writing.

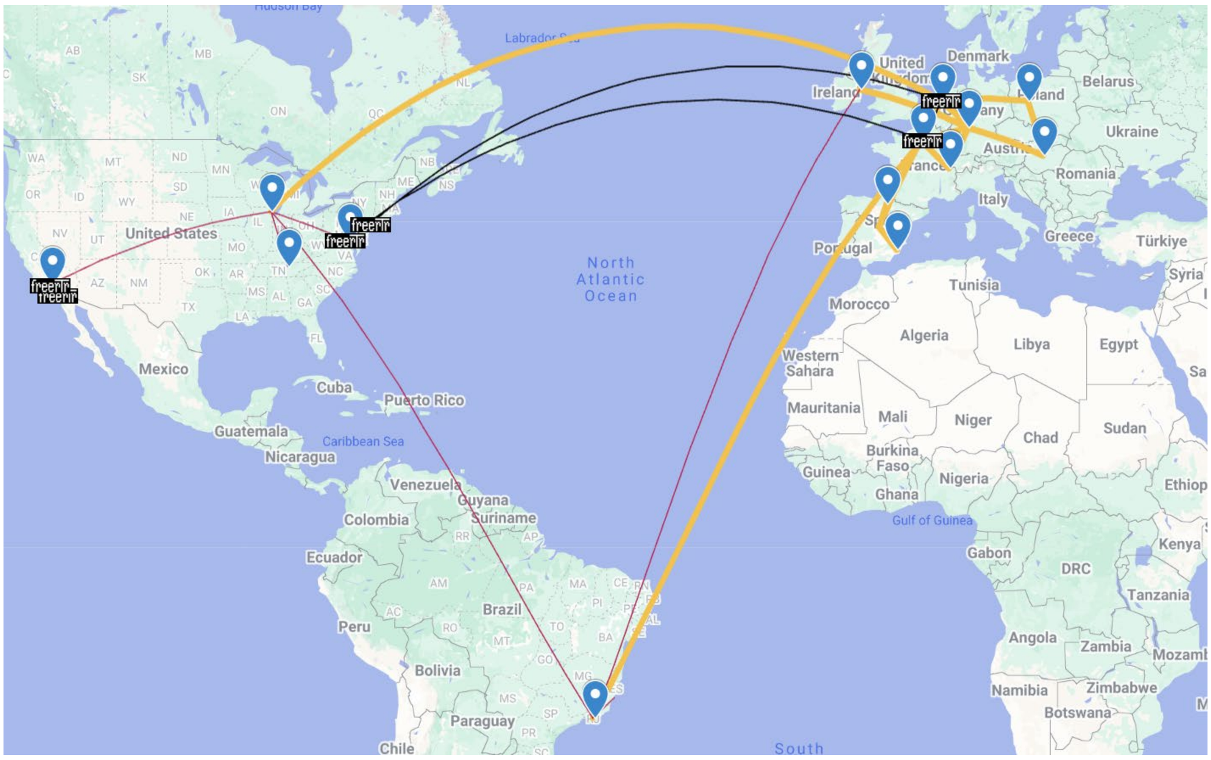

Figure 5.1: GP4L with RARE deployments [April 2025]

Today, GP4L is an experimental infrastructure distributed across the globe which is mainly dedicated to (i) testing and validating code developed within the RARE project, (ii) enabling researchers to develop applications based on programmable data planes using the GP4L platform, and (iii) supporting geographically distributed networking experiments.

5.2.1 RARE and GP4L Experiments

Since their origins, RARE and GP4L have been relied on by the R&E community to enable evaluation and testing of network programmability concepts and solutions. Programmable switches connected in the Global P4 Lab - Global platform for Labs - with or without the RARE/freeRtr software stack, continue to be used by network developers, engineers and researchers in R&E organisations. The following sections provide some examples of how RARE and GP4L have been used by the community to test and validate different network technologies and solutions using the programmable network software and platform.

5.2.1.1 RARE and GP4L SuperComputing Network Research Exhibitions

Network Research Exhibitions that take place on a yearly basis during the SuperComputing (SC) conference have provided a good opportunity for project demonstrations, such as:

- “Programmable Networking with P4, GEANT RARE/freeRtr and SONIC/PINS”, SC22 [117]
- “Global P4 Lab”, SC23 [118]
- “PolKA routing approach to support traffic engineering for data-intensive science”, SC23 [119]
- “PolKA routing approach to support traffic steering for data-intensive science”, SC24 [120]

Each of these presented examples where GP4L has been used to test software solutions developed and deployed in the RARE operating system and then heavily tested on the global P4-based platform.

5.2.1.2 Standardisation Activities and Collaboration with Vendors

RARE/freeRtr has been used and developed further to explore and support development of new IETF drafts and protocols. In some cases, further experiments have followed using the GP4L platform for deeper validation and testing. Some examples include:

- **AMT & Unicast 2 Multicast implementation** – The RARE/freeRtr project implemented two complementary components jointly with Juniper: Automatic Multicast Tunnelling or AMT, and a Unicast to Multicast server. The basic idea behind AMT is to provide a mechanism for a client host to receive traffic of interest via a tunnel laid down on top of a non-multicast network with a multicast-aware gateway called AMT. The unicast to multicast server that was created and implemented in RARE/freeRtr is a component that converts a unicast traffic stream to a multicast stream.
- **TreeDN** - By combining the two components of the AMT with a multicast network, a content distribution tree can be created, as described in detail in RFC 9706 – TreeDN: Tree-Based Content Delivery Network (CDN) for Live Streaming to Mass Audiences. This document also explicitly mentions RARE: *"The RARE network is a global testbed interconnecting several National Research and Education Networks (NRENs) via routers running BIER. AMT relays are deployed to deliver multicast traffic from sources on the RARE network to receivers on unicast-only networks across the Internet. Details of the RARE network are described in [BIER-AMT-Deployment]."*
- **EANTC MPLS BGP-CT interoperability** – The RARE/freeRtr team also contributed to the standardisation effort for BGP-CT in [57] draft-ietf-idr-bgp-ct-35 - BGP Classful Transport Planes and draft-ietf-idr-bgp-car-11 - BGP Color-Aware Routing (CAR) and conducted practical BGP-CT interworking with Juniper during EANTC's multi-vendor MPLS and SDN interoperability test event.
- The RARE/freeRtr team implemented the **Bit Index Explicit Replication** (**BIER) protocol** [121] on the BMv2, Tofino and DPDK data plane., This was presented at IETF 110 as part of the BIER session.

The RARE/freeRtr team participated in an NVIDIA hackathon in March 2022, finishing in 2nd place. The abstract of the hackathon from NVIDIA's website [55] reads: "*FreeRTR Router is a Swiss army knife meant to be used as a primary router but can also be used as a specific appliance. The team evaluated accelerated DPDK and DOCA FLOW and enabled the routing functionality with multiple routing protocols on the DPU leveraging the large and programmable flow tables. As part of their planned innovation, the team also evaluated added services by linking DOCA libraries to provide additional functionality including Firewall, RegEx scanning, MACsec encryption, and DPI engine at line rate. This allowed Team Rare to create one control plane to rule all data planes.*"

6 Conclusions

Over the last decade, interest in network programmability has grown both within the commercial and vendor environment and the R&E community, and a wide range of such products are now on offer. These include hardware platforms of devices, Application-Specific Integrated Circuits, Network Interface Cards and Network Processing Units, software solutions for operating systems, development environments, languages and models applied to different scenarios and use cases. This document has provided an overview of all these elements and examined in detail both the completed and ongoing activities taking place in these areas in the research and education community.

In particular, the Router for Academia, Research and Education - RARE - operating system and its deployment on the Global Platform for Labs - GP4L are spotlighted as ongoing activities of the Network Development work package of the GÉANT project. The history, development endeavours and reasoning behind them are discussed and illustrated through their operational, production or production-grade use cases. In parallel with its ongoing work, the RARE team is continuously exploring and evaluating possible directions for future work. The team's direction of work is usually influenced by changes in the market, as new control plane and data plane hardware and software solutions, as well as new programming languages and models become available.

The activities of NRENs in Europe and globally as well as the team members' areas of knowledge and interest also have a bearing on which areas of work are considered for future development. Presently these include but are not limited to:

- Using XDP to develop custom BGP extensions and enable efficient network traffic filtering based on fine-grained attack signatures [8] derived from the packet headers and payload. Unlike hardware data planes (e.g. P4), XDP facilitates the adoption of novel experimental features across organisations with diverse network devices. XDP programs mainly depend on device drivers, thus can run on different platforms with minimum software modifications.
- Employing HackP4 or similar toolsets to verify the safety of the P4 programs that have already been developed. Since the RARE platform supports various data planes, the HackP4 toolset principles could be extended and applied to the vulnerability assessment of eBPF/XDP and DPDK programs.
- Employing LLMs, specifically ChatGPT, to automatically generate P4 or eBPF/XDP code. Such code could address various interesting use cases: (1) DDoS attack detection and mitigation, (2) load balancing, and (3) traffic engineering.

The examples presented in this paper highlight how network engineering today relies heavily on network programmability, with more and more solutions from both vendors and network operators aiming to simplify and speed up its use, thus helping further research and development of network infrastructure and services.

Glossary

A

ABI	Application binary interface
AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit

B

bmv2	Behavioral model version 2
BRI	Barefoot Runtime Interface
BSP	Board Support Package

C

CDN	Content Delivery Network
CPU	Central Processing Unit

D

DCI	Data Center Interconnect
DCN	Data Centre Network
DDoS	Distributed Denial of Service
DPDK	Data Plane Development Kit
DPP	Data Plane Programmability
DPU	Data Processing Unit

E

eBPF	extended Berkeley Packet Filter
EoL	End of Life

F

FBOSS	Facebook Open Switching System
FoD	Firewall-on-Demand
FPGA	Field-Programmable Gate Array

G

GIX	Global Internet Exchange
GN4-3	GÉANT Network 4 Phase 3, a project part-funded by the EC's Horizon 2020 programme under Specific Grant Agreement No. 856726
GN5-1	GÉANT Network 5, Phase 1, a project funded by the European Union's Horizon Europe research and innovation programme under Grant Agreement No. 101100680 and one of the projects implementing the actions defined in the GN5-FPA

GN5-2	GÉANT Network 5, Phase 2, a project funded by the European Union’s Horizon Europe research and innovation programme under Grant Agreement No. 101194278 and one of the projects implementing the actions defined in the GN5-FPA
GP4L	Global Platform for Labs
GREN	Global Research and Education Network

H

HFA	History-based Finite Automaton

I

IDS	Intrusion Detection System
INT	In-band network telemetry
IPDK	Infrastructure Programmer Development Kit
IPS	Intrusion Prevention System
IR	Intermediate Representation

L

LLM	Large Language Models

M

MA	Match Action
MAT	Match-Action Table
ML	Machine Learning

N

NeMo	Network Monitoring
NFV	Network Function Virtualisation
NIC	Network Interface Card
NOS	Network Operating System
NPL	Network Programming Language
NPN	Non-Public Networks
NPU	Network Processing Unit
NREN	National Research and Education Network

O

OCP	Open Compute Project

P

PISA	Protocol-Independent Switch Architecture
PMD	Poll-Mode Driver
PSA	Portable Switch Architecture

Q

QoS	Quality of Service

R

RAN	Radio Access Network

RARE	Router for Academia, Research, and Education

S

SAI	Switch Abstraction Interface
SDE	Software Development Environment
SDN	Software-Defined Networking
SIG-NOC	Special Interest Group – Network Operations Centres
SONiC	Software for Open Networking in the Cloud
SRAM	Static Random-Access Memory
STP	Spanning Tree Protocol
SVE2	Scalable Vector Extension V2

T

Tbps	Terabits per second
TCAM	Ternary Content-Addressable Memory

U

UFES	Universidade Federal do Espírito Santo, Brasil
UPF	User Plane Function
URLLC	Ultra-reliable low-latency communication

V

VHSIC	2.5.4 Very High Speed Integrated Circuit
VNF	Virtual Network Functions
VPP	Vector Packet Processing

W

WAN	Wide Area Network

X

XDP	eXpress Data Path
XR	Extended Reality

References

[1] Marinos DImolianis (NTUA), David Schmitz (LRZ), Henrik Wessing (DTU), Pavle Vuletić (UoB), White Paper: High-Performance Flow Monitoring Using Programmable Network Interface Cards, Grant Agreement No.: 856726, GN4-3-22-N3R07E WP6 Task 3 https://resources.geant.org/wp-content/uploads/2023/02/GN4-3_White-Paper_High-Performance-Flow-Monitoring-Using-Programmable-NICs.pdf

[2] Poutievski, L., Mashayekhi, O., Ong, J., Singh, A., et al (2022, August). Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. In Proceedings of the ACM SIGCOMM 2022 Conference (pp. 66-85)

[3] Jupiter evolving: Reflecting on Google's data center network transformation, August 2022. https://cloud.google.com/blog/topics/systems/the-evolution-of-googles-jupiter-data-center-network

[4] RARE (Router for Academia, Research & Education), GÉANT https://docs.rare.geant.org

[5] OpenDaylight – The Linux Foundation https://www.opendaylight.org/

[6] Open Network Operating System (ONOS) - Open Networking Foundation (ONF) https://opennetworking.org/onos/

[7] Ryu-SDN https://ryu-sdn.org/

[8] A component-based software defined networking framework in OpenStack. https://github.com/openstack/os-ken

[9] Cisco ACI (Application Centric Infrastructure) https://www.cisco.com/site/mx/es/products/networking/cloud-networking/application-centric-infrastructure/index.html

[10] TerafFlowSDN – ETSI https://www.teraflow-h2020.eu/

[11] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. 38, 2 (April 2008), 69–74

[12] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, https://datatracker.ietf.org/doc/html/rfc6241

[13] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, https://www.rfc-editor.org/info/rfc4271

[14] B. Pfaff, "The Open vSwitch Database Management Protocol", RFC 7047, DOI 10.17487/RFC7047, December 2013, https://datatracker.ietf.org/doc/html/rfc7047

[15] VyOS https://vyos.io/

[16] OpenBGPD https://openbgpd.org/

[17] BIRD https://bird.network.cz/

[18] ExaBGP https://github.com/Exa-Networks/exabgp

[19] GitHub – osrg / gobgp https://github.com/osrg/gobgp

[20] FRRouting https://frrouting.org/

[21] GitHub – holo-routing / holo https://github.com/holo-routing/holo

[22] Hauser, F., Häberle, M., Merling, D., Lindner, S., Gurevich, et al. (2023). A survey on data plane programming with p4: Fundamentals, advances, and applied research. *Journal of Network and Computer Applications*, 212, 103561

[23] Franco, D., Higuero, M., Sanz, A., Unzilla, J., & Huarte, M. (2024). vFFR: A Very Fast Failure Recovery Strategy Implemented in Devices With Programmable Data Plane. IEEE Open Journal of the Communications Society

[24] Zaballa, E. O., Franco, D., Zhou, Z., & Berger, M. S. (2020, February). P4Knocking: Offloading host-based firewall functionalities to the network. In 2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN) (pp. 7-12). IEEE

[25] P4 Ecosystem https://p4.org/ecosystem/?_product_category=hardware

[26] Intel, Intel Tofino. P4-programmable Ethernet switch ASIC that delivers better performance at lower power, 2021, https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html

[27] Broadcom Trident3-X7 / BCM56870 Series https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56870-series

[28] Broadcom Tomahawk 5 https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm78900-series

[29] Broadcom Trident 5 https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm78800

[30] Marvel Teralynx 10 https://www.marvell.com/content/dam/marvell/en/public-collateral/switching/marvell-teralynx-10-data-center-ethernet-switch-product-brief.pdf

[31] Marvell Prestera 98DX73xx https://www.marvell.com/content/dam/marvell/en/public-collateral/switching/marvell-switching-prestera-98dx73xx-product-brief.pdf

[32] RENATER official website https://www.renater.fr/?lang=en/

[33] NORDUnet Open eXchange (NOX) – https://nordu.net/nordunet-open-exchange-nox/

[34] Announcing Next-Generation P4-Programmable Datacenter Switching https://www.linkedin.com/pulse/announcing-next-generation-p4-programmable-datacenter-switching-skzoc?trk=public_post

[35] Intel Tofino – see [26]

[36] 'Intel is halting development of the networking chip it got from Barefoot Networks' https://www.bizjournals.com/sanjose/news/2023/01/26/intel-halts-development-of-tofino-switch-chips.html

[37] 2024 P4 Workshop https://p4.org/p4_events/2024-p4-workshop-postevent/

[38] Deb Chatterjee, 'The Past, Present and Future of P4', in P4 Workshop 2024 https://p4.org/wp-content/uploads/2024/10/2-2024-P4-Workshop-past-present-and-future-of-p4.pdf

[39] P4 language repository https://github.com/p4lang/p4c

[40] AMD Pensando https://www.amd.com/es/products/accelerators/pensando.html

[41] Netronome Agilio CX SmartNICs. https://netronome.com/agilio-smartnics/

[42] GN4-3 White Paper *In-Band Network Telemetry Tests in NREN Networks,* GN4-3 WP6 Task 1 (Grant No. 856726) https://resources.geant.org/wp-content/uploads/2022/02/GN4-3_White-Paper_In-Band-Network-Telemetry.pdf

[43] Liu, Z., Mah, B., Kumar, Y., Guok, C., & Cziva, R. (2020). Programmable per-packet network telemetry: From wire to kafka at scale. In *Proceedings of the 2021 on Systems and Network Telemetry and Analytics* (pp. 33-36)

[44] AMD Vitis Networking P4 environment https://www.xilinx.com/products/intellectual-property/ef-di-vitisnetp4.html

[45] AMD Zynq 7000 SoCs https://www.amd.com/es/products/adaptive-socs-and-fpgas/soc/zynq-7000.html

[46] AMD Virtex 7 Boards, Kits, and Modules https://www.xilinx.com/products/boards-and-kits/device-family/nav-virtex-7.html

[47] AMD Versal AI Core Series https://www.amd.com/en/products/adaptive-socs-and-fpgas/versal/ai-core-series.html

[48] DPU vs Smart NIC https://www.servethehome.com/dpu-vs-smartnic-sth-nic-continuum-framework-for-discussing-nic-types/

[49] Cavium Octeon II https://datasheet.octopart.com/CN6335-1300BG900-AAP-Y-G-Cavium-Networks-datasheet-25352747.pdf

[50] GitHub – osrg / gobgp https://github.com/osrg/gobgp

[51] Marvell Octeon 10 https://www.marvell.com/content/dam/marvell/en/company/media-kit/octeon-10/marvell-octeon-10-media-deck.pdf

[52] Marvel Octeon 10 block diagram https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-octeon-10-dpu-platform-product-brief.pdf

[53] CloudSwitch Asterfusion ET2500 https://cloudswit.ch/wp-content/uploads/2024/08/Datasheet-ET2500-Open-Intelligent-Gateway.pdf

[54] SONiC https://sonicfoundation.dev/

[55] NVIDIA Cumulus Linux https://www.nvidia.com/es-es/networking/ethernet-switching/cumulus-linux/

[56] RtBricks https://www.rtbrick.com/

[57] IP Infusion https://www.ipinfusion.com/

[58] Arrcus https://arrcus.com/

[59] ‘Cumulus first to power Facebook’s Minipack’ https://datacentrereview.com/2019/03/cumulus-first-to-power-facebook-s-next-gen-open-modular-platform/

[60] Oxide https://oxide.computer/

[61] Linux Foundation (2015), *Data Plane Development Kit (DPDK)*, http://www.dpdk.org

[62] Express Data Path https://en.wikipedia.org/wiki/Express_Data_Path

[63] Cilium, *BPF and XDP Reference Guide*, https://docs.cilium.io/en/stable/bpf/

[64] XDP https://docs.cilium.io/en/stable/bpf/progtypes/#xdp

[65] P4 community, Behavioral Model version 2 (bmv2). The reference P4 software switch, 2021, GitHub repository https://github.com/p4lang/behavioral-model

[66] Github - p4lang / open-p4studio https://github.com/p4lang/open-p4studio

[67] See [66]

[68] P. Bosshart, D. Daly, G. Gibb, M. Izzard, et al., ‘P4: Programming protocol-independent packet processors’, *SIGCOMM Comput. Commun*. Rev. 44 (3) (2014) pp. 87–95

[69] F. Hauser, M. Häberle, D. Merling, S. Lindner et al. (2023). A survey on data plane programming with p4: Fundamentals, advances, and applied research. Journal of Network and Computer Applications, 212, 103561

[70] See [69]

[71] NPL – Network Programming Language Specification v1.3 https://github.com/nplang/NPL-Spec/

[72] See [71]

[73] See [71]

[74] ISO/IEC. (2014). *ISO International Standard ISO/IEC 14882:2014(E) – Programming Language C++* [Working draft], Geneva, Switzerland: International Organization for Standardization (ISO), https://isocpp.org/std/the-standard

[75] Netronome Agilio CX SmartNICs https://netronome.com/agilio-smartnics/

[76] IEEE 1076-2019 https://standards.ieee.org/ieee/61691-1-1/11428/1076/5179/

[77] ONE Summit 2024: P4-Based Automated Reasoning https://p4.org/wp-content/uploads/2024/05/2024-ONE-Summit-P4-Based-Automated-Reasoning.pdf

[78] FBOSS Experience of Migrating Massive Scale Networking Systems to SAI https://www.youtube.com/watch?v=Yq5_g5H7vbg

[79] Revolutionizing Data Center Networks: Alibaba's SONiC Journey https://sonicfoundation.dev/revolutionizing-data-center-networks-alibabas-sonic-journey/#:~:text=In%20the%20quest%20for%20a,for%20Alibaba's%20expansive%20network%20infrastructure

[80] The Next Horizon: What Lies Beneath the 6 Pillars of 6G?, Huawei. https://blog.huawei.com/2023/01/03/next-horizon-beneath-6-pillars-6g/

[81] What to expect from 6G: Here are nine important takeaways from early global research, Ericsson https://www.ericsson.com/en/blog/2023/2/6g-early-research-global-takeaways

[82] Shaping the future of 6G, McKinsey & Company https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/shaping-the-future-of-6g

[83] See [80]

[84] See [81]

[85] CloudSwitch Asterfusion ET2500 https://cloudswit.ch/wp-content/uploads/2024/08/Datasheet-ET2500-Open-Intelligent-Gateway.pdf

[86] F. Hauser, M. Häberle, D. Merling, S. Lindner et al. (2023), 'A survey on data plane programming with p4: Fundamentals, advances, and applied research', *Journal of Network and Computer Applications*, 212, 103561

[87] Mihai-Valentin Dumitru, Dragos Dumitrescu and Costin Raiciu, "Towards Automatic Exploitation of Programming Networks", in the Proceedings of IEEE NetSoft 2024

[88] Snort https://www.snort.org/

[89] RFC 7938 https://datatracker.ietf.org/doc/html/rfc7938

[90] Poutievski, L., Mashayekhi, O., Ong, J., Singh, A., et al. (2022, August). Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. In Proceedings of the ACM SIGCOMM 2022 Conference (pp. 66-85)

[91] Jupiter evolving: Reflecting on Google's data center network transformation, August 2022. https://cloud.google.com/blog/topics/systems/the-evolution-of-googles-jupiter-data-center-network

[92] 2024 P4 Workshop https://p4.org/p4_events/2024-p4-workshop-postevent/

[93] Steffen Smolka, Jonathan DiLorenzo, Ali Kheradmand, 'P4-Based Automated Reasoning (P4-BAR) for the (Networking) Masses', P4 Workshop 2024 https://p4.org/wp-content/uploads/2024/10/17-2024-P4-Workshop-P4-Based-Automated-Reasoning-for-the-Networking-Masses.pdf

[94] Paglierani, P. (2015, May), 'High performance computing and network function virtualization: A major challenge towards network programmability', in *2015 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pp. 137-141, IEEE

[95] Paolucci, F., Cugini, F., Castoldi, P., & Osiński, T. (2021), 'Enhancing 5G SDN/NFV edge with P4 data plane programmability', *IEEE Network*, *35*(3), pp. 154-160

[96] Franco, D., Atutxa, A., Sasiain, J., Ollora, et al. (2022). 'Towards integrating hardware Data Plane acceleration in Network Functions Virtualization', *arXiv preprint arXiv:2203.10920*

[97] GN4-3 White Paper *High-Performance Flow Monitoring Using Programmable Network Interface Cards*, 2023 https://resources.geant.org/wp-content/uploads/2023/02/GN4-3_White-Paper_High-Performance-Flow-Monitoring-Using-Programmable-NICs.pdf

[98] GN4-3 White Paper *In-Band Network Telemetry Tests in NREN Networks,* GN4-3 WP6 Task 1 (Grant No. 856726) https://resources.geant.org/wp-content/uploads/2022/02/GN4-3_White-Paper_In-Band-Network-Telemetry.pdf

[99] See [98]

[100] Tutan F. Wijnsma, Biemolt W. (2024), *Mitigating Layer 7-based DDoS Attacks in the SURF Network*, Research Report delivered for University of Amsterdam Courses

[101] GN4-3 White Paper *White Box Performance Testing and Evaluation – Recommendations for NRENs* https://resources.geant.org/wp-content/uploads/2022/02/GN4-3_White-Paper_White-Box-Testing-and-Evaluation.pdf

[102] GN4-3 White Paper *GIX Implementation Based on White Box* https://resources.geant.org/wp-content/uploads/2022/02/GN4-3_White-Paper_White_Box_GIX.pdf

[103] GN4-3 White Paper *White Box: GRNET Data Centre Use Case* https://resources.geant.org/wp-content/uploads/2022/02/GN4-3_White-Paper_White-Box-GRNET-Data-Centre-Use-Case.pdf

[104] GN4-3 White Paper *Funet CPE – White Box Edge Router* https://resources.geant.org/wp-content/uploads/2022/02/GN4-3_White-Paper_White_Box_Funet-CPE.pdf

[105] *PolKA: Polynomial Key-based Architecture for Source Routing in Network Fabrics* https://ieeexplore.ieee.org/document/9165501

[106] C. Dominicini et al., 'Deploying PolKA Source Routing in P4 Switches' (Invited Paper)," 2021 International Conference on Optical Network Design and Modeling (ONDM), Gothenburg, Sweden, 2021, pp. 1-3, https://ieeexplore.ieee.org/document/9492363

[107] freeRtr https://github.com/mc36/freeRtr

[108] bmv2 https://github.com/p4lang/behavioral-model

[109] Switchdev https://www.kernel.org/doc/html/v5.9/networking/switchdev.html

[110] CRIANN https://www.criann.fr/

[111] NeMo https://security.geant.org/nemo-ddos-software/

[112] FoD https://security.geant.org/firewall-on-demand/

[113] 'Relying on RARE for DDoS Attack Protection' https://www.youtube.com/watch?v=0rOTo1rh18w

[114] Containerlab https://containerlab.dev/

[115] RARE and GP4L https://zenodo.org/records/7842459

[116] RARE documentation https://docs.rare.geant.org/

[117] https://sc22.supercomputing.org/app/uploads/2022/11/SC22-NRE-016-Marcos_Schwarz-Programmable_Networking_with_P4.pdf

[118] https://sc23.supercomputing.org/wp-content/uploads/2023/11/SC23-NRE-020-MarcosSchwarz-Global_P4_Lab.pdf

[119] https://sc23.supercomputing.org/wp-content/uploads/2023/11/SC23-NRE-032-MagnosMartinello-PolKA.pdf

[120] https://sc24.supercomputing.org/wp-content/uploads/2024/11/nre032.pdf

[121] BIER https://datatracker.ietf.org/doc/html/rfc8279