Towards Accelerating the Network Performance on DPUs by optimising the P4 runtime

Iliadis-Apostolidis, Dimosthenis; Manaa, Khalid; Kadosh, Matty; Ioannou, Iacovos; Vassiliou, Vasos; Kosta, Sokol; Olmos, Juan Jose Vegas

Published in:
2024 32nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)

DOI (link to publication from Publisher):
10.1109/PDP62718.2024.00040

Publication date:
2024

Document Version
Accepted author manuscript, peer reviewed version

Towards Accelerating the Network Performance on DPUs by optimising the P4 runtime

Dimosthenis Iliadis-Apostolidis
Department of Electronic Systems
Aalborg University and NVIDIA Corporation Ltd.
Copenhagen, Denmark
dia@es.aau.dk

Khalid Manaa
NVIDIA Corporation Ltd.
Hermon Building,
Yokneam, Israel
khalidm@nvidia.com

Matty Kadosh
NVIDIA Corporation Ltd.
Hermon Building,
Yokneam, Israel
mattyk@nvidia.com

Iacovos Ioannou
University of Cyprus
University Avenue 1
Nicosia, Cyprus
ioannou.iakovos@ucy.ac.cy

Vasos Vassiliou
University of Cyprus
University Avenue 1
Nicosia, Cyprus
vasosv@ucy.ac.cy

Sokol Kosta
Department of Electronic Systems
Aalborg University
Copenhagen, Denmark
sok@es.aau.dk

Juan Jose Vegas Olmos
NVIDIA Corporation Ltd.
Hermon Building,
Yokneam, Israel
juanj@nvidia.com

***Abstract*—Data Processing Units (DPUs) are becoming increasingly popular, especially for use in conjunction with Warehouse-Scale Computers (WSCs) due to their ability to handle networking functions and data-centric workloads. Cost-performance, energy efficiency, network I/O, and batch processing workloads are important design factors for WSCs. Recent developments in AI and the never-ending increase in demand for data processing, cloud computing, and HPC set the optimisation of all those design factors as a high priority. DPUs can be utilised to achieve significant improvements in all those areas. This includes in-line network processing and upcoming enhanced security paradigms such as post-quantum cryptography (PQC) for quantum resilient communications or software-defined perimeters (SDP) for confidential computing implementations. Being P4-enabled and dRMT-based, DPUs allow for the reconfigurability of the network traffic without the need to change the hardware. However, the network performance on such devices is only sometimes deterministic since the actual traffic and the rules, both of which have to do with packet processing, are not known during compile time. In this paper, we envision how the network performance on DPUs can be accelerated. We describe the challenges that negatively impact the bandwidth and latency: the complex steering pipeline and the massive runtime needed to optimise. These challenges arise from the lack of information during compile time that is only known during runtime. Thus, we envision optimising during runtime by leveraging DPUs' reconfigurability on the network I/O. For this, we discuss the significant factors that must be considered to accelerate the network performance on such devices and we propose a solution for them.**

***Index Terms*—Networks, Programmable Networks, SDN, Data Plane Control Devices, DPUs, SmartNICs, WSCs, HPC**

I. Introduction

The never-ending increase in demand for big data processing, cloud computing, and HPC among others, results in particular interest for optimising the performance and efficiency of Warehouse-Scale Computers (WSCs). Cost-performance, energy efficiency, network I/O, and batch processing workloads are important design factors for systems of this scale. Data Processing Units (DPUs) are becoming increasingly popular due to the many advantages and improvements they can provide in all those areas. DPUs are SmartNIC devices with ARM cores and accelerators on board, enabling efficient handling of data-centric workloads. By offloading those workloads to DPUs, the central servers' utilisation can be reduced. An architecture of this setup can be seen in Figure 1, where hosts can offload workloads to the DPUs to improve their performance and energy efficiency while also utilising the DPUs for high-speed connection within the WSC.

Much research is being done that focuses on offloading intensive tasks onto DPUs. In [1], [2], [3], [4], many use cases are described and implemented on DPUs and SmartNICs. Some of the main directions of the research work are either focusing on application-specific designs or proposing the automation of offloading onto such devices, the computational performance benefits, the avoidance of overloading the central server and making the systems more energy efficient. In such a way, cloud computing and many intensive data-processing and high-performance computing applications can find acceleration benefits by utilising DPUs. With ARM cores and flexible and programmable accelerators on board, those tasks can also increase the performance by leveraging the concurrency such hardware can provide on the computational level. At the same time, a significant advantage of ARM-based devices that DPUs are, is energy efficiency since using these devices, on a WSC level, can lead to lower power consumption than having similar tasks run on x86-based devices or hosts [5], [6], [7], [8]. That means that in general, a task will consume less power when offloaded onto a DPU compared to being executed on a x86 host, as this is one of the main benefits of having DPUs on WSCs. The main limitation of this though, has to do with the task itself i.e., the level of parallelism it allows as an application, the utilisation it requires, the Dataflow etc.

A primary factor distinguishing DPUs is that they can also be used as data plane control devices. This means that they can take on the task of processing and forwarding the packets for

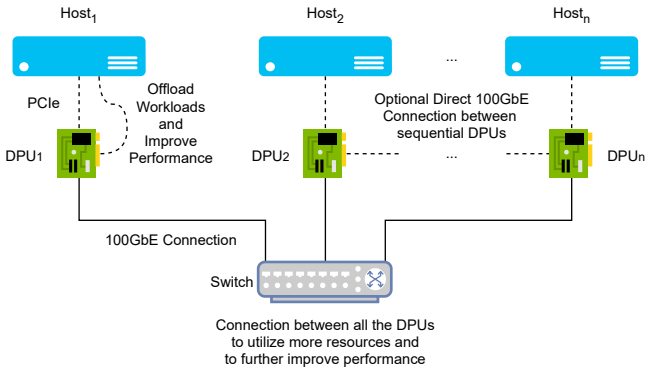

Fig. 1. Example of a WSC scenario. On a high level, each host has a DPU. Depending on how the system is orchestrated (Master-Servant, etc.), data-centric and computational heavy tasks can be offloaded onto the DPU. Note that optionally, there can be a connection between the DPUs by using a switch or a direct connection between adjacent DPUs so that the performance can be increased further by utilising more available resources, i.e., more DPUs concurrently.

the Network I/O. To do this, current state-of-the-art hardware uses P4 [9], a programming language that specifies how a data plane control device will process packets. DPUs are based on hardware and software architectures that enable the reconfigurability of the network without the need to change the hardware. Other SmartNICs might have different capabilities or be based on different architectures, but the challenges remain the same. The main focus of our work though will be Nvidia's DPUs.

Depending on the device that P4 targets and the features that need to be supported, there are several software architectures that P4 incorporates such as *i)* the Portable NIC Architecture (PNA)[1], which is used for programming the packet processing that takes place on DPUs, SmartNICs, and NICs and *ii)* the Portable Switch Architecture (PSA)[2], which is mainly used for programmable switches, thus having differences in the programming components compared to PNA. The main reason for using the PNA architecture has to do with the fact that physical interfaces found on DPUs are different from the ones found on switches. Also, PNA extends the software capabilities to support features often found on such devices but rarely on programmable controllers, such as Remote Direct Memory Access Offload (RDMA) and IPSec encrypt/decrypt, among others[3].

ASICs found on DPUs are responsible for handling the packet processing. They are based on the Disaggregated Reconfigurable Match+Action Table (dRMT) architecture, which was implemented to tackle some key challenges that occurred by using the RMT architecture. RMT uses sequentially-wired pipeline stages, meaning that the order in which the operations are executed is fixed. On the other hand, dRMT increases the inter-packet concurrency for the packet processing task. This is achieved by having a multiprocessor design and each one of those processors processes a packet to completion. Each processor can perform MA operations on more than one packet at a time. To achieve this, shared memory clusters, which are available to the processors through a crossbar, are used. This expands the capabilities of packet processing and introduces several challenges that must be addressed to optimise the performance of P4-enabled, dRMT-based devices on packet processing and Network I/O.

This paper explores how to accelerate the network performance by optimising the P4 runtime on Disaggregated Reconfigurable Match+Action Table (dRMT)-based Data Plane Control Devices, such as DPUs are. We describe the challenges, i.e., the complex steering pipeline and the massive runtime to optimise. We also discuss the parts that should be considered to be optimised during runtime to accelerate the network performance. Those parts are the table entries, which are unknown at compile time, their dependencies and the actual paths that will be used (traffic). We also discuss the need to implement tools to benchmark the performance and test the validity of the optimisations.

II. Background and Motivation

Packet processing on DPUs with the dRMT architecture. DPUs are based on the dRMT architecture [10], which uses a multiprocessor design for packet processing and shared memory clusters available to each processor through a crossbar. Every Match+Action Table (MAT) gets passed into a processor and runs to completion. RMT [11] is an architecture that has sequentially hardcoded pipeline stages. One significant advantage of dRMT is the flexibility and adaptability that it can provide during runtime. This can overcome an essential pipeline restriction used for packet processing on traditional switch ASICs. The execution is sequential for each MAT: match, then action on one stage, the same on the second stage, etc. This wastes cycles if, for example, the action is to drop a packet since it will happen at the end of the pipeline. DPUs with reconfigurable dRMT-based ASIC processing cores dropped packets can be discarded by halting the execution and then fetching the next packet, not wasting any cycles or resources. The other issue that has to do with RMT is that unused memory is unavailable from stage to stage. This is a significant advantage of dRMT, where the memory clusters are shared, meaning that every processor can access them through a crossbar, meaning that otherwise unused memory can be utilised by the processors.

Scheduling on dRMT-based devices. Nondeterminism in network processors' performance has to do with several challenges. Cache misses and contention in the processor-memory interconnect are factors that must be considered. This is why dRMT offers a static scheduling algorithm at compile time-based on the P4 program's dependencies and the capabilities of the available hardware resources. This scheduling mechanism solves the problem of contention and minimises cache misses.

[1]https://p4.org/p4-spec/docs/PNA.html
[2]https://p4.org/p4-spec/docs/PSA.html
[3]https://opennetworking.org/wp-content/uploads/2021/05/2021-P4-WS-Andy-Fingerhut-Slides.pdf

During runtime though, networking bottlenecks come into play, requiring different optimisations to resolve and increase the network performance. Even so, this contribution is significant to packet processing, especially regarding the hardware architecture for data plane control devices.

Challenges of packet processing on DPUs. Since DPUs are devices that can also handle the task of packet processing, several challenges arise regarding the network performance. P4 is used to enable the programmability of the network while also benefiting from the flexibility that it can provide. In essence, the network handling can be programmed on software without changing the hardware. One great challenge though, occurs when defining a complex steering pipeline. Information such as the MAT entries are unknown at compile time and volatile during runtime. This can result in an increase of the number of hops at the steering stage, having a negative impact on the network performance and needing massive runtime to optimise.

The need for optimisations during runtime. Network performance, i.e., the bandwidth and the latency, depends on the table entries and the traffic. This kind of information is unknown at compile time. That is why it is essential to take a deep dive into optimisations that can take place during runtime. In order to optimise the Network I/O, there is a need for optimising the P4 runtime.

III. Related Work

Enabling Network Programmability. With the advent of Software-Defined-Networks (SDNs) [12], many capabilities were added in networking, while SDN also described a way for programmable networks to become a reality. That is, by implementing the control plane functions as a different service, separating it from the data plane. In today's networks, where the I/O is higher than ever in an upward trend, state-of-the-art SDN-based solutions are also implemented in real-life scenarios. There is an adaption to this notion on both hardware and software to provide the quality and quantity needed for today's systems and their users. Although SDNs are implemented in software, the hardware architecture also adapts to deliver better performance and service the increasing I/O.

OpenFlow SDN [13] was the first enabler for the programmability of the control plane. This allowed for adaption to any changes needed on packet processing without changing or upgrading the hardware. One of the main issues was that by having different hardware data path designs, each vendor had to offer hardware-specific code. Also, programming new protocols was not an easy task, time-wise. That is why Programming Protocol-Independent Packet Processors (P4) [9] is considered a next-gen SDN-based programming language that tackles many of the challenges that occurred with OpenFlow.

P4 is a state-of-the-art programming language that enables programming of the control plane and there is a great adaptation of it throughout the industry. The necessary software components to accommodate a specific protocol can be implemented by a programmer and they have to be based on a particular software architecture that P4 encompasses, depending on the target. By providing a flexible and programmable parsing, control, and deparsing stage, P4 can modify the packet processing procedure without needing a hardware change. This can result in more performance and efficiency gains and adaptability to the volatile needs that occur on the network level. A significant advantage of P4, compared with OpenFlow, is that it describes the behaviour of the forwarding plane. By offering more specialised software architectures, P4 enables the re-programmability of data plane control devices on more familiar ground among different vendors.

Other work that contributes to enabling re-programmability during runtime is [14] and [15]. These papers emphasise the importance of making changes to network functions during runtime. Moreover, in [15], the ability of partial reconfiguration of switch data planes at runtime is described while also solving many challenges that arise with it.

Packet processing optimisations on RMT-based devices. Previous work such as [16], [17], [18], has focused on optimising RMT-based devices. To achieve this, all the optimisations that have been proposed have to do with optimising the procedure that involves the Match-Action Tables. This is done by reducing table dependencies based on policies, merging multiple tables with similar behaviour into one, or reducing match operations. More specifically, P5 [16] proposes a system based on information from application deployment, which can be used to simplify and reduce mutually exclusive or unused features from applications, thus decreasing the table dependencies and resource utilisation. B-Cache [17] proposes a caching framework for the Programmable Data Plane. One of the contributions is keeping track of packet processing behaviours, which are dispersed into multiple tables and enabling them to be compiled into one table. To deal with the increase in the pipeline size and the complexity of the P4 runtime, the authors propose exploiting behavior-level caching while maintaining the cache coherence during runtime. At the same time, they show increased network performance by using these optimisations. The solution presented by MATReduce [18] is to reduce the duplicate match operations between Match+Action Tables (MATs). The proposed system achieves this by using a preprocessor and a runtime manager, where the former merges duplicate match operations of the pipeline while the latter maintains policy consistency. The authors show that an increase in performance is obtained by merging the duplicate match searches, reducing the number of operations needed for packet processing.

All of the above accelerate the performance of the P4 runtime. They are not designed for dRMT devices though, but for RMT instead, with which every packet goes through specific paths, passing through all the stages. Still, methodologies have been proposed that can positively impact the performance of the P4 runtime.

IV. Packet Processing with P4

In this section, we explain how MATs work at the control stage and how to work towards optimising the P4 runtime.

A. *Match+Action Tables*

Match+Action Tables (MATs) are an essential component of packet processing and, more specifically, of the control/steering stage of the P4 flow. With these tables, the procedure is to match on packet header fields or metadata and then perform actions on the packet (e.g., forward, modify, drop, etc.). MATs are a significant part of the P4 program. Their main characteristics are known at compile time and other software components that are associated with them, such as the match algorithms, which are (pre-)defined inside the software architecture.

Those tables must contain the type of data to match on, i.e., which header fields to match on (key), as well as possible actions and action data for the packets. Optionally, they can contain table properties, such as size, default action, etc. An entry of such a table includes the content of the key to match (e.g., the content of the destination IP of the packet header) and a specific action to be executed if there is a match. In the Alg. 1, we clearly show the inner workings of MATs.

Algorithm 1 Process of MATs in P4 for Packet Processing

Initialization:
Define MATs with keys and actions.
Set table properties.
At Runtime:
for each incoming packet **do**
    Identify packet type from header.
    Select appropriate MAT based on packet type.
    Perform matching operation.
    if match is found **then**
        Execute the corresponding action.
    else
        Execute default action.
    end if
end for
Handling Different Packet Types:
Ethernet-type: Use exact match for MAC address.
IPV4-type: Use partial match algorithms.
Optimising Runtime:
Update MAT entries.
Maintain network functionality.
Optimise operations based on runtime data.

The biggest challenge is that the entries, i.e., the data to match on, are unknown during compile time. That means that based on the content of the key, the selection of a specific match operation (where the entry will be added) as well as the selection of the action to be performed if there is a match (what action will be inserted in the entry), is done during runtime. Exact Match, Longest Prefix Match (LPM), and Ternary Memory Lookups are match operations/algorithms that can be selected. Details on the parameters of the match can be given through the entries. To select a specific match operation on specific bits, a variable must be (pre-)defined inside the tables and the value should be passed through the

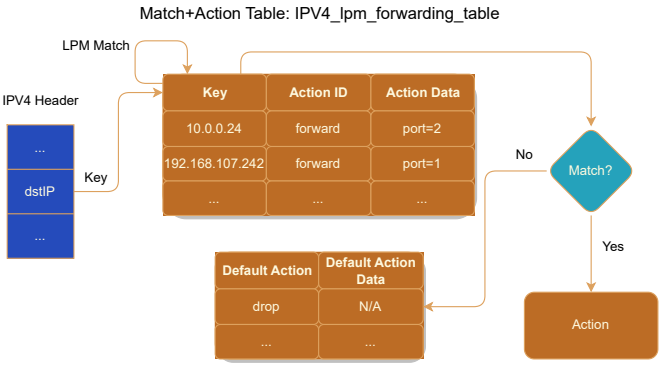

Fig. 2. Example of an IPV4 forwarding MAT. Note that the matching algorithm (Longest Prefix Match) that will be executed is MAT-specific and defined in the P4 program. The content of the key is extracted from the header at the parsing stage and it will be matched against the entries of the MAT at the control stage on the fields of that key during runtime. The entries of the MAT are unknown during compile time, i.e., the MAT can be empty when the program is initialised.

entries. Figure 2 gives an example of a MAT and an overview of the entire procedure.

When a packet is received, what happens next depends on the packet type, which is information that can be found inside the packet header. With P4, the programmer can select to implement different tables with different algorithms and flexible header parsers. In general, though, we describe common scenarios in which the process followed differs depending on the header type of the packet. If this packet has a header type of Ethernet which needs to be sent to a MAC address, the exact match is the operation that will be executed. How the packet will be processed depends entirely on the MATs implemented inside the P4 program and their entries, which are added/removed/modified during runtime. If the P4 program supports the processing of Ethernet-type headers, they will be parsed and there will be a search on the entries of the MATs to find the same key, which in this case is the destination MAC address. If there was a match, the action of the MAT rule that had this same key will be executed. A default (predefined) action, such as drop, will be performed if there is no match.

On the other hand, when a packet with a header type of IPV4 is received, a successful partial match can be enough to perform the corresponding action. Some actions, such as forwarding, might need more parameters to be specified for execution, such as the interface from which it will be sent. To make this decision, the operation that has to take place is to look for a match. It is helpful to describe two crucial partial match algorithms to demonstrate the main issues and challenges. LPM is an algorithm in which the longest prefix, i.e., the most specific prefix in the table, is selected. A bit-by-bit comparison on the entry field of the destination IP address is made and the prefix with the most matching bits is the prefix that is eventually the match.

In Ternary Memory Lookups, a mask bit is added, a ”care” or ”don’t care” bit. In P4, the ”don’t care” value is reserved on the _ (underscore character). The entry for such a match

operation is compared with the contents of the MATs. Then, it is determined which is the longest match to return as the match result and move to the action. The ternary match is given by the mask together with the value, as well as the priority, to resolve overlapping issues.

Although we mention those specific match algorithms, P4 enables the programmability of the network, allowing for those operations to be used on various header fields and for different situations. A P4 programmer cannot add more match algorithms. This can only be implemented inside the model description files, i.e., inside the various software architectures that P4 provides. Maintaining the correctness of the network functionality is essential and it is one of the factors that must be considered to optimise during runtime.

The above significantly impacts the P4 runtime and the complex steering pipeline that can be defined. That is since the selection of the match algorithms that will be executed and the table entries are given and changed during runtime and are unknown at compile time. An essential part of the steering pipeline is processing the packets that will take place on the control data plane. The entries of MATs are an indispensable component of the P4 runtime. Thus, it is crucial to tackle this challenge by optimising the P4 runtime.

B. The P4 runtime

P4 is a programming language that defines how the packets are processed on a data plane control device. In essence, it enables the definition of the behaviour of such devices' data plane. The main benefit that P4 offers is that operators can change the network behaviour, i.e., they can reprogram how a DPU will handle the traffic without the need to change the hardware. P4 has many tools that come with it, which can be used to simplify the implementation procedure. A P4 program defines a pipeline for packet processing. All the objects incorporated within this program are used and modified during runtime.

A P4 program defines the parsing, the control/steering, and the deparsing stages of packet processing, as shown in Figure 3. The way that the packets have to be processed is broken down into three stages, so that:

- The information that is extracted from the header is defined at the parsing stage.
- The matching algorithm that will be used and the definition of the action that will occur if there is a match or by default, is described at the control stage.
- The packet header is reassembled at the deparsing stage.

Each stage is critical for having a complete implementation for processing the packets. The parsing and the deparsing steps are pretty straightforward. In the beginning, the header of the packet is processed so that important information is extracted, such as the type of the header (e.g., Ethernet or IPV4), the destination address (MAC or IP), and other information, accordingly (time-to-live, etc.). At the end of the procedure, the header is reassembled as computed by the pipeline and the outgoing packet is constructed. It is also important to note that the parser is flexible. Since P4 is protocol-independent,

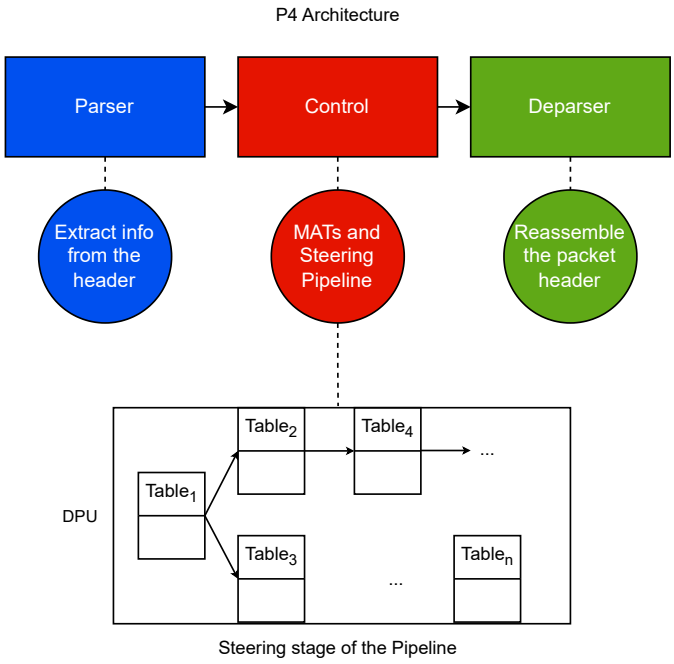

Fig. 3. A more detailed look into the pipeline. In RMT, the pipeline is implemented into stages sequentially. In dRMT, the pipeline is implemented on each processor. The packets cannot move between different processors. The packets are passed into each processor in a round-robin fashion and they are Run-To-Completion (RTC). The path that each packet will follow, though, can be different since it is dependent on the entries and the actual traffic that will go through the steering stage.

the programmer can code the necessary software components to support any protocol needed at the parsing stage. The headers' formats, types, and fields are implemented inside the P4 program. This is a significant advantage of P4 since it makes it future-proof and, at the same time, adaptable for any use case that needs to be implemented.

Between the parsing and the deparsing stages lies an important one, the control/steering stage. The MATs are defined in this part of the P4 program. For this purpose, the match algorithms are selected for each table and the appropriate action to be executed if a match is defined. Optionally, a default action is also determined to occur if there is no match. The significance of this stage, though, doesn't arise from the implementation itself but rather from the entries of the tables that will be inserted, modified or deleted, which will be used during the P4 runtime.

P4 runtime is a control plane specification for controlling the data plane elements of a device defined or described by a P4 program. The pipeline is created during compile time and it might be insufficient in terms of performance because information that can have a negative impact on its performance is unknown at compile time. The definition of the pipeline is mainly connected to the description of the Match+Action tables, their dependencies and optional fields and parameters. The performance of the pipeline, though, is dependent on the entries of the MATs and the actual traffic.

As described above, the entries of the MATs, also known as rules, are not known at compile time. Thus, depending on how the user will add, remove, or modify those entries during runtime, it can negatively impact the network performance. By network performance, we refer to the overall traffic, latency, and bandwidth patterns. Thus, there is an excellent need for optimising the P4 runtime to accelerate the network performance on dRMT-based devices so that line-rate packet processing can be achieved.

V. Towards optimising the P4 runtime

Many avenues can be explored to optimise the P4 runtime. This paper focuses on specific factors that must be considered to accelerate the network performance on DPUs and dRMT-based devices. It is general knowledge that not every SmartNIC has ARM cores and accelerators on board (other vendors use different solutions) and that the ASICs of the SmartNICs can be based on different architectures. P4 is an important common block, as an open-source aspect of this research topic, that enables reconfigurability and adaptability when it comes down to e.g., how packets will be processed, across different solutions. The main challenges i.e., *i)* the complex steering pipeline and *ii)* the massive runtime required to optimise the network performance, remain the same.

This pipeline has to be able to adapt to the changes that have to do with the entries and how they will result in a path that might be costly for the system and the network performance. For this to be possible, a mechanism has to be in place that will be able to make *optimisations on the tables during runtime with the use of an implementation that will be able to also make changes on the control plane. Many algorithms can be implemented to efficiently manipulate the tables and their entries, resulting in better performance*.

Thus, our approach provides the Alg. 2 that enhances the pipeline. Each packet must go through a specific path, i.e., several specific tables, to be processed. One of the main challenges of the complex steering pipeline is this traffic, which, in essence, encompasses the actual paths that will be used and it plays a significant role in network performance. As such, traffic is another critical factor to be taken into consideration when optimising the P4 runtime. During runtime, when the table entries will be added, removed or modified, it is crucial to *monitor the paths* that will be needed so that the optimisations that will take place onto the tables will also be concerning the traffic that is expected to happen when executed. The importance of this is the table dependencies that can occur when multiple packets have to go through overlapping nodes in the path.

There is also a need for *an algorithm that resolves the table dependencies during runtime*. dRMT-based devices have a scheduling algorithm that can deal with data races on the universally accessible memory through the crossbar, but this is not enough to increase performance. Although there might not be any (unresolved) data dependencies during runtime, the memory's availability has to be considered for the optimal pipeline version, exploiting the benefits of dynamic resource allocation on dRMT-based devices.

Algorithm 2 Enhanced P4 Runtime Processing with Path Determination, Dependency Resolution, and Performance Tools

```
Initialization:
Define MATs with keys and actions.
Set table properties.
Path Determination and Processing:
for each incoming packet do
    Identify packet type and determine the specific path.
    for each MAT in the path do
        Perform matching operation.
        if match is found then
            Execute the corresponding action.
        else
            Execute default action.
        end if
    end for
end for
Monitoring and Optimisation:
Monitor paths and optimize MAT entries based on traffic.
Resolving Table Dependencies:
Implement algorithm for runtime dependency resolution.
Consider dynamic resource allocation in dRMT-based devices.
Performance Tools Implementation:
Develop tools for measurement, validation, and benchmarking.
Extend tools for complex pipeline testing.
Runtime Table Management:
Dynamically manage MAT entries considering dependencies.
```

Last but not least, there is a need for implementing *tools* that will be able to *i)* measure the performance, *ii)* validate the execution and *iii)* benchmark against unoptimized versions are crucial. Another part of those tools has to do with the complexity of the testing scenarios, meaning not only adding, removing or modifying entries but also creating complex steering pipelines to be resolved. In this way, testing can be automated by implementing new tools or by extending the capabilities of current ones.

VI. Conclusion

To further contribute to accelerated computing, there is a need for network performance optimisations on state-of-the-art hardware with such capabilities. Network performance on DPUs, based on the dRMT architecture, is not deterministic. It depends on many factors, but the main issue is that information such as table entries (rules) are unknown during compile time. Since those rules are added/removed/modified during runtime, a result of a complex steering pipeline can have a negative impact on the performance. Additionally, a massive amount of time will be needed to optimise since the number of hops will increase. In this work, we envision how the P4

runtime can be optimised. We make a detailed assessment of the challenges while discussing essential factors to consider to achieve line-rate packet processing on DPUs. Such line-rate packet processing enables hyper-distributed computing paradigms that seamlessly interconnect and support abstracting the storage and computing capacity of the cloud and the data processing and low latency features of the edge. DPUs enable the management of such complex fabrics while guaranteeing network security, data privacy, and confidentiality in such a cloud-edge continuum.

VII. Acknowledgements

This work was partly funded by the QUARC project by the European Union Horizon Europe research and innovation program within the framework of Marie Skłodowska-Curie Actions with Grant Agreement No 101073355.

ADROIT6G project has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101095363.

References

[1] T. Cui, W. Zhang, K. Zhang, and A. Krishnamurthy, "Offloading load balancers onto smartnics," APSys '21, (New York, NY, USA), p. 56–62, Association for Computing Machinery, 2021.

[2] Y. Gao, Z. Wang, and S.-B. Tsai, "A review of p4 programmable data planes for network security," *Mob. Inf. Syst.*, vol. 2021, jan 2021.

[3] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading distributed applications onto smartnics using ipipe," in *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, (New York, NY, USA), p. 318–333, Association for Computing Machinery, 2019.

[4] Y. Qiu, J. Xing, K.-F. Hsu, Q. Kang, M. Liu, S. Narayana, and A. Chen, "Automated smartnic offloading insights for network functions," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, (New York, NY, USA), p. 772–787, Association for Computing Machinery, 2021.

[5] K. Gupta and T. Sharma, "Changing trends in computer architecture : A comprehensive analysis of arm and x86 processors," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pp. 619–631, 06 2021.

[6] D. Yokoyama, B. Schulze, F. Borges, and G. Mc Evoy, "The survey on arm processors for hpc," *J. Supercomput.*, vol. 75, p. 7003–7036, oct 2019.

[7] E. Blem, J. Menon, and K. Sankaralingam, "Power struggles: Revisiting the risc vs. cisc debate on contemporary arm and x86 architectures," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–12, 2013.

[8] E. Blem, J. Menon, and K. Sankaralingam, "A detailed analysis of contemporary arm and x86 architectures," 01 2013.

[9] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, p. 87–95, jul 2014.

[10] S. Chole, A. Fingerhut, S. Ma, A. Sivaraman, S. Vargaftik, A. Berger, G. Mendelson, M. Alizadeh, S.-T. Chuang, I. Keslassy, A. Orda, and T. Edsall, "Drmt: Disaggregated programmable switching," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, (New York, NY, USA), p. 1–14, Association for Computing Machinery, 2017.

[11] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," *SIGCOMM Comput. Commun. Rev.*, vol. 43, p. 99–110, aug 2013.

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

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, p. 69–74, mar 2008.

[14] J. Xing, Y. Qiu, K.-F. Hsu, H. Liu, M. Kadosh, A. Lo, A. Akella, T. Anderson, A. Krishnamurthy, T. S. E. Ng, and A. Chen, "A vision for runtime programmable networks," in *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*, HotNets '21, (New York, NY, USA), p. 91–98, Association for Computing Machinery, 2021.

[15] J. Xing, K.-F. Hsu, M. Kadosh, A. Lo, Y. Piasetzky, A. Krishnamurthy, and A. Chen, "Runtime programmable switches," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, (Renton, WA), pp. 651–665, USENIX Association, Apr. 2022.

[16] A. Abhashkumar, J. Lee, J. Tourrilhes, S. Banerjee, W. Wu, J.-M. Kang, and A. Akella, "P5: Policy-driven optimization of p4 pipeline," in *Proceedings of the Symposium on SDN Research*, SOSR '17, (New York, NY, USA), p. 136–142, Association for Computing Machinery, 2017.

[17] C. Zhang, J. Bi, Y. Zhou, K. Zhang, and Z. Ma, "B-cache: A behavior-level caching framework for the programmable data plane," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 00084–00090, June 2018.

[18] X. Chen, D. Zhang, and H. Zhou, "Matreduce: Towards high-performance p4 pipeline by reducing duplicate match operations," in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, Dec 2018.