› DESIRE6G ‹

D4.1: Initial report on the DESIRE6G unified programmable data plane layer

HEU 6G SNS JU Project

Grant No. 101096466

Co-funded by
the European Union

Document properties

Document number	**D4.1**
Document title	Initial report on the DESIRE6G unified programmable data plane layer
Work Package	4
Editors	Andrea Sgambelluri (SSSA)
Authors	Andrea Sgambelluri, Alessandro Pacini, Faris Alhamed, Luca Valcarenghi (SSSA), Attila Mihály (ERI-HU), Marta Blanco Caamaño, Rafael A. López Da Silva, Alejandro Muñíz Da Costa (TID), Juan Jose Vegas Olmos (NVIDIA), Anastassios Nanos, Christos Panagiotou, Charalampos Mainas, (NUBIS), Revaz Berozashvili, Kiran Chackravaram, Simon Pryor (ACC), Vincent Lefebvre (TSS), Péter Vörös, Gergő Gombos, Sándor Laki (ELTE), Francesco Paolucci, Filippo Cugini (CNIT), Tom Wassing, Angelos Dimoglis, Chrysa Papagianni (UVA), Luis Velasco, Marc Ruiz, Jaume Comellas, Sima Barzegar (UPC), Roberto Gonzalez, Nicolas Weber (NEC)
Internal reviewers	1. Attila Mihály (ERI-HU), 2. Roberto González (NEC)
External reviewers	1. Anastassios Nanos (NUBIS), 2. Carlos J. Bernardos (UC3M)
Dissemination level	Public (PU)
Status of the document	Final
Version	1.0
File name	DESIRE6G _D4_1_Initial_report_unified_programmable_data_plane
Contractual delivery date	November 30, 2023
Delivery date	November 30, 2023

Document history

Revision	Date	Issued by	Description
First draft	16/10/2023	Andrea Sgambelluri (SSSA)	First draft for internal review
External Review	14/11/2023	Andrea Sgambelluri (SSSA)	Complete draft for external review
Final Revision	28/11/2023	Andrea Sgambelluri (SSSA)	Complete and clean version
Submission	30/11/2023	Andrea Sgambelluri (SSSA)	Ready for submission

Abstract

The deliverable describes the architecture design of the unified data plane layer, including the identified interfaces to access, configure and customize the data plane functions, the Infrastructure Management Layer (IML) that provides a unified abstraction of the Programmable Data Plane (PDP) targets, the network functions of the DESIRE6G infrastructure including the partial implementation of 6G network functions in PDPs. The deliverable also identifies the key concepts, use cases and procedures that need to be supported by the unified programmable data plane layer of DESIRE6G: multitenancy support and the deployment of customized network functions, securing their isolation, offloading computations, or accelerating infrastructure services, allowing customized and fine-grained pervasive monitoring over end-to-end paths across the entire DESIRE6G network domain.

Keywords

Infrastructure Management Layer (IML), In-band telemetry (INT), Network/Infrastructure monitoring, Hardware/application acceleration

Disclaimer

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096466.

DESIRE6G is supported by the Smart Networks and Services Joint Undertaking. This report reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

Table of Contents

1. Introduction 14

2. High-level architecture or the programmable data plane 16

2.1 DESIRE6G Architecture Vision 16

2.2 Definition of the Programmable Data Plane 18

2.2.1 Role of IML 20

2.2.2 IML interfacing the other components 22

3. Main procedures 24

3.1 Procedures inside PDP 24

3.1.1 Runtime load optimization in the data plane 24

3.1.2 Quality of Service 30

3.1.3 Approaches ensuring Quality of Service 32

3.1.4 NF routing 34

3.2 Procedures involving PDP 35

3.2.1 NF deployment on a single DESIRE6G site 35

3.2.2 Deployment of accelerated software applications 36

3.2.3 Monitoring-based run-time optimization 38

4. Deep slicing support 40

4.1 Network Service Deployment in the DESIRE6G Infrastructure 40

4.2 Network Service deployment on a single DESIRE6G site 41

4.3 Multitenancy support for P4 data planes 42

5. Pervasive monitoring 44

5.1 Reference scenario 44

5.2 Pervasive monitoring architecture 47

5.3 Service Monitoring 48

5.3.1 Telemetry approaches 48
5.3.2 INT DESIRE6G Case Studies 52
5.3.3 Telemetry Collector 58
5.4 Telemetry and Distributed Intelligence 61
5.5 Infrastructure Monitoring 64
5.5.1 Network Monitoring 65
5.5.2 Cloud Monitoring 66
5.5.3 RAN Monitoring 67
5.5.4 Software Monitoring 69
6. Acceleration functionality 73
6.1.1 Introduction 73
6.1.2 HW acceleration 73
6.1.3 Application acceleration 75
6.1.4 Neural Network Acceleration 76
6.1.5 Hardware Accelerated DP NFs 79
7. Conclusions 86
8. References 87

List of Figures

Figure 1. Generalized DESIRE6G architecture. 18
Figure 2. Infrastructure with heterogeneous devices that can execute data plane logic. 19
Figure 3. High-level overview on how IML interacts with other components of the infrastructure 23
Figure 4. Data plane optimization is hidden from NF-CP by IML 24
Figure 5. A general application scenario for LO-CP and LO-DP components of the load balancer 25
Figure 6. Different variants for DP implementation that can be used in packet core networks 26
Figure 7. The place of Load Optimizer in the forwarding path. 28
Figure 8. Flowchart showing the High-level functionality for heavy-hitter handling 30
Figure 9. Network resources shared between three network services. 31
Figure 10. NF routing across DESIRE6G sites 35
Figure 11. NF deployment on PDP HW through vAccel. 36
Figure 12. vAccel integration with container runtimes 37
Figure 13. IML, Monitoring. MAS interaction 39
Figure 14. NF deployment procedure. 41
Figure 15. Example data path service slice deployment on a single DESIRE6G site 42
Figure 16. Multitenant deployment of an NF data plane on P4-programmable Tofino ASIC 43
Figure 17. Reference scenario of application for the pervasive monitoring system. 45
Figure 18. Service and infrastructure pervasive monitoring. 48
Figure 19. Postcard telemetry scenario. 50
Figure 20. INT scenario. 50
Figure 21. Int and clock synchronization scenario. 52
Figure 22. Bidirectional Int for link latency evaluation in case of no clock synchronization. 53
Figure 23. P4-enabled UE INT scenario with clock synchronization 54
Figure 24. Example of Multitechnology end-to-end scenario with radio and transport domain. 55
Figure 25. P4-enabled path tracing scenario 56
Figure 26. Percentage of flows where the path update IS detected. 57
Figure 27. Time for path update detection 57
Figure 28. Structure of a sample telemetry header for PLINT 58

Figure 29. Two stage telemetry collector used for postcard telemetry implementation 59
Figure 30. P4 Collector: aggregation mode 61
Figure 31. P4 Collector: correlation mode 61
Figure 32. Different Report stack in No aggregation, Agg and Cor modes. 61
Figure 33. Illustrative scenario supporting an E2E 6G service 63
Figure 34. Pervasive Telemetry And Distributed Intelligence Solution. 64
Figure 35. DESIRE6G RAN Telemetry interfaces 67
Figure 36. General principle of the time series extraction on the protected software control flow graph during its execution 70
Figure 37. Trampoline insertion performance impact on demanding primitives 71
Figure 38. Gain brought leveraging buffers for trampoline logs 72
Figure 39. SOL NN acceleration architecture 76
Figure 40. SOL optimization example 77
Figure 41. Load balancing schema in the LO-DP packet processing digital circuit 80
Figure 42. Hybrid gNodeB model using P4 programmable hardware and SW components 83
Figure 43. Algorithm for buffering service. 84
Figure 44. Acceleration architecture 85

List of Tables

Table 1. Resource Utilization....58
Table 2. Available metrics at CU-CP and CU-UP....68
Table 3. Available cell-based metrics....68
Table 4. Available UE based metrics....69
Table 5. Hw acceleration....74

List of Acronyms

AAL	Acceleration Abstraction Layer
AF	Application Function
AI	Artificial Intelligence
AI/ML	Artificial Intelligence/Machine Learning
ASIC	Application Specific Integrated Circuit
BWE	Bandwidth Enforcer
CFS	Control Flow Shadowing
CID	Connection ID
CPU	Compute Unified Device Architecture
CUDA	Compute Unified Device Architecture
DFP	Data Flow Programming
DLINT	Deterministic Lightweighted In-Band Network Telemetry
DNN	Deep Neural Network
DPU	data processing unit
FPGA	Field Programmable Gate Array
GPU	Graphical Processing Unit
HH	Heavy hitter
HQoS	Hierarchical Quality of Service
HW	Hardware
HLIR	High-Level Intermediate Representation
K8S	Kubernetes
IML	Infrastructure management layer
INT	In-band Network Telemetry
LO	Load Optimizer control plane
LO-CP	Load Optimizer control plane
LO-DP	Load Optimizer data plane
L4S	Low Latency, Low Loss, Scalable Throughput
MAS	Multi Agent System
ML	Machine Learning
NF	Network Function control plane
NFR	Network Function Routing
NF-CP	Network Function control plane

NF-DP	Network Function data plane
NIC	Network Interface Card
NN	Neural Network
NSM	Open RAN
O-RAN	Open RAN
OVS	Open vSwitch
PDP	programmable data plane
PLINT	Probabilistic Lightweighted In-Band Network Telemetry
PM	Performance Monitoring
PPV	Per Packet Value
P4	Programming Protocol-Independent Packet Processors
QoS	Quality of Service
RAN	Radio Access Network
RNN	Recurrent Neural Network
SDN	Software-Defined Networking
SDN-CP	Software-Defined Networking Control Plane
SLA	Service Level Agreement
SMO	Service and Management Orchestrator
SRv6	Segment Routing IPv6
SW	Software
UE	User Equipment
uCID	Micro Connection ID
uFC	Micro Flow Cache
VES	VNF Event streaming
VIM	Virtual Infrastructure Manager
WAN	Wide Area Network

Executive Summary

The main task of WP4 is to develop the unified programmable data plane (PDP) layer of DESIRE6G architecture. To this end, the first phase of the study, covering the first 11 months of work on WP4, focuses on the architecture concept for the programmable data plane (PDP) proposed for DESIRE6G project, along with a description of the functionality of the main components and their interworking during the different procedures involving the data plane.

The tasks and activities in WP4 are designed to address use-cases and requirements identified in WP2. The components developed in WP4, for controlling, accelerating, and monitoring the programmable data-plane, will be integrated with the management and orchestration plane modules, designed in WP3, being experimentally validate in WP5 demonstrators.

Key Contributions

In T4.1 (Data plane support for deep slicing), the key contributions have been the following:

- Developing an architecture concept that enables programmability in the data plane, cloud-native features like automatic scaling and failover and transparent acceleration of both network functions and applications [1].
- Defining high-level mechanisms in PDP to support
  - Service deployment, including deployment of accelerated network functions and applications as well as ensuring proper traffic routing among the different functions.
  - Run-time load optimization in the data plane using programmable devices.
  - Real-time service assurance in the PDP via traffic management using a combination of existing methods.
  - Near-real time network service assurance supported by service monitoring.
- Developing the concept of deep slicing supporting multi-tenancy in the programmable data plane

In T4.2 (Accelerated network functions), the key contributions have been the following:

- Selecting the candidate packet processing NFs (both business logic NFs and infrastructure NFs) to implement acceleration functionality, including selection of hardware type.
- Developing concepts for the acceleration functionality of AI-based NFs (i.e., Mobile NFs and Infra NFs) and application servers and their integration into the system.

In T4.3 (Pervasive Monitoring and intelligent data aggregation), the key contributions have been the following:

- Developing an architecture for pervasive monitoring
- Defining service monitoring for several DESIRE6G use cases.
- Developing the concept for providing telemetry for a distributed intelligence to ensure near-real time service assurance.
- Defining and enabling the adoption of integrity software monitoring
- Study of the state of the art for infrastructure monitoring.

1. Introduction

In the rapidly evolving landscape of network and data processing technologies, the demand for flexibility, efficiency, and scalability has driven the development of programmable data planes. The traditional concept of a network's data plane, responsible for forwarding packets or processing data, has been undergoing a significant transformation. Programmable data plane represents a paradigm shift, empowering network operators and data processing professionals with the ability to reprogram the behaviour of the network bottom up to meet the dynamic demands of modern applications and services.

Traditionally, network devices relied on fixed-function hardware for tasks like packet forwarding, filtering, and transformation. These fixed-function devices often limited the adaptability and innovation required to keep pace with the evolving nature of network traffic and data processing requirements. Data plane programmability is instead a technology that allows for the customization and dynamic reconfiguration of network data processing elements.

The concept of programmable data plane extends beyond just network routing and switching. It has found applications in various domains, including edge computing, cloud computing, and even the Internet of Things (IoT).

Pursuing deep programmability of the network fabric both vertically (i.e., control and data plane) and horizontally (i.e., from the radio access network to edge and core network) is expected to accelerate in 6G, in order to support 6G performance requirements and increase flexibility and sustainability of the infrastructure.

This deliverable describes all the activities of DESIRE6G related to programmable data plane and it is structured as follows:

- Section 2 focuses on the high-level description of the programmable data plane architecture, introducing the IML component and the interfaces towards the other functional components.
- Section 3 reports the main procedures, designed, and defined to involve the programmable data plane.
- Section 4 introduces and describes the concept of deep slicing supporting multi-tenancy in the programmable data plane.

- Section 5 presents the pervasive monitoring activities, adopted to collect service and infrastructure metrics from the end-to-end environment.
- Section 6 reports the work considered in terms of acceleration at both hardware and application levels.
- Section 7 reports the final conclusions, points to further work in the project regarding these use cases and services presented in this deliverable.

2. High-level architecture or the programmable data plane

2.1 DESIRE6G Architecture Vision

Note: this section is a copy of Chapter 3 from Deliverable 2.1: Definition of Use Cases, Service Requirements and KPIs/KVIs [2]. It is included here to facilitate understanding of the terms and concepts discussed in this document.

Figure 1 shows the components of the DESIRE6G system. The network consists of topologically separated DESIRE6G sites, including local HW and compute resources running NFs necessary for the execution of the network services and application functions (AFs) required by the (application-specific) services. Each DESIRE6G site has almost identical setup with slight differences based on necessity, e.g., the sites running Radio Access Network (RAN) functionality are equipped with sufficient hardware entities and/or non-programmable (static) functions, all of which are still considered NFs by the system. Also note that the User Equipment (UE) might also be able to run a stripped-down version of the DESIRE6G stack using its own hardware and software stacks. This is a big difference compared to existing UE approaches and it makes E2E service control possible. Of course, legacy UEs can still use the system as today attaching to some "edge-to-edge" service, e.g., normal Internet access.

Between the DESIRE6G sites there can be non-programmable elements. These can be controlled by Software-Defined Networking Controller(s) (SDN-C) or by traditional routing protocols. The minimum requirement is that the DESIRE6G infrastructure must understand the reachability of the other sites from each given site (e.g., IP addresses of the other site's gateway).

The main components are the following:

- Service Management and Orchestration (SMO): Similar to existing specifications (i.e., NFV orchestrator for ETSI MANO), the SMO is responsible for end-to-end service life-cycle management, including service provisioning and deployment, network slice management and network optimizations. It contains several modules responsible for these different tasks and it is the main interface towards the external world. In addition, the SMO is also responsible for the management and orchestration of the Open RAN components, including Open Central Unit (O-

CU), Open Distributed Unit (O-DU) functions, and near-real-time radio intelligent controller (Near RT-RIC)

- Multi-Agent-based network intelligence System (MAS) and telemetry: MAS implements distributed network intelligence closer to the physical infrastructure, as it is responsible for receiving service-specific monitoring information and fine-tuning the network and compute resources. It configures and uses a pervasive telemetry system to receive service specific KPIs, e.g., by monitoring end-to-end latency for latency-sensitive or latency-critical services.
- Programmable Data Plane (PDP) layer including the Infrastructure Management Layer (IML) and the HW/SW specific implementations of the NFs: The E2E Programmable Data Plane is employing NFs to carry out the logic of the selected service. NFs contain a Control Plane component (NF-CP) and the packet processing Data Plane logic (NF-DP). Note that at this level the main role of NF-CP is to configure and update the objects (e.g., tables, registers, etc.) in the corresponding NF-DP. These are separated by the IML responsible for transparent scaling, acceleration, and deployment, i.e., it is the bridge between the logical and the physical network function. IML hides the implementation and deployment details of NF-DP from NF-CP by providing a simple unified view of the data plane. For example, even if NF-DP is vertically or horizontally disaggregated into multiple data plane programs running on different HW/SW targets, IML ensures that NF-CP only sees a single NF-DP instance and IML handles the complexity of managing the disaggregated packet processing components.

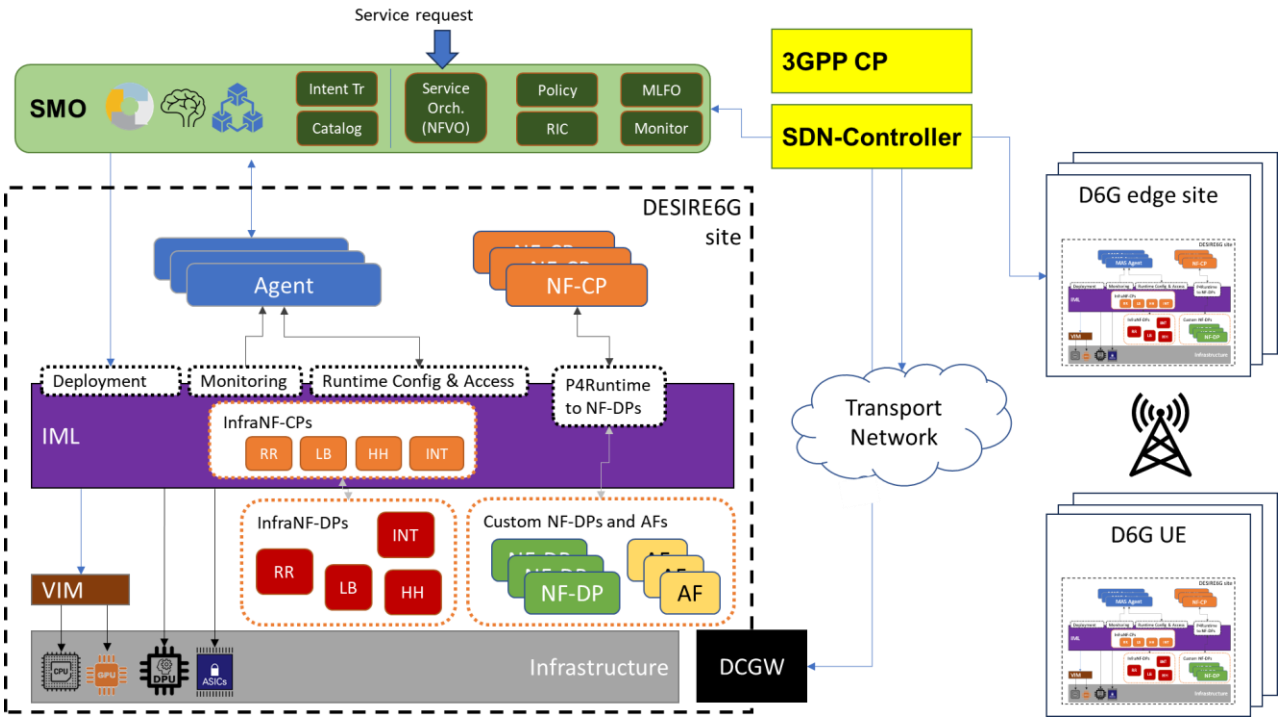

FIGURE 1. GENERALIZED DESIRE6G ARCHITECTURE.

2.2 Definition of the Programmable Data Plane

Adopting Programmable Data Plane (PDP) solutions (a.k.a. deep programmability) in the packet processing and forwarding infrastructure of DESIRE6G extends the support to custom-made, slice-specific data path functions and leads to better performance and higher visibility, needed for near real-time control. Seamless migration of such functions between heterogeneous systems (e.g., ASICs, DPUs, FPGAs, etc.) is currently challenging. Furthermore, current SDN platforms do not support the complexity of the heterogeneous “softwarized” networks envisioned in 6G, as current control lacks advanced and automatic forwarding capabilities, per-packet treatment and manipulation functions, high precision traffic monitoring/telemetry, and support for services with ”highly demanding network requirements. Pursuing deep programmability of the network in 6G both vertically (i.e., control and data plane) and horizontally (i.e., end-to-end from the RAN to edge and core network) to fulfil stringent performance requirements, can provide significant benefits, including dynamic traffic engineering where decisions can be made (near) real-time, In-band Network Telemetry (INT) for latency-critical services, slicing and multi-tenancy, and offloading network functions to the data plane.

In DESIRE6G, we assume that along every end-to-end path there are different PDP resources (e.g., smartNICs, ASICs, x86/ARM servers, etc.) that can execute packet processing steps of network functions defined by the network service to be implemented in each slice. Figure 2 illustrates this concept: the forwarding path contains different P4 programmable switches while at some locations (edge, core) there are other devices (servers, DPUs, smartNICs, GPUs, ASICs, FPGAs, etc.) that can execute and/or accelerate network function data planes or host high-level services (including network function control planes, third-party service applications, etc.).

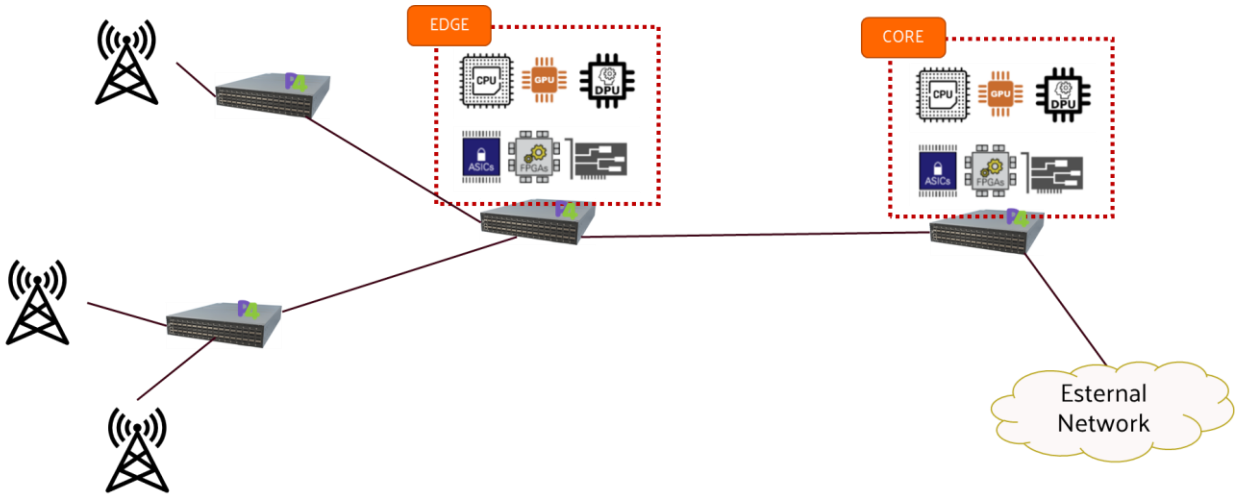

FIGURE 2. INFRASTRUCTURE WITH HETEROGENEOUS DEVICES THAT CAN EXECUTE DATA PLANE LOGIC.

A cloud-native approach will be considered for the design and implementation of the unified programmable data plane layer, where the resources will be provided though an abstract infrastructure management layer (IML) unifying the deployment and runtime optimization processes on top of heterogeneous PDP targets. In DESIRE6G, a network service will be defined as a logical service graph composed of different network functions together with different constraints on the components, links, and E2E paths that need to be considered during both deployment and operation. Resource allocation, deployment of the service between end-point locations and runtime optimizations of the data plane functions (e.g., scaling, traffic differentiation, etc.) are all implemented by the DESIRE6G infrastructure, similarly to cloud-native application management, e.g., in Kubernetes. The main goals of the unified PDP layer is to 1) hide the heterogeneity of different PDP targets by providing an abstract API to the upper layers, 2) provide support for deep slicing including multitenancy support on PDP hardware and QoS/SLA enforcement., 3) provide support for network function acceleration or service acceleration by

offloading some tasks to PDP, 4) provide support for fine-grained and real-time visibility of the network infrastructure with in-band network telemetry (INT), and 5) provide data plane primitives for runtime optimization (e.g., load balancing, scaling, heavy hitter handling, etc.).

2.2.1 Role of IML

The heterogeneous PDPs used in DESIRE6G will be managed by a common Infrastructure Management Layer (IML) that will be responsible to deploy data plane pipelines, configure them through a common API and interconnect their appropriate ports via virtual links. As shown in Figure 1, the IML will interact with the heterogeneous data plane software and hardware targets through a driver system and provide a northbound interface for other components of DESIRE6G for deployment, configuration, management, and monitoring. More details on the overall architecture can be found in [2]. Control planes of specific data plane programs will be able to manage their data plane objects (tables, registers, counters, etc.) via dedicated P4Runtime interfaces. Note that P4Runtime is a generic control plane interface that can also work with non-P4-based data planes. IML will primarily integrate P4 programmable data planes and seamlessly handle the different architecture models, language dialects and target-specific (mostly hardware) constraints. In addition to the general support of P4 targets, IML will also provide support for some selected network accelerators like FPGAs and non-P4 programmable DPUs. IML will also include ORAN's AAL functionalities (esp. inline model for packet processing offloading to programmable NICs). The roles of IML can be split into deployment and runtime functions.

For the deployment of a customized network function, IML will provide a unified API through a given network function that can be deployed on a data plane target with an associated flavour of a deployment setup in a specific DESIRE6G site. Note that the scope of the IML is limited to a DESIRE6G site (i.e., a computing cluster extended with data plane acceleration/execution hardware). Such flavours may regard heavy hitter handling with hybrid data plane hardware, load balancing with scaling options, single instance deployment, etc. It also determines the capabilities for runtime optimization. In addition, hardware PDPs are often not prepared for shared usage and thus hosting multiple data plane programs at the same time becomes challenging. IML will also contain components providing multitenancy support for P4 PDPs by enabling the aggregation of separate P4 data plane programs and their

deployment on P4-programmable switches (e.g., Tofino or software switches) or smartNICs (e.g., NetFPGA).

IML will both handle the deployment of application and network services on virtual resources and PDP resources. For traditional service deployment, it will provide a narrow wrapper layer on top of a Virtual Infrastructure Manager (VIM) (e.g., K8S), used to control computational resources of a cloud environment. While, for PDP resources, it will introduce an extensible set of functionalities to support deployment of customized network functions on various PDP targets (Tofino ASIC, NetFPGA, etc.) and support the multitenant usage of selected HW PDPs (focusing on P4 programmable ones). Note that different PDP targets may have different deployment workflows (e.g., Tofino ASIC requires the compilation of the new P4 code and the binary needs to be loaded to the device, causing interrupts in the operation and the switch will start with reset states (e.g., empty tables, etc.). In addition to the deployment of data plane components, IML will also deploy the control plane components of the customized network functions and configure the connection between control and data plane components via the flexible P4Runtime interface. In case of multitenant deployment of data plane components, IML will ensure security isolation between the different control plane components responsible for controlling the data plane programs located on the same PDP device. This step also requires further configuration during the deployment. IML will support different deployment flavours. In the case of heavy hitter offloading feature, different instances of the same data plane components will be responsible for serving heavy-hitter users/traffic group and non-heavy hitter ones. For load balancing, two data plane instances will be managed by infrastructure network functions (infraNFs) provided by DESIRE6G balancing the served traffic. In the case of a customized NF with limited capabilities (in terms of throughput, number of users, etc.), scaling up or down the data plane instances can solve the adaptation to the changing network conditions. In DESIRE6G, a load balancing infraNF will be responsible for handling the load balancing between changing number of customized NF instances. All the infraNFs will be configured and managed by the IML. For vertical scaling of containerized NFs, the in-built autoscaling mechanisms of Kubernetes will be applied. We will also investigate if there is a need for vertical scaling of hardware data planes (e.g., scaling up/down the match-action table sizes in run-time) and how it could efficiently be implemented in practice.

In runtime, IML will also enable further optimization of deployed NF data planes via the configuration of different infraNF components and dynamic scaling of software (NF) instances.

2.2.2 IML interfacing the other components

Figure 3 depicts an overview on how IML interacts with other components in the DESIRE6G ecosystem. Note again that the responsibility and scope of an IML instance is limited to a single DESIRE6G site. The different arrows represent the direction of API calls between the components:

- The deployment is orchestrated by the SMO, providing all the information including NF graph partitions, NF and network configurations (incl. transport network configuration to SDN-CP), the service-specific Multi-agent system (MAS), etc. that is needed for the deployment on a set of specific DESIRE6G sites. IML provides extended VIM and network configuration capabilities, enabling the local deployment of data and control plane elements, MAS agents, application services and the logical network links between network function data plane (NF-DP) instances.
- MAS agents can interact with IML in multiple ways: 1) A MAS agent can obtain traditional monitoring data (e.g., CPU usage, memory consumption, etc.) collected in the infrastructure. 2) MAS agents can get information, triggers, events from the pervasive telemetry collector(s), that can be seen as an infraNF, relying on INT. More specifically, the telemetry collectors can feed MAS agents with accurate real-time status information about latency components, end-to-end latency, congestion, etc. 3) MAS agents can also trigger the reconfiguration of the infraNFs via IML to optimize the computational and network resources. (e.g., turning heavy hitter offloading on or off, moving the traffic of a specific user to an alternative path, activating the pervasive telemetry functionality, etc.).
- Some optimizations cannot be done by configuration only. In these cases, MAS agents notifies SMO that can request IML to modify the existing deployment, e.g., to deploy more instances from a given NF.
- InfraNFs are implemented as data plane programs that enable load/traffic-based optimization, tunnelling over a transport network segment and troubleshooting (load balancing, rerouting) or are needed for monitoring purpose (e.g., INT). These data plane programs also require runtime configuration which is implemented by small control plane components inside IML.

- Both the data plane and control plane components of custom NFs are deployed by IML. In runtime, IML will interconnect the data plane instances with appropriate control plane, ensuring unified data plane view for the control plane (even if the NF's data plane is disaggregated) and security isolation by controlling network function control plane (NF-CP) access to the data plane objects. The communication between control and data planes (incl. both IML -- NF-DP and IML -- NF-CP liaisons) relies on the protocol independent P4Runtime API.
- SDN-CP should also have access to IML's routing (RR in the figure) functions to enable inter-site connections. SDN-CP also obtains transport-level network configurations from the SMO.

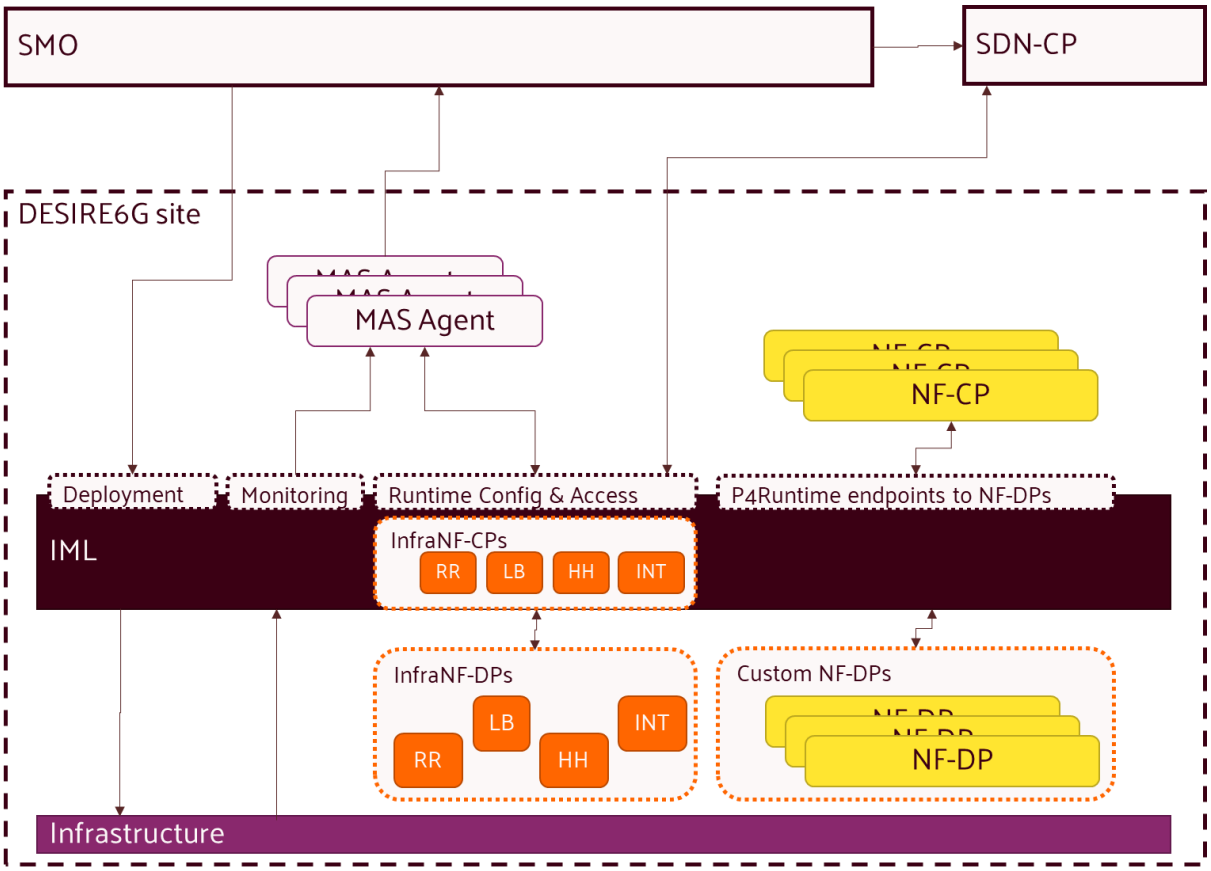

FIGURE 3. HIGH-LEVEL OVERVIEW ON HOW IML INTERACTS WITH OTHER COMPONENTS OF THE INFRASTRUCTURE

3. Main procedures

This section lists the main deployment and runtime related functionality involving the PDP.

3.1 Procedures inside PDP

3.1.1 Runtime load optimization in the data plane

As mentioned previously, IML and the unified data plane layer of DESIRE6G will provide capabilities for optimizing the execution of NF data planes. Specific infraNFs, e.g., load balancing and heavy hitter offloading, have been designed and introduced to optimize the runtime execution. As depicted in Figure 4, the IML hides the implementation details of the data plane and abstracts the NF data plane into a single logical data plane component for the NF control plane. In this way, IML introduces an abstract view of the data plane implementation. In Figure 4, the data plane is disaggregated to multiple instances that may run on different targets. The traffic handler knows what traffic needs to be forwarded to which instance, implementing load balancing or heavy hitter offloading features. The control plane communication is based on P4Runtime as it can also handle non-P4 programmable data planes.

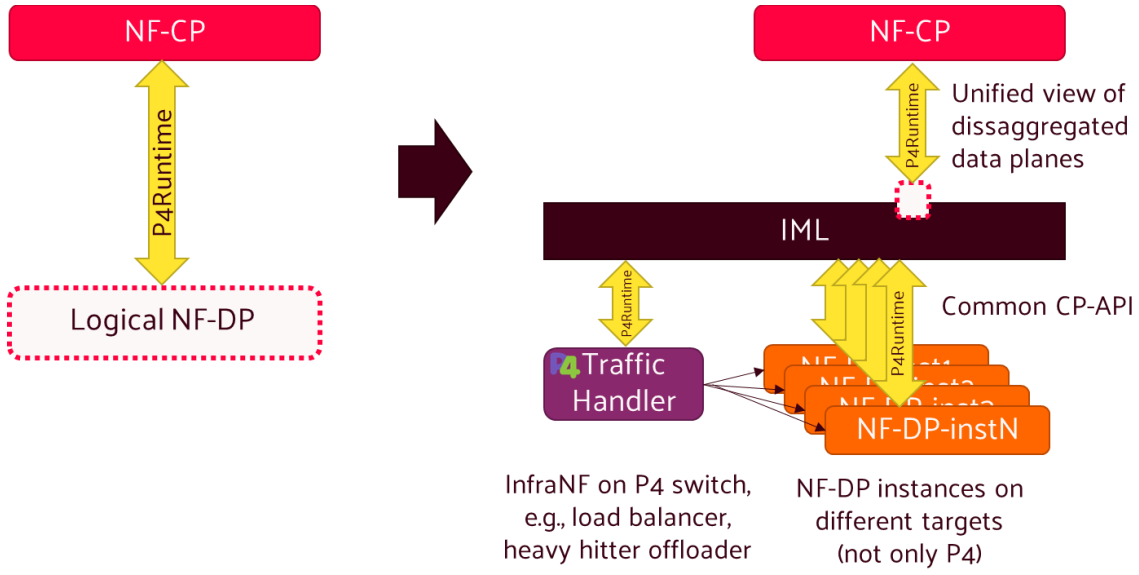

FIGURE 4. DATA PLANE OPTIMIZATION IS HIDDEN FROM NF-CP BY IML

3.1.1.1 Load Balancing

In general, physical network deployments consist of multiple network function (NF) instances for the same functionality, thus they need load balancers to distribute the traffic among them.

Typical load balancers act on some keys calculated based on packet characteristics (e.g., 5-tuple) to route traffic consistently among the different NF instances. This is needed to minimize cache invalidations and packet reordering within one specific flow – both problems could occur with random (non-consistent) packet forwarding. Another method, used by e.g., Kubernetes (k8s) is flow tracking. This is typically used together with a randomization-based load balancer which is used when the first packet of a flow arrives to the system. Upon arrival of said packet, the load balancer selects the appropriate worker instance (in k8s it's called POD or container) then stores the flow parameters in its flow tracking table. This way it is possible to combine the advantages of randomness and consistency with the price of increased memory footprint.

In order to achieve the goal stated in the previous section to hide the data plane implementation details from the control plane, we design a Load Optimizer (LO) method incorporating both control plane (LO-CP) and data plane (LO-DP) components, in accordance with Figure 4. LO-CP is part of IML (implemented as a control plane proxy) and it has the configuration of the required NF instances. The monitoring of NF instances can be used to make a decision on the number of NF instances, as depicted in Figure 5. This is then configured by the LO-CP into the LO-DP. LO-CP then updates the data plane objects of LO-DP and launches provisions for the number of instances using existing APIs like VIM on top of a K8S cluster running containerized network functions (CNFs).

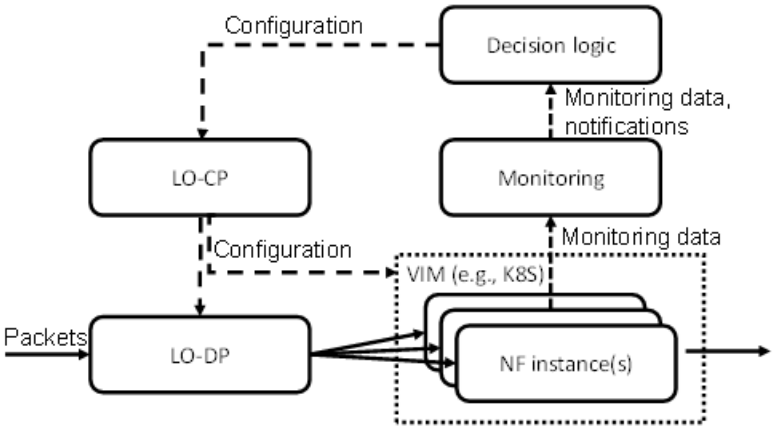

FIGURE 5. A GENERAL APPLICATION SCENARIO FOR LO-CP AND LO-DP COMPONENTS OF THE LOAD BALANCER

3.1.1.2 Load optimization based on heavy hitter detection

There is a softwarization trend in current packet core networks, driven by the need of being able to deliver new functionality in a timely and customized way. This means that future data networks will be more heterogeneous than today, where different variants of certain Network Functions (NFs) may exist on the data plane (DP) to provide more customized performance for specific applications, as shown in Figure 6. For example, general purpose hardware (CPU) based NFs may host high-complexity services that require better programmability. On the other hand, hardware-based NFs (DPUs, smartNICs, GPUs, ASICs, FPGAs, etc.) may considerably accelerate the execution of specific DP functions, which is especially useful for latency critical applications since the CPU-based NFs have unpredictable latency as commodity hardware is not designed for packet processing. HW-based solutions can also reduce the energy consumption. To handle the complexity of heterogeneous data plane targets, new mechanisms, following the cloud system approach, will be needed for making the data plane cloud native (higher flexibility, target/HW independence, transparent load optimization, etc.).

Typically, the deployment of a certain NF (or in case of disaggregated NF of a partial deployment) to a certain hardware or software block is managed by the application itself. An example is Open vSwitch (OVS) that uses a so called “micro flow cache” (uFC) mechanism to speed up the execution of already classified flows. The uFC was initially just a separate table inside the main NF that precedes the main pipeline with a cache containing “heavy hitter” flows to offload the main pipeline. After a while the OVS community realized that this block is rather easy to implement on hardware blocks, so these days smartNIC offload is widely available. But the contents of the uFC is still managed by the application.

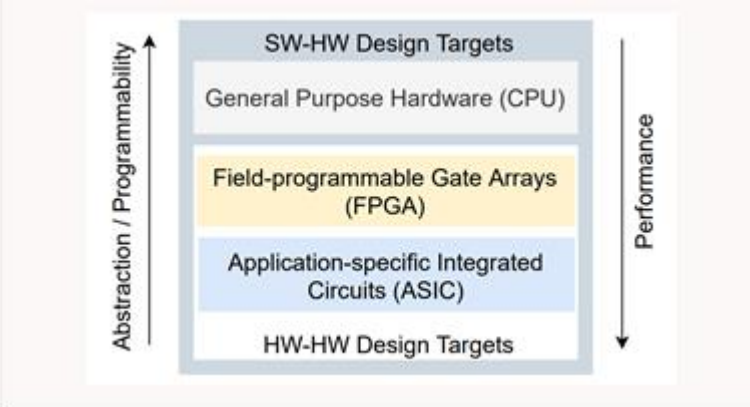

FIGURE 6. DIFFERENT VARIANTS FOR DP IMPLEMENTATION THAT CAN BE USED IN PACKET CORE NETWORKS

One issue that heterogeneous packet-based networks pose is that software and hardware-based network functions typically require custom control and traffic routing. This way, a developer needs to take care of the nuances of the infrastructure, increasing the overall complexity (without focusing solely on the business logic of the function). With the limited accelerators of the past this was acceptable.

With programmable hardware devices it became simpler to have the same functionality on different (SW and HW) targets. In such a system, the transparent selection of what to process on which target is beneficial: it offers similar advantages that we see with cloud-based applications, e.g., automatic scaling, failover, etc. In such cases, multiple instances of the same packet processing functionality are implemented and deployed.

For example, HW-based solutions could be used to handle the so-called heavy-hitters, i.e., users generating high volume of traffic, which may load the CPU heavily, which can fully utilize the CPU cores and thus degrade the throughput of CPU-based NFs. However, CPU-based NFs may still be needed to handle large number of users generating low to moderate traffic, as the HW-based NFs could only handle a limited number of users. A numerical example using hypothetical figures is as follows: assuming a peak traffic load of 1.2Tbps generated by 12k users, this could be handled efficiently by a CPU-based NF with a capacity of 11k users and 0.2Tbps and a HW-based NF with a capacity of 1k users and 1Tbps, given that those 1k users generating the heavy traffic of 1Tbps are properly directed to the HW-based NF (note that if only the CPU or the HW based NF were used, handling the same traffic mix would require 6 CPU-based NFs or 12 HW based NFs, respectively).

In a network with a mixture of different DP implementations (HW-based, CPU-based) of the same NF, it is then a new issue to be solved, i.e., how to select a specific NF implementation to handle specific traffic (e.g., heavy-hitters in our case) in a way that is transparent to the NF business logic and also the control plane. Fort this, there is a need to create an automated data plane mechanism separating the load optimization of heavy hitter offloading from the data plane logic of NFs, allowing the developers to deploy their NF implementations without taking care of load balancing between heavy hitting (HH) and nonHH traffic. Therefore, a solution in Figure 6 is desired, where the control plane of an NF (NF-CP) sees a single logical data plane component to be controlled (left side) and is unaware of the load optimization mechanism implemented.

The Load Optimizer (LO) is a middlebox processing network packets and forwarding them to one of the NF instances. LO is composed of two components: a data plane component (LO-DP) implementing the packet processing logic in a packet processing digital circuit and a control plane component (LO-CP) that is part of IML and runs on a general-purpose computing unit (CPU). LO-CP and LO-DP are located close to each other (e.g., LO-DP is implemented by the switching ASIC while the LO-CP runs on the switch CPU).

We also assume that the network packets are tagged at some points of the network (e.g., by the UE2Service mapper infraNF). The tags carry information about how LO-DP needs to handle the packet, e.g., how traffic is grouped (per user, per application, per QoS class, etc.), activity of the given group (heavy hitter user or non-heavy hitter user, active user or passive user, etc.). Appropriate tagging may require additional methods (e.g., heavy hitter detection) that have recently been studied in [1], [3]-[5] . Tags are stored in additional header fields (e.g., in a DESIRE6G header).

Figure 7 illustrates the case when the traffic is tagged at both upstream and downstream direction. Both marking include two fields: a load balancing key that is used to identify packets grouped together (traffic of a UE, an application, etc.), and a HH-flag denoting if the packet is generated by a heavy hitter (i.e., a user/UE/application generating higher traffic volume than others). Other fields may also be added, depending on the applied load optimization methodology.

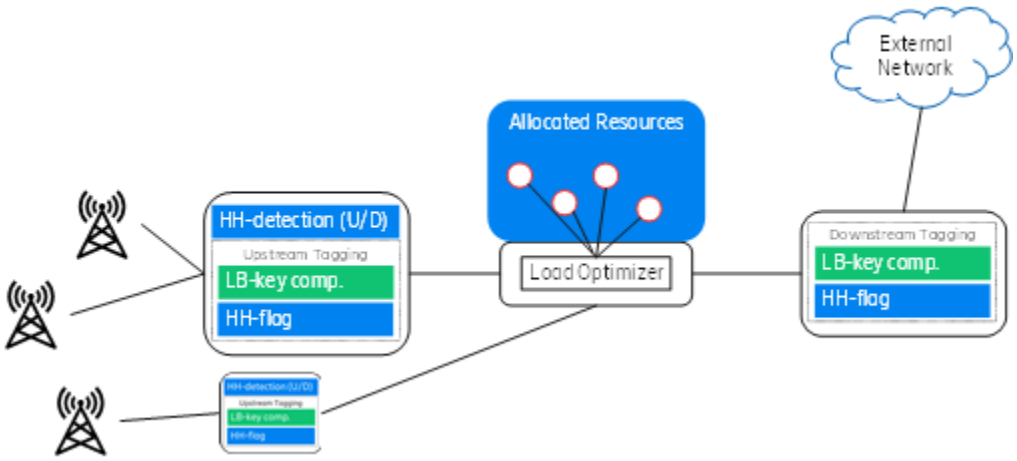

FIGURE 7. THE PLACE OF LOAD OPTIMIZER IN THE FORWARDING PATH. PACKETS ARE TAGGED IN BOTH UPSTREAM AND DOWNSTREAM DIRECTION. TAGS ARE USED AS INCENTIVES ON HOW TO HANDLE/GROUP PACKETS IN THE DATA TRAFFIC.

As denoted in Figure 7, we assume that heavy hitter (HH) detection is performed somewhere in the network. Two extra fields as tags are carried with the packets:

- key that is used for identifying packets that belong together (e.g., to the same UE, application, flow, etc.). Packets with the same key are handled by the same data plane instance of the network function to be executed.
- HH-flag that is only set if the traffic intensity of the group identified by the key is larger than most of the other traffic groups (i.e., UEs, applications, etc.). The HH state is non-static property and can dynamically change over time. For example, when a user (UE) does some web browsing is non-HH, while if he/she starts playing a cloud-rendered game with 4K video the traffic is marked as HH. After finishing the game, the traffic volume goes back to normal, and the packets will not be tagged with HH.

A high-level functionality for heavy hitter handling is shown in Figure 8. The digital circuit of LO-DP contains a packet parser that can parse the extra fields: key and HH-flag tags. It also has a lookup table that maps keys of HH packet groups to a routable location identifier of a HH node, or that of a non-HH node (alternatively, to a load balancer distributing among different non-HH instances).

After parsing the packet, LO-DP identifies the HH flag's value. If the flag indicates a non-HH packet, then it is forwarded to the nonHH path (this assumes that the nonHH NFs always have the states needed to process any traffic flow). If the HH flag indicates a non-HH packet, however, then LO-DP will only forward it to the HH node in the case when it has a HH NF-DP entry in its lookup table for this key. This entry is provided by the LO-CP, after receiving notification (e.g., from a traffic classification infra-NF) on some traffic aggregate (denoted by a key) becoming HH traffic. After the receipt of such notification, LO-CP selects a HH NF-DP node, i.e., a programmable HW-based NF-DP that may efficiently handle HH traffic and installs the required states for this key to this HH NF-DP, and then it also updates the entry corresponding to this key LO-DP to point to this HH NF-DP. In this way it is ensured that only that traffic is forwarded to the HH node that has the required states for packet processing. In order to cope with the limitations for the HW-based NF implementations, the states for the traffic that becomes nonHH are also removed from the HH NF-DP (and the corresponding entry from LO-DP) by the LO-CP.

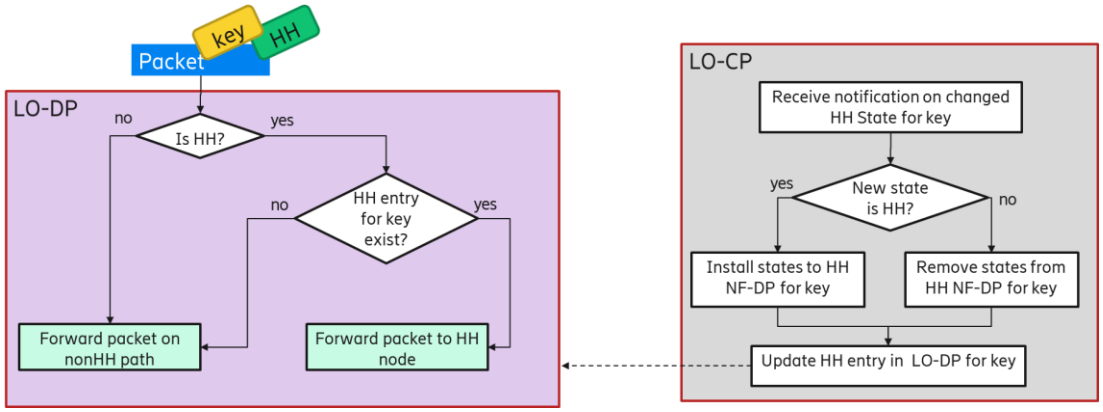

FIGURE 8. FLOWCHART SHOWING THE HIGH-LEVEL FUNCTIONALITY FOR HEAVY-HITTER HANDLING FOR LO-DP AND LO-CP, RESPECTIVELY

The investigation of the heavy-hitter detection and treatment will continue with a deep-dive study on how to implement the required functionality for the LO-DP in P4 hardware with limited CPU and memory resources. Further study is also needed to understand potential optimizations related to a seamless state handling by the LO-CP, including dynamic states.

3.1.2 Quality of Service

Ensuring quality of service requirements of different traffic groups is a two-fold problem: 1) routing along a path that satisfies the requirements and rerouting if needed, and 2) traffic management in individual network nodes to handle access to link resources and latency requirements in case of congestion. Routing requires higher level information and in DESIRE6G it can be handled by the SMO and the MAS. The MAS can also trigger reallocation of data flows to a better path, or allocation of resources (e.g., scaling computing resources) in runtime if the service requirements are not met. This method only requires minimal data plane support for implementing the respective logic. Note that MAS can react to changes in time scales of milliseconds (near-real time), but handling network congestion requires faster mechanisms that can operate at the time scales of packet forwarding (microseconds or below; real-time). Thus, traffic management needs to be implemented by the data plane.

Novel applications and network scenarios challenge existing traffic management strategies. Hierarchical Quality of Service (HQoS) provides a fine control of resource sharing and buffering delay, but traditional HQoS solutions have challenging complexity that prevents their deployment at scale in the traffic management engine of high-speed switches. Ensuring strict performance isolation per user

requires as many packet buffers as the number of users with a deficit round-robin scheduling between them. Most hardware data planes only allow to define limited number (e.g., 8 in Tofino-1) of queues for each egress port that is far less than the number of users. Though software data planes can overcome this limitation, their packet processing performance is limited. Programmable switches have emerged to make the packet processing pipelines flexible and reconfigurable, but their traffic management capabilities still rely on fixed functions that cannot handle complex HQoS policies.

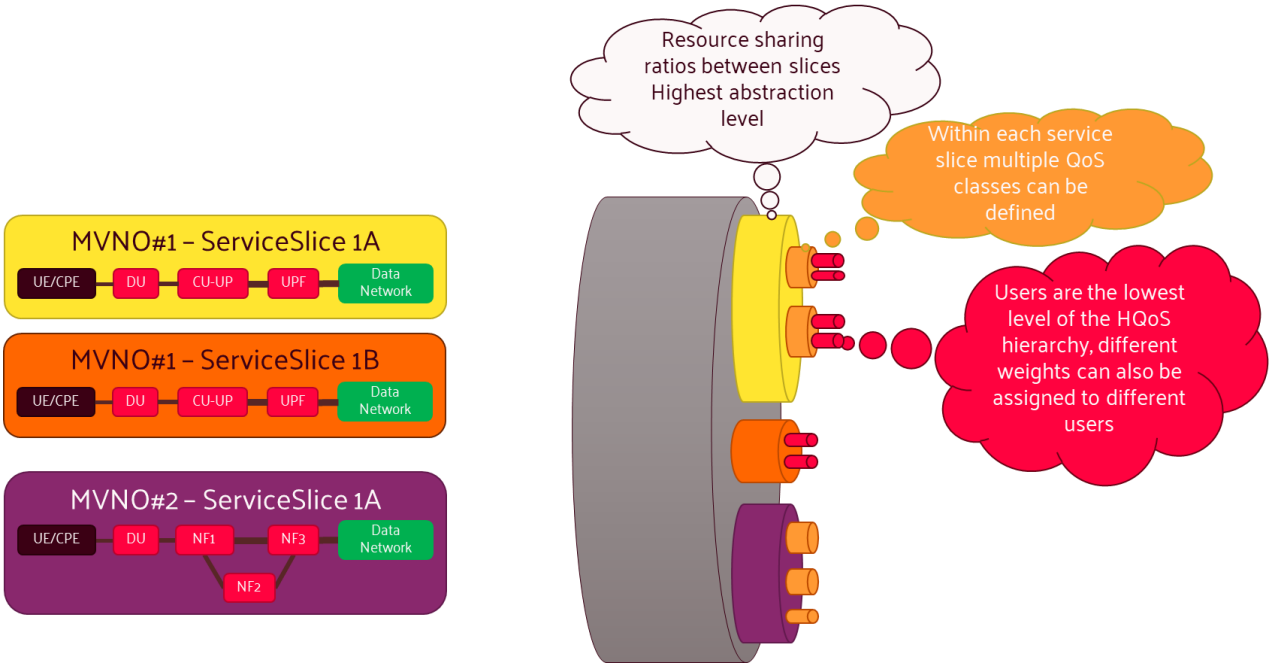

FIGURE 9. NETWORK RESOURCES SHARED BETWEEN THREE NETWORK SERVICES. FURTHER SPLIT IS SUPPORTED TO SUBSLICES AND UES IN A HIERARCHICAL WAY.

With the advent of programmable data planes, new programmable traffic management approaches have appeared. Most such solutions compute a rank or a value for each packet according to a predefined policy, and use these tags to determine the place of the packet in a buffer (e.g., PIFO [6]) or make an admission control decision dropping packets with less important rank values if the buffer occupancy is above a threshold (AIFO [7], RIFO [8], PPV [9]). Admission control-based methods have lower computational complexity and could be implemented on programmable hardware data planes like Intel Tofino as they only require some filtering rules to be applied at ingress and normal FIFO queues assigned to the egress ports of the device.

In DESIRE6G, traffic management needs to handle HQoS resource sharing policies illustrated by Figure 9. The highest level in this hierarchy represents the network services (or service slices) where the

resource sharing requirements between the services need to be defined. The policy on this level is described as a combination of weighted fair and strict priority scheduling. Each service slice can however define multiple layers of QoS classes with additional resource sharing rules. The UEs in one of the bottommost QoS class can also be handled differently (e.g., by assigning different weights to them). Some applications require low latency transport that at traffic management level can be guaranteed with the use of L4S (Low Latency, Low Loss, Scalable Throughput) technology introducing separate buffers with different congestion signals for classic and low latency L4S traffic [10].

3.1.3 Approaches ensuring Quality of Service

Novel rank/value-based approaches have been proposed in the past few years as more flexible alternatives to traditional solutions where the policy is expressed as weighted fair, strict priority scheduling or their combination and applied on each link in the network independently. Rank or value-based methods have the benefit that the policy is encoded into a single value that can be computed once and assigned to the packet (e.g., storing it in a dedicated header field in the DESIRE6G header. Note that a DESIRE6G header is a packet header in a DESIRE6G domain that contains fields that help the packet identification and proper packet handling). At any potential congested links, the carried tags can be used to decide how to handle the packet. For example, the Per Packet Value (PPV) method proposed in [11], [12] (see Section 3.1.3) requires a 10-bit long header field to store the packet tag and can implement HQoS policies with several QoS sub-classes and even isolate the resource usage of end-users. An additional benefit of this method is that it optimizes the resource allocation at network-level (ensuring a hierarchical max-min weighted fairness) which is much closer to the expectations of both service operators and end-users than the link-level focus of traditional methods. The method can also handle low-latency and normal traffic assuming L4S congestion control in case of low latency sources.

The application of this method in the DESIRE6G network however requires that all the potential bottleneck links implement the value-based scheduler. Alternatively, it can be deployed partially in DESIRE6G edge sites, where resources are scarce, but in that case the translation of rank/value-based policies to the policy description of other traffic management solutions is needed. Moreover, using PPV for HQoS still faces the typical constraints on scale and complexity for enforcing per-flow policies. To this end, the approach can be complemented with the use of virtual queues in the

programmable pipeline [13] . Virtual queue-based traffic management schemes (e.g., AQM [14], scheduling [15] etc.,) have been widely employed for legacy switching devices. In this case they can be used to enable rate limiting per flow or set of flow(s). Assuming that different sets of flows correspond to different network slices, such customization can be performed on a per (service) slice basis, allowing for dynamic (inter) slice traffic management, enabling different levels of (guaranteed) performance isolation, depending on the fixed function capabilities of the programmable network devices, as indicated in [13] .

Ensuring QoS on non-programmable network segments.

Both virtual queues and rank/value-based methods can only be used in network domains with programmable data planes as these methods are not supported by commercially available fixed-function switches. If a DESIRE6G deployment contains non-programmable network segments, these methods cannot work, and other techniques need to be involved to ensure the desired quality of service level. Google proposed a resource sharing framework called BwE [5] to manage the flow rates in its private WAN between the data centers few years ago. This method can achieve the same network-wide resource allocation as the PPV method but while PPV is fully distributed, BwE follows a more centralized approach. It relies on continuous flow rate monitoring and periodically executes a centralized rate allocation method on the network topology, using the rate measurements with the known routing. The allocated rates are then enforced by policer nodes deployed close to the traffic sources. The method also implements lots of heuristics that are not needed in PPV to make the method capable of handling throughput fluctuations more elastically. One big advantage of BwE is that it cannot require complex traffic management methods in the network. The data plane only needs to implement flow (or UE-level) rate monitoring and traffic shaping. All the complex computations are performed periodically on a server. Note that pure data plane solutions like PPV can react to changes in the network conditions much faster than the BwE where the update time is at the time scales of seconds. In DESIRE6G, we investigate how the programmable traffic management approaches like virtual queues and PPV can be extended to handle rate allocations on non-programmable segments. To this end, we investigate how methods similar to BwE can be used to enforce DESIRE6G policies over selected part of the network. At the entry points of non-programmable segments, policer nodes need to be deployed to measure the users' rates and enforce the policy by traffic shaping or DSCP marking (also supported by most

commercial switches). The rate allocation is then computed by a dedicated resource where the measured load, the topology of the non-programmable segment, link capacities and the policies to be applied for the users' traffic are all taken into account.

In DESIRE6G, the applicability of the different methods has been investigated, proposing a solution that can be integrated with other mechanism of the architecture (e.g., MAS-based decisions).

3.1.4 NF routing

NF routing (NFR) is a key component of the PDP layer. The "service ID" can be put into the DESIRE6G header, helping the NF Routing (NFR) function. But this alone is not enough. Many users can share the same service and their site mappings can be partially different. In uplink direction routing is more unambiguous (although in some cases there can be still more sites to choose from), while in downlink direction to find the actual edge and/or RAN site the user is active at the NFR must make a lookup on the "user ID" (e.g., IPv6 address) too. Note that this is only valid when the NFR notices that the rest of the graph is on a different site. Until that step simple "service ID" based routing is available between the NFs.

As the next NF does not depend only on the service (especially in downlink direction), but on the location of the UE, NFR also needs to set a "location" parameter together with the "service ID" when the packet first enters the DESIRE6G system, similarly to the concept of location/ID separation in LISP [16] or SRv6-based NF-chaining approaches [17] . This way the inter-site lookup can be simplified: instead of a "user ID", the "next site" mapping of the NFR can use the "location" parameter, which itself is a RAN site, but can also be used as a lookup input for intermediate (edge) sites – here we assume that the proper intermediate site can be selected if the user location is known. If that is not the case, as e.g., there are multiple intermediate sites equally equipped to serve a given location. Then, the selection can only be made with the help of some load balancing / equal cost routing function. As we allow heterogeneity in the transport network, we need a lightweight solution. Introducing a new DESIRE6G header with service specific fields could be one option in addition to SRv6 with CIDs or uCIDs. The NF routing concept is depicted in Figure 10.

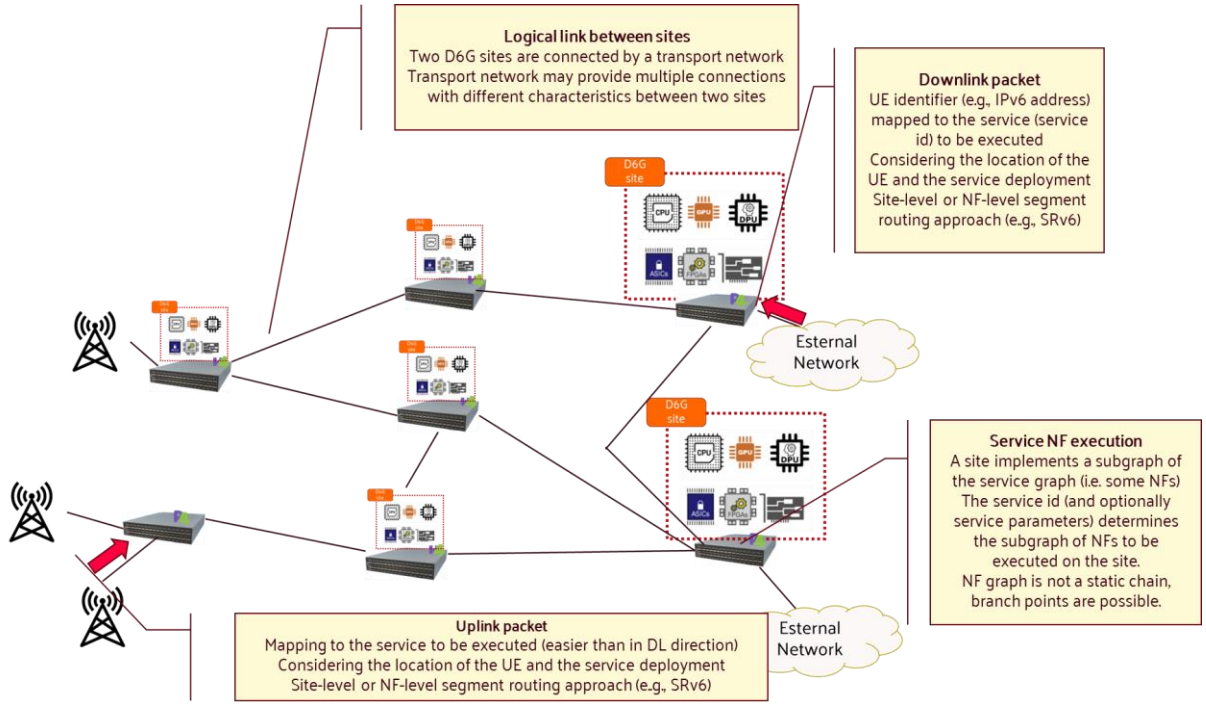

FIGURE 10. NF ROUTING ACROSS DESIRE6G SITES.

3.2 Procedures involving PDP

3.2.1 NF deployment on a single DESIRE6G site

IML of DESIRE6G will extend the capabilities of a Kubernetes cluster (providing an extended VIM interface). It will allow the deployment of NFs on both software and hardware data planes and the creation of virtual links between heterogeneous NF instances implementing the network service graph. In DESIRE6G, we assume that each NF consists of a packet processing NF-DP and a low-level control entity called NF-CP. NF-CP is a software component responsible for filling or reading the objects (e.g., match-action tables, counters, registers) in the NF-DP data plane. NF-CPs will be deployed as Kubernetes pods/containers [18] using the existing mechanism. The deployment of software NF-DPs (containerized NFs) will also rely on the existing mechanism of Kubernetes. For PDP hardware, the deployment of NF-DP binaries will be done through the remote execution framework of vAccel as depicted in Figure 11. IML will prepare the NF-DP binary. In case of multitenancy on P4 targets it requires program aggregation as shown in Section 4.3 and the configuration that is sent to a PDP HW-specific vAccel Agent. The agent implements the deployment and configuration procedure for the given platform and returns with status

messages. Note that agents need to be implemented for all the possible PDP HW used in DESIRE6G. The configuration procedure includes the setup of the corresponding local control plane (i.e., P4Runtime) interfaces [19]. The logical links between NF-DPs will be implemented by tunnelling methods or other methods like NetworkServiceMesh.io (NSM) [20] enabling the definition of multiple interfaces for service chaining between CNFs. We will extend these methods with the support of NF-DPs running on PDP HW.

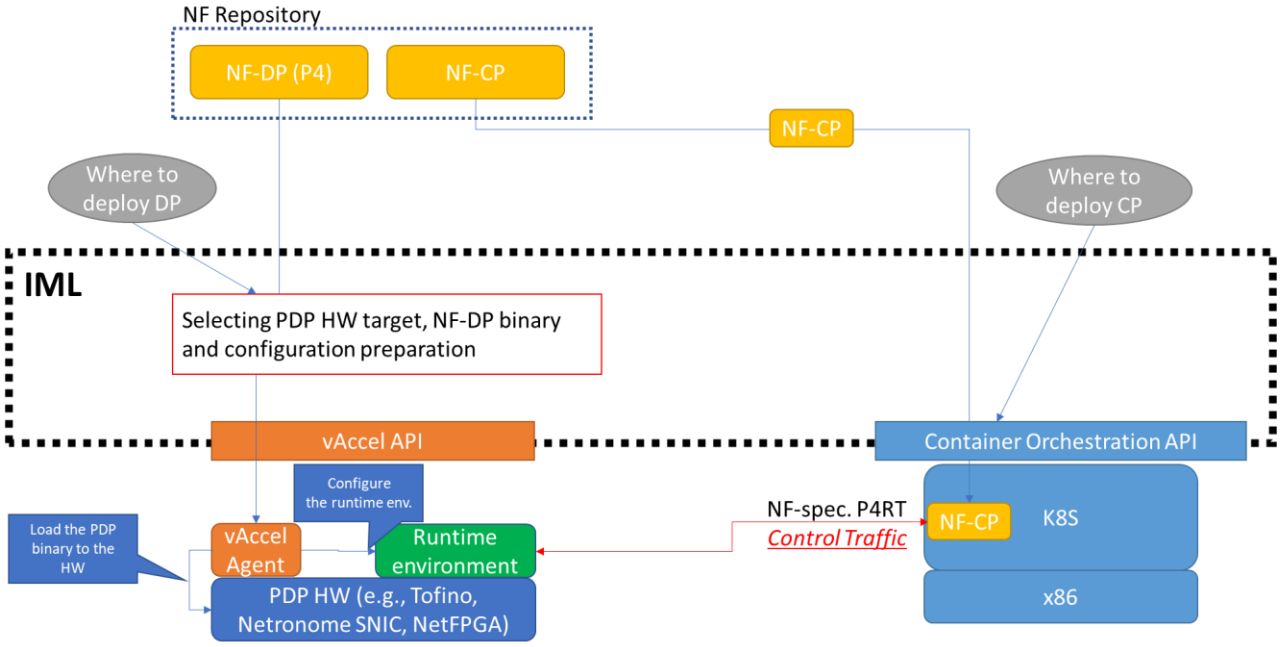

FIGURE 11. NF DEPLOYMENT ON PDP HW THROUGH VACCEL.

3.2.2 Application Function deployment on a single DESIRE6G site

vAccel [21] is a software framework that enables workloads to access hardware accelerators securely and efficiently. vAccel enables developing hardware-independent applications by logically separating an application into two parts: (i) the user code which is part of the application logic itself and (ii) the hardware specific code which is the part of the application that runs on a hardware accelerator. Additionally, vAccel enables hardware acceleration within virtualized guests by employing an efficient API remoting approach at the granularity of function calls, to delegate accelerable code in a vAccel agent on the host system.

The semantics of the transport layer are hidden from the programmer. A vAccel application that runs on bare metal with an Nvidia GPU can run as is inside a VM using the appropriate VirtIO backend plugin.

We have implemented the necessary parts for our VirtIO driver in our forks of QEMU [22] and Firecracker [23] hypervisors.

Additionally, we have designed the above transport protocol over sockets, allowing vAccel applications to use any backend, if there is a socket interface installed between the two peers. Existing implementations include VSOCK and TCP sockets. Any hypervisor supporting virtio-vsock can support vAccel.

To facilitate the deployment of vaccel-enabled applications, we integrate vAccel to a popular container runtime, kata-containers [24].

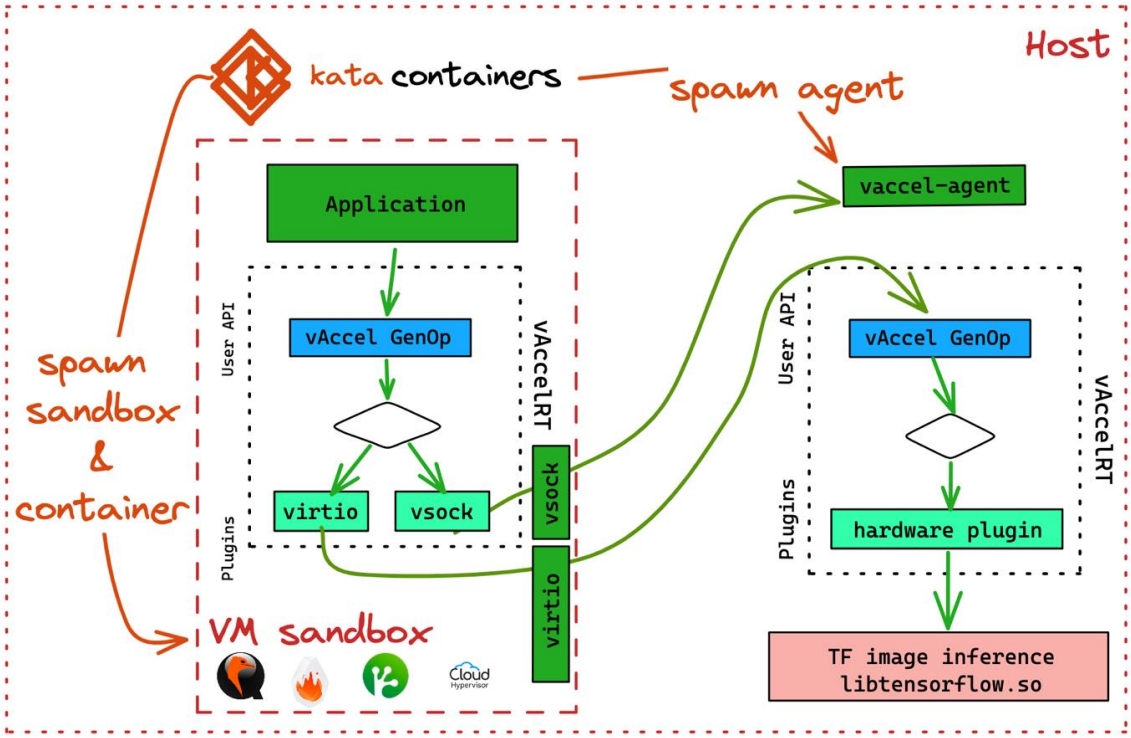

FIGURE 12. VACCEL INTEGRATION WITH CONTAINER RUNTIMES

vAccel integration to kata comes in both modes: virtio and vsock. An overview of the software stack is shown in Figure 12.

Our current downstream implementation for Kata-containers v3 includes support for both the AWS Firecracker sandbox and their custom, tailor-made Dragonball backend, using the vsock mode of vAccel.

In DESIRE6G we enhance vAccel to address the challenge of deploying hardware-accelerated components of user applications in various and diverse hardware architectures. Coupled with SOL

(Section 6.1.4), we enable efficient and transparent deployment of accelerated application execution throughout the continuum of DESIRE6G Core, and Edge sites and equipment (SmartNICs, Switches, Generic x86 servers and ARMv8+ edge hardware equipped with GPUs).

3.2.3 Monitoring-based run-time optimization

In DESIRE6G a distributed decision-making solution will be considered. In particular, a DESIRE6G service, will be deployed by the SMO, relying on IML. The deployment includes the distributed agents to perform the near real-time control of the different segments of the service (i.e., different DESIRE6G sites).

In the same phase, all the required components for the telemetry-based monitoring are enabled. This includes a set of modules (i.e., data producer, data aggregators and data consumers) that are responsible for the data collection over the end-to-end instantiation. As an example, telemetry modules can be enabled at the RAN segment, retrieving the service metrics from the virtualized RAN NF (i.e., per-UE or per-slice statistics) or in the networking domain, activating the generation of per-packet data collection, exploiting INT data augmentation. In the latter case, the telemetry data can be augmented with service specific data to be used for the deep service monitoring and /or the in-network actuation of re-optimization. This includes the per-packet conditional traffic steering (i.e., forwarding) according to the metadata information included in the augmented packets. In DESIRE6G the telemetry data is envisaged in order to feed the distributed decision-making nodes with proper data, being capable to make autonomous decisions as a function of the observed conditions.

Figure 13 shows an example of the interactions among the IML, the MAS and the monitoring modules for the execution and optimization of a service. In the example, a service composed by virtualized RAN NF (that can be accelerated in the PDP devices), PDP networking and cloud entities (in a K8s environment). Along this document, MAS functionalities are depicted separately (radio, telemetry, cloud, packet) for the sake of clarity, but a possible solution, where possible, is that all the metrics will be aggregated in a single agent per service and per site.

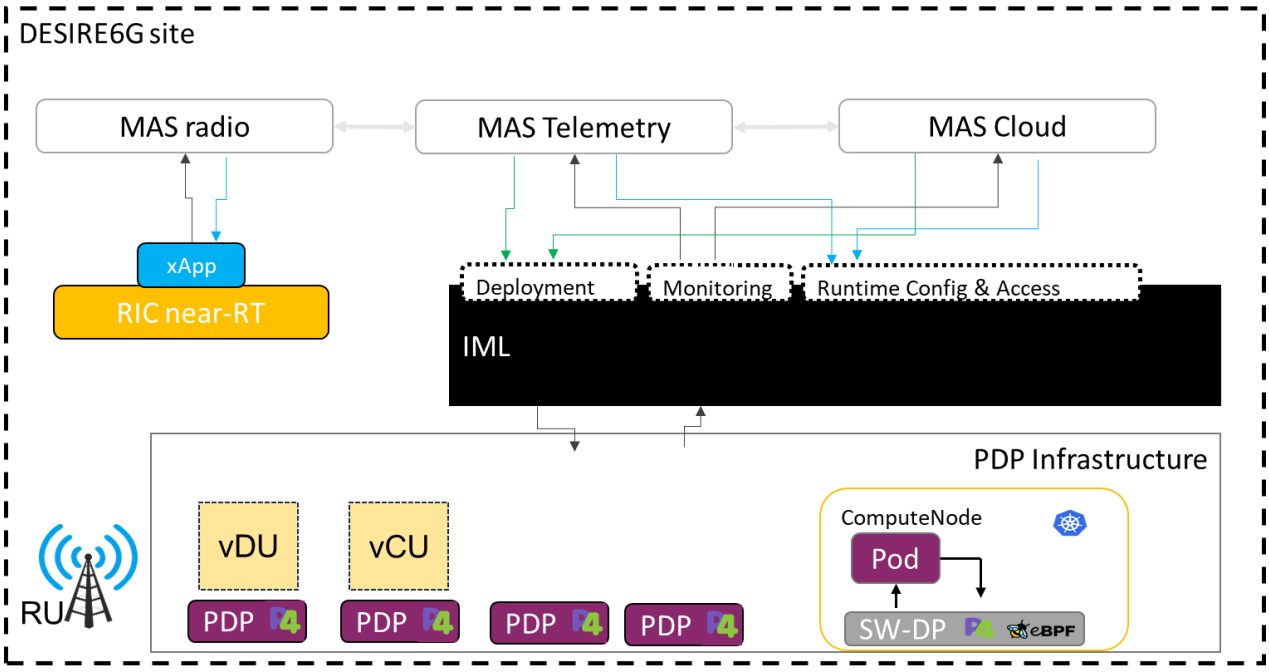

FIGURE 13. IML, MONITORING. MAS INTERACTION.

The IML is responsible for the deployment of the different PDP element and their runtime management. Telemetry data flows from the PDP elements towards the pertinency MAS agents. The agents are defined to aggregate and distribute collected data in order to optimize the end-to-end service operation. In the example, the networking metrics, collected via P4 collectors can be aggregated and shared with the telemetry MAS agent and/or cloud MAS agent by the Network MAS agent., enabling the distributed and coordinated service re-optimization. The radio MAS agent can be also fed with the data collected in the radio segment, including UE metrics and/or slice metrics, collected via xApp,

In case of service degradation, the MAS agents identify the possible cause of the degradation and attempt to re-optimize the end-to-end service. Possible re-optimizations are the network rerouting, the radio adaptation, that require the modification at runtime of the current configuration or the NF scaling, with the deployment of new instance of the affected NF. If needed, the MAS agents communicate with IML, requesting the desired re-configuration.

4. Deep slicing support

The deep slicing support for PDP hardware will enable MVNOs to deploy, configure and control their customized data plane pipelines, called network services in the DESIRE6G network, in an isolated way. Deep slicing support for PDPs relies on two key pillars: 1) multitenancy on PDP hardware enabling the deployment of different data path services on the top of a shared PDP infrastructure (Data Path Service Slice) and 2) flexible creation and definition of QoS slices and subslices in a hierarchical way (i.e., HQoS).

4.1 Network Service Deployment in the DESIRE6G Infrastructure

As mentioned previously, DESIRE6G defines the network services implemented in slices as an execution graph of NFs. A network service often covers multiple sites, and some NFs will be deployed close to the base station or edge, while others can run in the core cloud sites. The deployment process will start at SMO level as shown in Figure 14. SMO will have a service graph repository and policy framework that are used for determining the DESIRE6G-site-level location of different NFs in the service-graph. Accordingly, the graph will be split into multiple partitions, each partition will be mapped to one or more DESIRE6G site(s), and service requirements (e.g., latency) will also be assigned to the generated subgraphs and the logical transport links. According to the site allocation, transport requirements and the partitions the high-level transport network configurations will be prepared and provided to SDN CP for configuring the inter-site connections in the DESIRE6G infrastructure. As a result of partitioning and site allocation, the service-subgraphs with the service requirements will be forwarded to the IML instances responsible for managing the local resources of DESIRE6G sites. IML only handles local resource allocation based on the requirements delivered together with service-subgraph and the known properties/characteristics of the PDP targets available in the site. This task also includes the selection of PDP targets, the number of NF instances needed to serve the demand and the configuration of infraNFs (e.g., load balancer, heavy hitter offloader, etc.). The NF-DP of each NF in the subgraph is then deployed on the selected PDP HW or SW environments. The corresponding NF-CPs are also deployed on the local Kubernetes cluster. Finally, the logical links between NF-DP instances are set by IML. Note the inter-site configuration is done by SDN-CP.

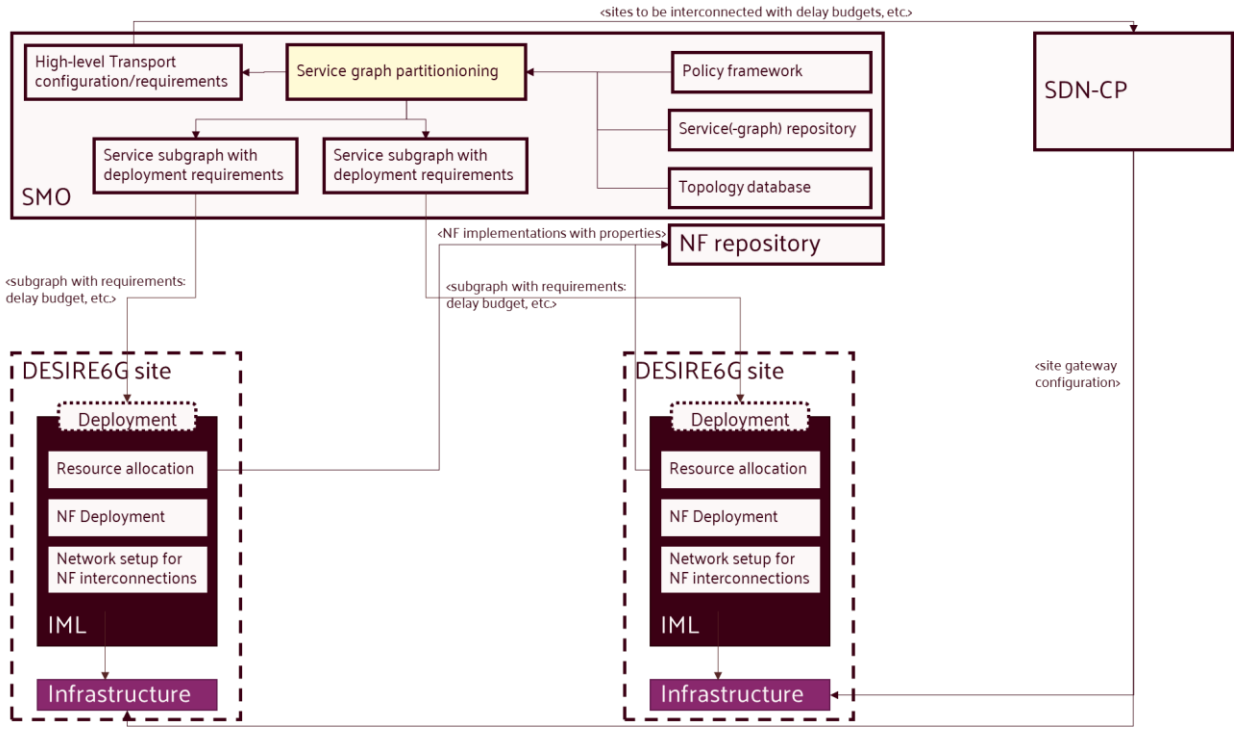

FIGURE 14. NF DEPLOYMENT PROCEDURE.

4.2 Network Service deployment on a single DESIRE6G site

As defined in 87[2] IML is responsible for the deployment of data plane functions in a given DESIRE6G site. In DESIRE6G a data path service slice is described as a network function graph that needs to be executed during packet forwarding from source to the destination. As an end-to-end path can cross multiple DESIRE6G sites, the data path graph will be split into segments and mapped to specific sites. Each segment will be deployed by the site-local IML instance. An example deployment of a data path NF graph segment is depicted in Figure 15. Accordingly, the site consists of traditional compute nodes (e.g., Kubernetes cluster) and PDP elements including smartNICs, ASICs and software data planes on the compute nodes. IML will extend 1) VIM functionalities to handle the deployment on PDP hardware including ASICs and SmartNICs and 2) networking functions to create virtual links between data plane elements. Virtual links will be realized as tunnels (VXLAN, GENEVE or SRv6) between the data plane components. The figure also shows an example data path where the packets go through various NF data planes deployed on heterogeneous hardware (ASIC, smartNICs, software data plane on the hosts and Kubernetes pods). InfraSwitches are also P4 programmable devices but they only host infraNFs that

can be customized by configuration without the need for modifying the data plane program. In contrast, AcceleratorSwitches can host custom NF data planes written in P4.

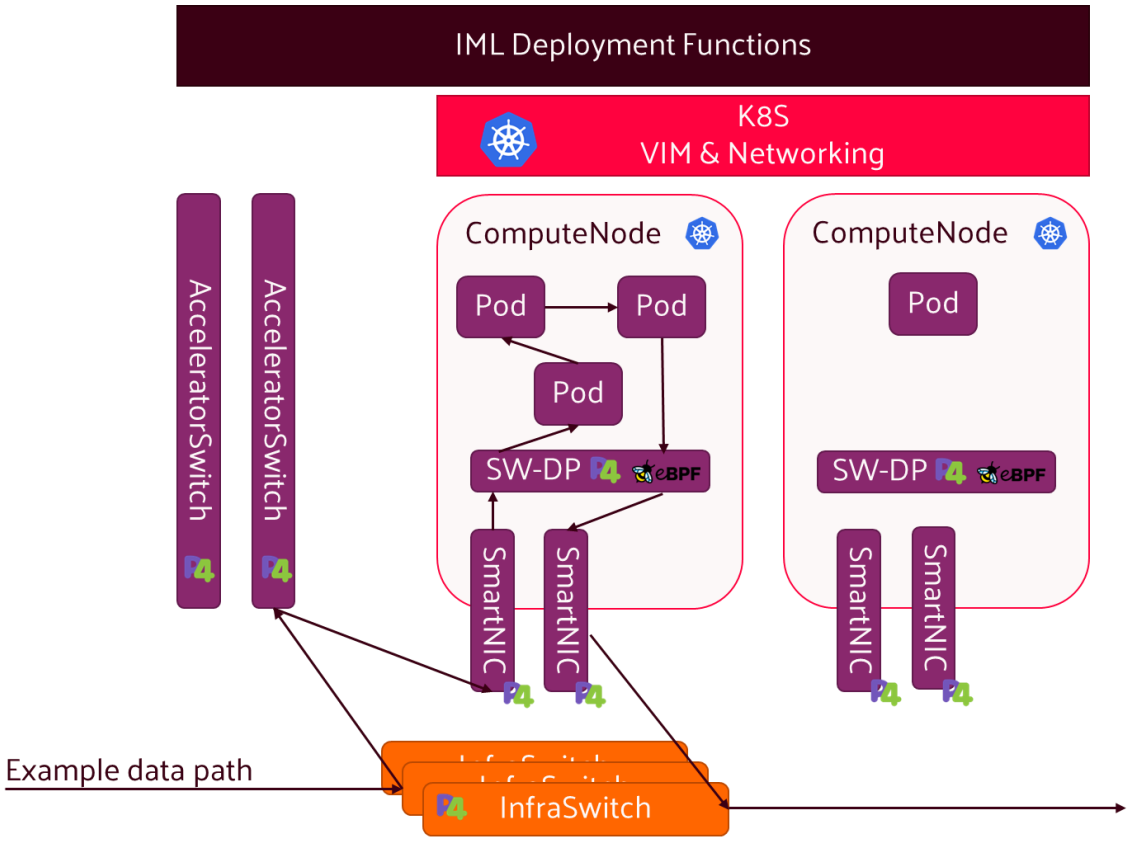

FIGURE 15. EXAMPLE DATA PATH SERVICE SLICE DEPLOYMENT ON A SINGLE DESIRE6G SITE

4.3 Multitenancy support for P4 data planes

PDP devices like Intel Tofino can only execute a single P4 program at a given time. To support the execution of multiple data plane programs (i.e., multitenancy), we develop a P4 code-aggregation method that will merge multiple P4 programs into a single data plane, step 2 in Figure 16. After compilation, the aggregated P4 program then can be loaded to a Tofino device (step 3) and an aggregation proxy is configured. After this, the previously deployed network functions lose their internal states. To make the changes seamless, the aggregation proxy restores the states from its persistency layer (i.e., database). For the newly added network function data plane the proxy creates a service end-point (step 5). Then control plane NF-CP1 is instantiated and configured (step 6). Finally, the control plane

connects to the proxy's service end-point. Note that the proxy ensures security isolation between the different control planes, ensuring that each control plane only sees the data plane it is responsible for. In addition to the vertical aggregation required by multitenancy support, the aggregation method we work on will also support horizontal aggregation of two P4 pipeline to be executed sequentially. This can also help in better utilizing data plane resources on hardware data plane and preventing unnecessary I/O operations in software P4 targets. Note that the aggregation method will be extensible and could also be used with other P4 targets.

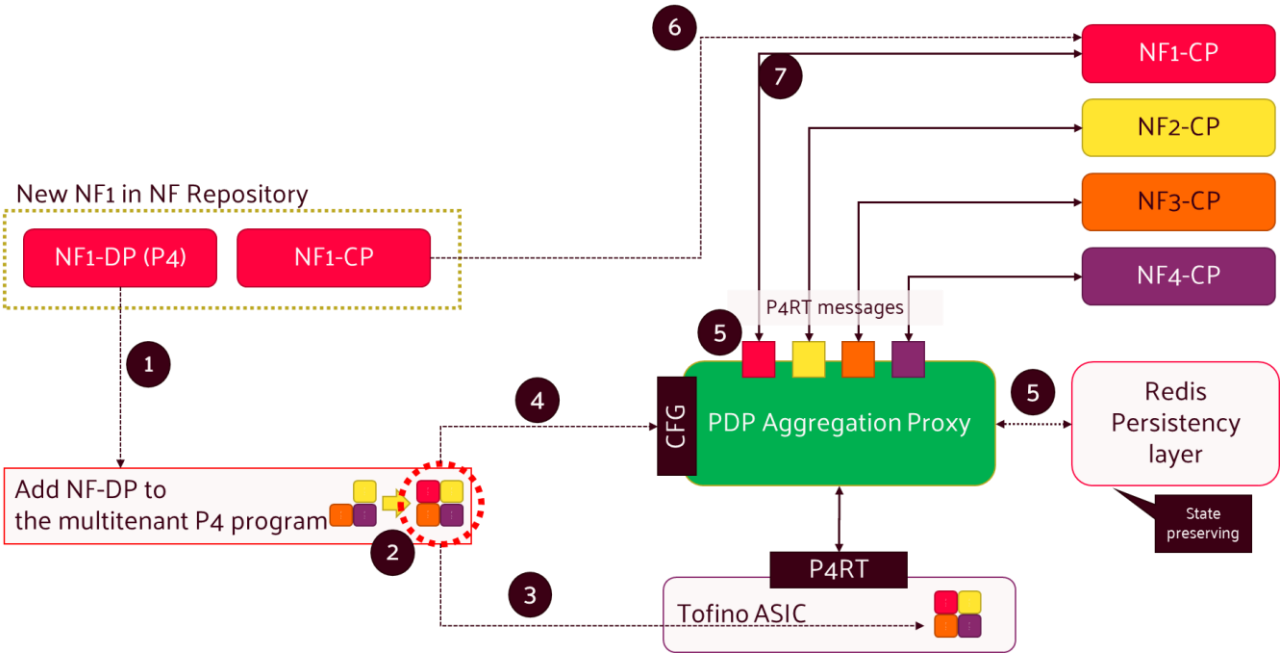

FIGURE 16. MULTITENANT DEPLOYMENT OF AN NF DATA PLANE ON P4-PROGRAMMABLE TOFINO ASIC

5. Pervasive monitoring

The success of the evolving 6G systems will depend on one hand on ubiquitous network intelligence to support comprehensive network automation, and on the other hand the effectiveness of network control to meet the stringent performance requirements of 6G services. Pervasive intelligence, automated management and (near) real-time control call for an equally pervasive and real-time monitoring system. However, monitoring has been so far applied without providing a comprehensive framework with global visibility on all resources, efficiently supporting scalable network management and control at different timescales. The DESIRE6G system leverages on the concept of pervasive data-driven monitoring to support zero-touch control, management, and orchestration. Monitoring data can originate from either service elements, adhering to the DESIRE6G definition of the E2E service or the infrastructure.

In DESIRE6G a distinction among Service Monitoring and Infrastructure Monitoring is considered. Service Monitoring focuses on the performance and availability of specific services or applications that are crucial for business operations. The data collection granularity can be very high, enabling the close loop operations to guarantee the Service Level Agreements. Infrastructure Monitoring encompasses the monitoring of the underlying hardware, software, and network components that support the services. It involves generalized metrics representing the health and performance of servers, storage, networking equipment, databases, and other infrastructure elements.

5.1 Reference scenario

This section describes the reference scenario of an application/service considered in the design of the pervasive monitoring system for DESIRE6G. In Figure 17 a simplified representation of the scenario including a multi-site DESIRE6G environment is highlighted. More specifically, the scenario involves the User Equipment (UE) attached to the RU at DESIRE6G site 1. The UE communicates with a ServiceApp, running in a different premise, e.g., at DESIRE6G site 2. Stemming from the DESIRE6G use cases, we consider service apps such as an AR/VR app or a Robot Digital Twin app (for more details on the DESIRE6G use-cases, please refer to the deliverable D2.2 [25]).

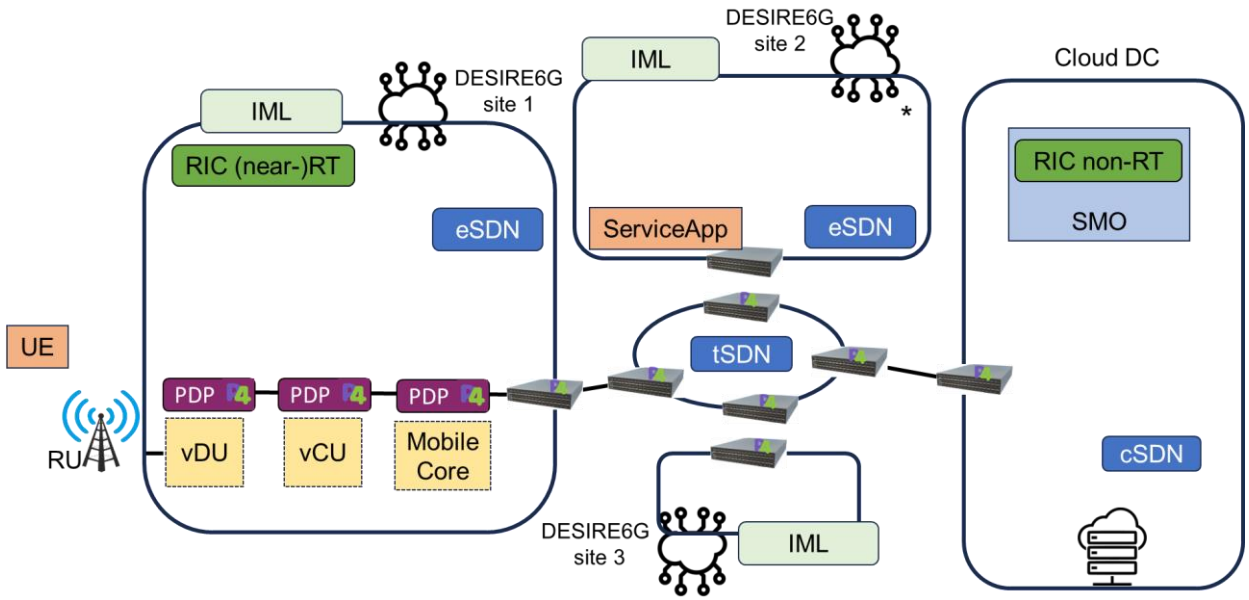

FIGURE 17.. REFERENCE SCENARIO OF APPLICATION FOR THE PERVASIVE MONITORING SYSTEM.

Cloud DC

The overall life-cycle management of the active network services is controlled by the DESIRE6G Service and Management Orchestrator (SMO) running in the central cloud data center (DC) of the MVNOs. The SMO is responsible for the end-to-end deployment and life-cycle management of a network slice, where DESIRE6G network services run. It interacts with the local components (i.e., the IML) at the DESIRE6G sites for the deployment of services, pods and applications as well as handling the SDN transport network.

In DESIRE6G, one of the crucial elements is the Radio Intelligent Controller (RIC). The RIC is typically deployed in the format of service by the SMO and encompasses different component/functionalities to control, manage and monitor the radio infrastructure. In particular, the non-real time RIC is the element responsible for the general control of RAN elements and their resources through specialized applications called rApps. This operation has less stringent time requirements. It communicates with remote sites to deploy and manage/monitor the life-cycle of the radio instances, enabling the deployment of the radio slices and exposing specific APIs. At the edge, the controller operates in (near) real-time, providing immediate or near-instantaneous control over the radio interface by relying on the (near) real time RIC.

The full details on the SMO architecture and its functional modules are reported in deliverable D3.1 "Intelligent and secure management, orchestration, and control platform" [26].

Transport Network

An instance of an SDN controller (cSDN) is responsible for intra-DC networking. It also acts as parent SDN controller for the end-to-end transport configuration, leveraging the hierarchical paradigm, with child SDN controller instances allocated at the edge nodes (intra edge networking) and the transport network. The cloud DC connects to the remote edge nodes by means of a transport network, representing a metro scenario. All connections are established using P4-programmable and legacy switches deployed across the metro network. The topology in the Figure 17 is a simplified example, reflecting the concept for monitoring purposes. The transport network is controlled by a transport SDN controller (tSDN) allowing flexibility and high data-rate for the communication among the different sites.

DESIRE6G sites

Three DESIRE6G sites are shown in the figure. Each one is controlled by a dedicated IML acting as a wrapper of the local VIM (i.e., K8S), extending its functionality and abstracting the available heterogeneous hardware resources. Essentially the IML is responsible for (local) deployment of the network service employing infrastructure cloud entities (i.e., K8s pods), exploiting available accelerators, and the network configuration.

DESIRE6G node 1 provides radio connectivity. More specifically, a functional split scenario is considered, where the Radio Unit (RU) is connected to the virtualized Distributed Unit (vDU). The vDU is then connected to the virtualized Central Unit (vCU). The considered scenario considers a latency-sensitive application that requires the deployment of the mobile core function at the same edge node (Mobile Core), implementing the complete RAN stack. For other network slices with no latency constraints, the core function can be potentially deployed in other DESIRE6G sites across the edge-cloud continuum or even at the cloud DC.

The adoption of a programmable data plane opens the way to leverage hardware acceleration for some of the network function components (i.e., the User Plane Function – UPF or the CU-UP). An instance of near real-time RIC is available at the edge enabling the support for fast closed loop operations.

5.2 Pervasive monitoring architecture

This section summarizes the key concepts and aspects related to pervasive telemetry system. Following the reference discussed in the previous section, an overview of the crucial components and relationship among them is presented in Figure 18.

At the each DESIRE6G site, monitoring can be performed at the level of the service (represented in red) or the level of the infrastructure (represented in yellow). In the example, three types of service metrics can be collected: (i) metrics form the radio segment (i.e., the statistics of the UE), by exploiting a dedicated xApp, (ii) the metrics collected from the P4-programmable components, collecting telemetry information (per-packet metrics during packet transmission) from the data plane and (iii) the metrics related to cloud elements of the service (i.e., the ServiceApp at DESIRE6G site 2).

In addition, relevant metrics can be collected from the infrastructure entities, in example the network devices (nodes and link) and the cloud resources. All those metrics are collected by specific frameworks/probes and are made available for the data consumers (i.e., AI/ML-based routines).

To support these high-performance data pipelines, we employ Apache Kafka [27] as the underlying message broker. Apache Kafka is an open-source distributed event streaming platform originally developed by LinkedIn and later donated to the Apache Software Foundation. It is designed for handling real-time data feeds and stream processing. All data is labelled, allowing data filtering and data aggregation. Apache Kafka is a robust and highly scalable platform that plays a critical role in modern data architectures, enabling real-time data streaming, event processing, and data integration for a wide range of applications, from log aggregation to real-time analytics and beyond.

Therefore, in DESIRE6G both service and infrastructure metrics are pushed to the local Kafka-based cluster. In fact, each DESIRE6G site is equipped with a Kafka cluster, being able to be act as metrics re-distribution system towards the entities responsible for intelligent decision-making pertaining to service assurance.

Considering the service metrics related to the considered example, at DESIRE6G site 1, the system collects service metrics form the radio segment and P4-enabled resources. While, at DESIRE6G site 2, the P4 based metrics and the cloud metrics, related to the ServiceApp are collected,

DESIRE6G site 3 is not involved in the service and so no service-related metrics are active.

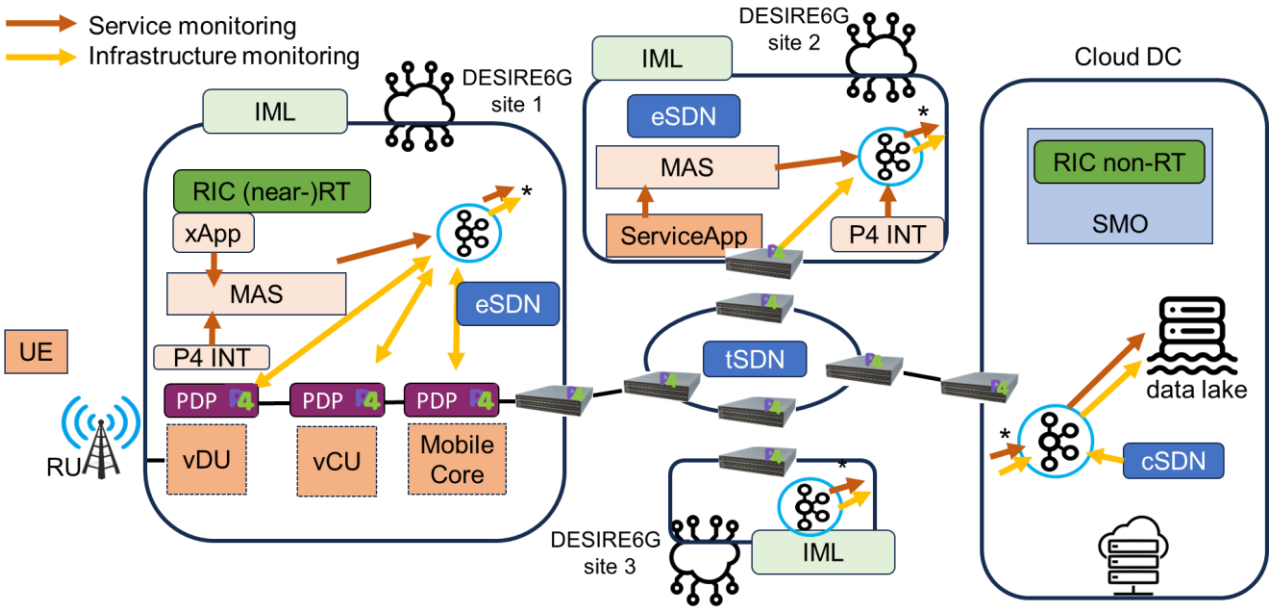

FIGURE 18. SERVICE AND INFRASTRUCTURE PERVASIVE MONITORING.

It is important to note that, all the monitored metrics, including service and infrastructure data, are also streamed to the central Kafka cluster, creating a single huge storage instance (i.e., the data lake) enabling the end-to-end data processing. For this reason, also in the central cloud DC, an instance of Kafka cluster is running, to perform the data injection in the data lake from the streaming of monitoring data from the remote sites. The data lake and the distribution system will be designed, guaranteeing privacy preservation.

5.3 Service Monitoring

5.3.1 Telemetry approaches

DESIRE6G monitoring system leverages on the pervasive monitoring concept. With pervasive monitoring, the elements of the system can report telemetry metrics with a per-packet rate. Pervasive monitoring is an adaptation of the INT approach, where an ad-hoc defined metadata header, appended to the data plane traffic, is adopted to convey crucial service metrics to a collection point.

By extending the original header, defined to transmit timing information, additional metadata (e.g., per hop latency, etc.) can be conveyed during the transmission of each packet, encompassing geo-localization information, radio and application metrics.

The pervasive monitoring data are collected by the *Telemetry Collector*, a novel module, that will be developed within DESIRE6G project. The module is responsible for the collection, the aggregation, the filtering, and the correlation of the data. The module is then integrated with an ad-hoc library, designed to parse and format data form different possible data-sources and data formats to be fed to the Multi-Agent System, where the near real time closed loop operations are executed.

The transmission of the telemetry data can be executed out-of-band, without following the path used to transmit the packet (Postcard telemetry) or in-band. Depending on the INT mode of operation [28], telemetry reports are either directly exported by each INT node (Postcard mode), from their data plane to the Telemetry Collector, or they are embedded into the packets along the data path along with the INT instructions (INT- MD mode). For the latter, deterministic and probabilistic per flow aggregation techniques can be employed to reduce transmission overhead [29].

The two scenarios are shown in Figure 19 and Figure 20, considering the transmission of data from a radio segment toward an edge node, adopting programmable data plane devices. In the first case [30], each traversed switch is responsible for the generation of a *Report*, conveying to the Telemetry Collector all the described metadata. In the second case, each intermediate node is responsible for the addition of a new header to the transiting packet [28]. Only the last switch is responsible for the generation of the Report towards the Telemetry Collector. The Telemetry Collector, alternative to existing approaches that usually employ a telemetry server, can be a dedicated switch of the network, devoted to processing of the data and the transmission of the parsed metrics to the MAS Telemetry, or it can be a normal network switch, encompassing processing and forwarding capabilities. Using data-plane collectors that are capable of collecting and processing telemetry values entirely on the data plane, along with the DESIRE6G MAS approach, can enable shorter control loops towards service assurance.

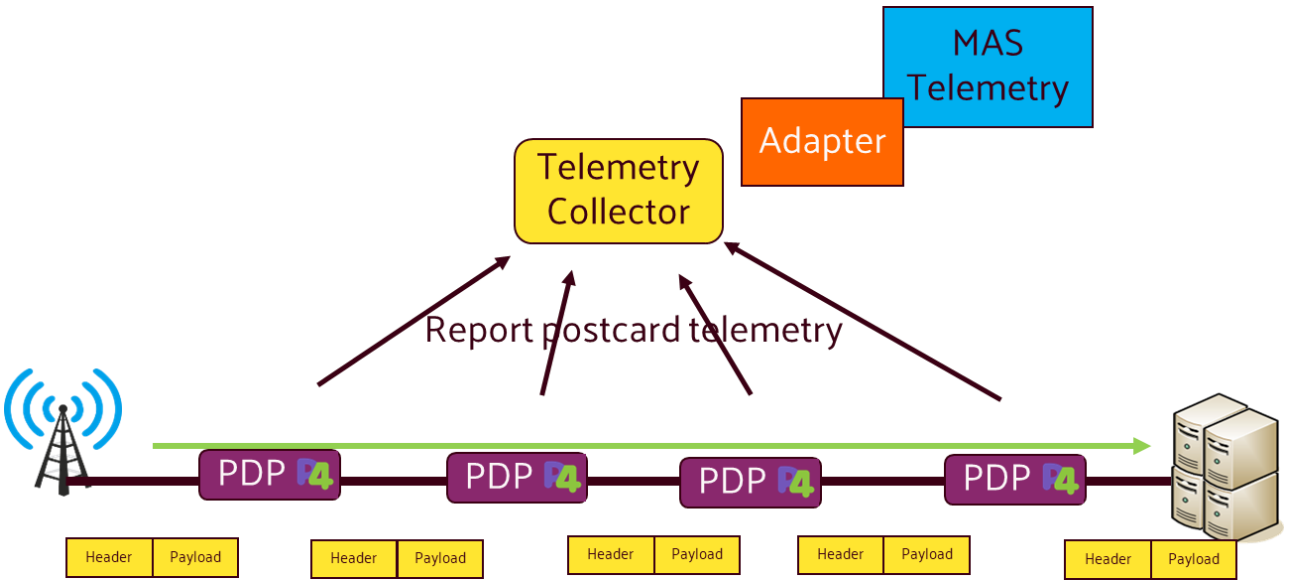

FIGURE 19. POSTCARD TELEMETRY SCENARIO.

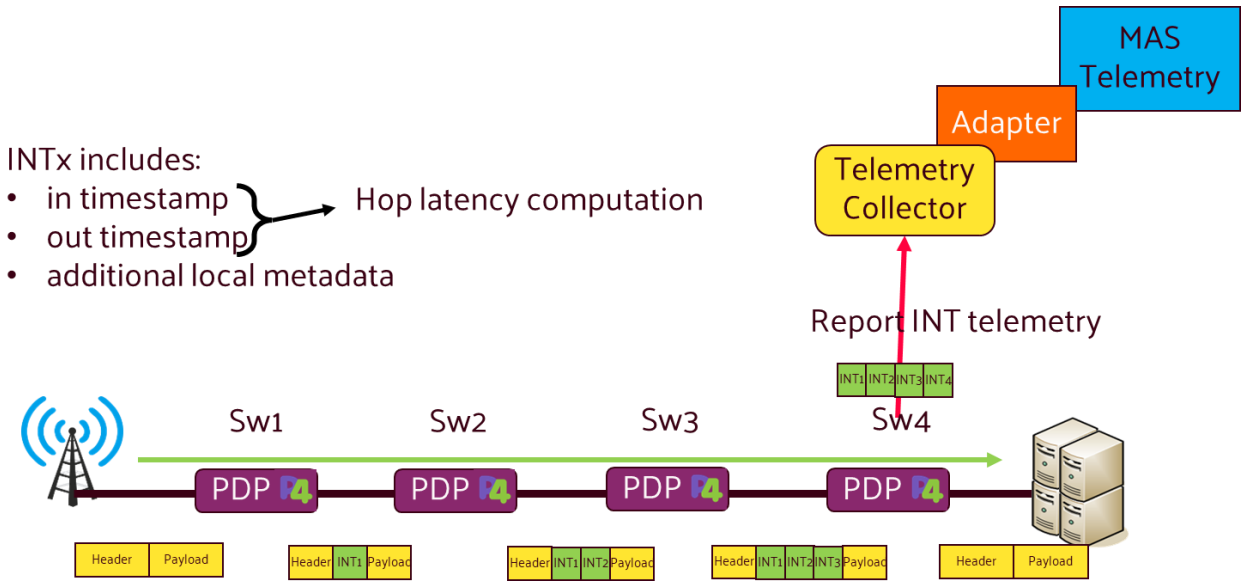

FIGURE 20. INT SCENARIO.

Postcard Telemetry in DESIRE6G

Considering the scenario of Figure 19 (Postcard telemetry), each report message includes:

- ingress timestamp
- egress timestamp
- additional local metadata

By combining the information of ingress and egress timestamps the effective time experienced by the packet to traverse the switch can be evaluated (i.e., hop latency). Additional metrics can be then included, referring to NF, application, GPS location and other (the list is not exhaustive).

In-Band Telemetry in DESIRE6G

Considering the scenario of Figure 20, where INT is considered, the same information is transferred to the Collector combining headers, appended to the packet, and a single Report towards the collector. This Report includes all the data provided by all the traversed switches, making the Telemetry Collector aware of the condition of all the devices.

Unfortunately, INT allows the collection of fine-grained network information in real-time, increasing network visibility, at the cost of network overhead e.g., increased bandwidth consumption or decreased packet processing speed due to the additional telemetry related information. Several lightweight INT approaches have been recently proposed that attempt to alleviate the transmission overhead of INT, while maintaining a high degree of monitoring accuracy, by exercising *per-flow aggregation (PFA)*, i.e., spreading telemetry values across the packets of a flow. The trade-off between monitoring accuracy and INT overhead has been investigated in literature with most of the proposed methods categorized as either deterministic or probabilistic. Deterministic approaches adopt a sampling rate for inserting INT fields in packets of a flow, as means to reduce overhead [31]-[34] where the rate can be adjusted at runtime either by the control plane (e.g.,[34]) or the data plane (e.g., [33]). Alternative propositions employ network topology partitioning [35] or clustering [36] to ensure full coverage for the network, scalability, and freshness/timeliness of telemetry information. Probabilistic approaches employ different probabilistic logic for inserting telemetry information into the data packets [37]-[39] that avoid the hassle of coordination among between the nodes or via the control plane.

We investigate the use of two lightweight INT approaches, namely DLINT and PLINT [29], in the context of DESIRE6G monitoring scenarios. The DLINT approach employs telemetry states, maintained in the programmable network nodes, to support inter-node coordination for the spreading of telemetry data across the packets of a flow to further reduce the network overhead. PLINT employs a probabilistic approach, obviating the need for any coordination among INT nodes.; each INT node can independently insert telemetry data to each packet with a certain probability. PLINT uses reservoir sampling, to

guarantee an equal probability for the encapsulation of telemetry indicators among all INT nodes across a data path.

5.3.2 INT DESIRE6G Case Studies

E2E Delay using Deep Clock Synchronization: The INT scenario can be further enhanced in case deep clock synchronization is exploited in the network switches. In fact, as shown in Figure 21 in the case of tight clock synchronization, the timestamps of the switches will be in synch, allowing the evaluation not only of the hop latency, but also of the link latency and the overall end-to-end latency, by processing the timestamps provided by the different switches.

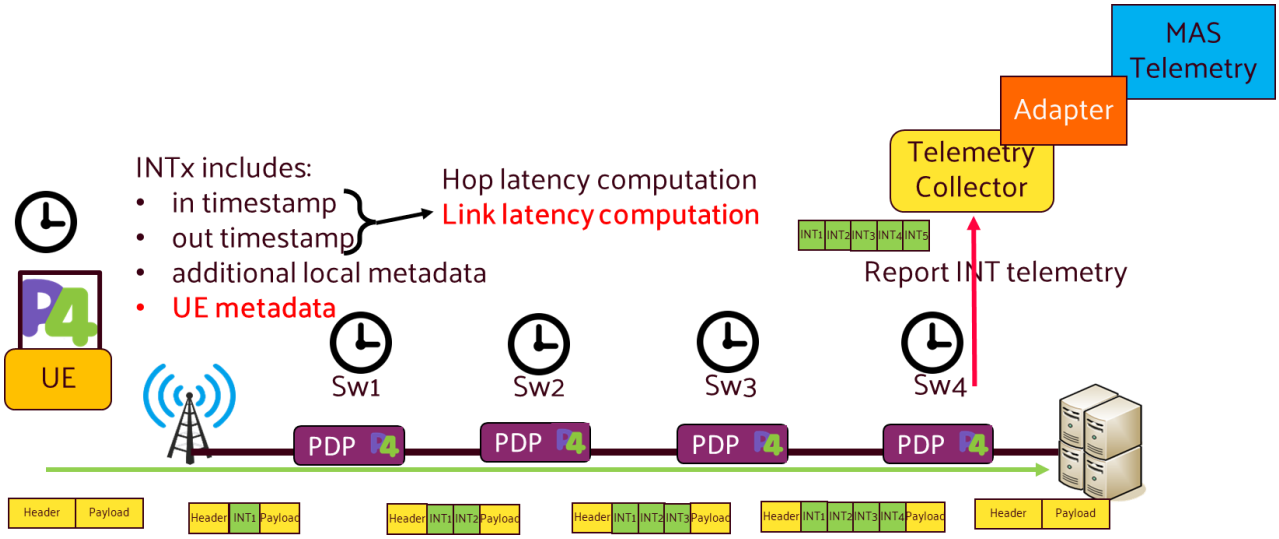

FIGURE 21. INT AND CLOCK SYNCHRONIZATION SCENARIO.

E2E Delay without Deep Clock Synchronization: As demonstrated in [40] , the per link and the end-to-end latency evaluation can be performed even in the case of lack of deep clock synchronization. Analysing Figure 22, a latency aware solution has been implemented, by combining INT headers and Reports. In this case, sw1 acts both as ingress switch for the traffic and sink node for the INT. On the other hand, the egress switch, apart from sending out each data packet towards the destination, also generates a Report message towards Sw1. The message is also enriched with INT headers, inserted during the transmission over the backward direction. Sending to the Telemetry Collector all the

information, including the forward INT and the backward INT values, it is possible to reconstruct the latency of each traversed link. This solution assumes that each link is symmetric in latency (i.e., in a bidirectional link A-B, the time required for the transmission from A to B is equal to the time required for the transmission from B to A).

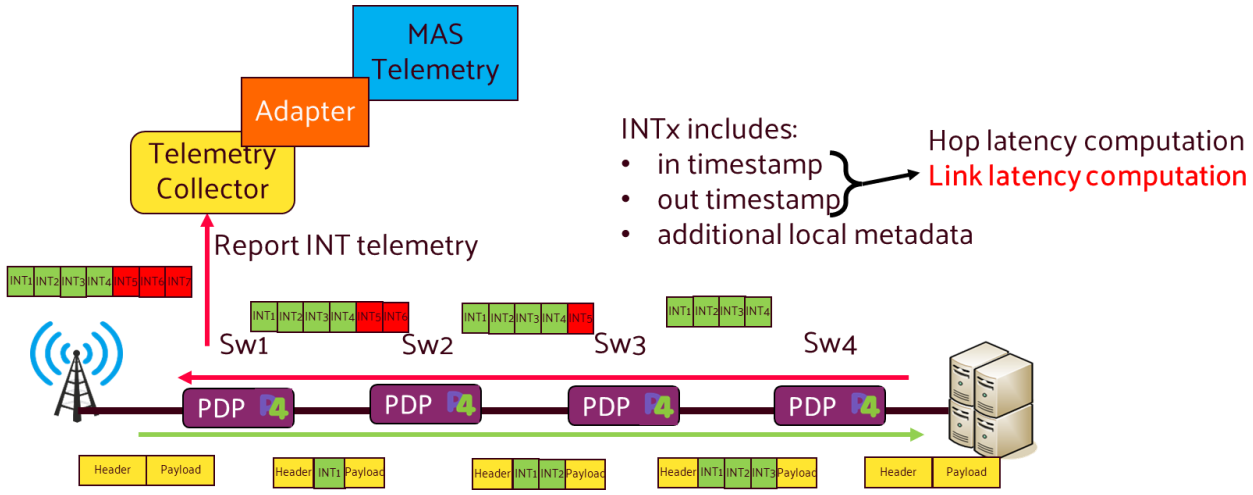

FIGURE 22. BIDIRECTIONAL INT FOR LINK LATENCY EVALUATION IN CASE OF NO CLOCK SYNCHRONIZATION.

E2E delay including the UE: In Figure 23, an evolution of the system is shown, including in the telemetry INT headers also the UE metadata. This can be achieved by employing data plane programmability at the UE side to produce UE metrics. In this case the INT header can include:

- ingress timestamp
- egress timestamp
- additional local metadata
- UE signal status/quality
- Application specific metrics

These additional metrics can be then adopted to perform traffic steering decisions and/or reconfigure the other segments of the network.

As shown in the previous scenarios (Figure 19 and Figure 20), the same concepts can be applied here. In Figure 23 the clock synchronization case is shown to demonstrate a possible application to evaluate the

latency including the radio link between the UE and the base station. The availability of the experienced latency in the radio part can be very useful for service adaptation, offering the possibility to apply different schemes to localize and compensate the source of end-to-end latency variation.

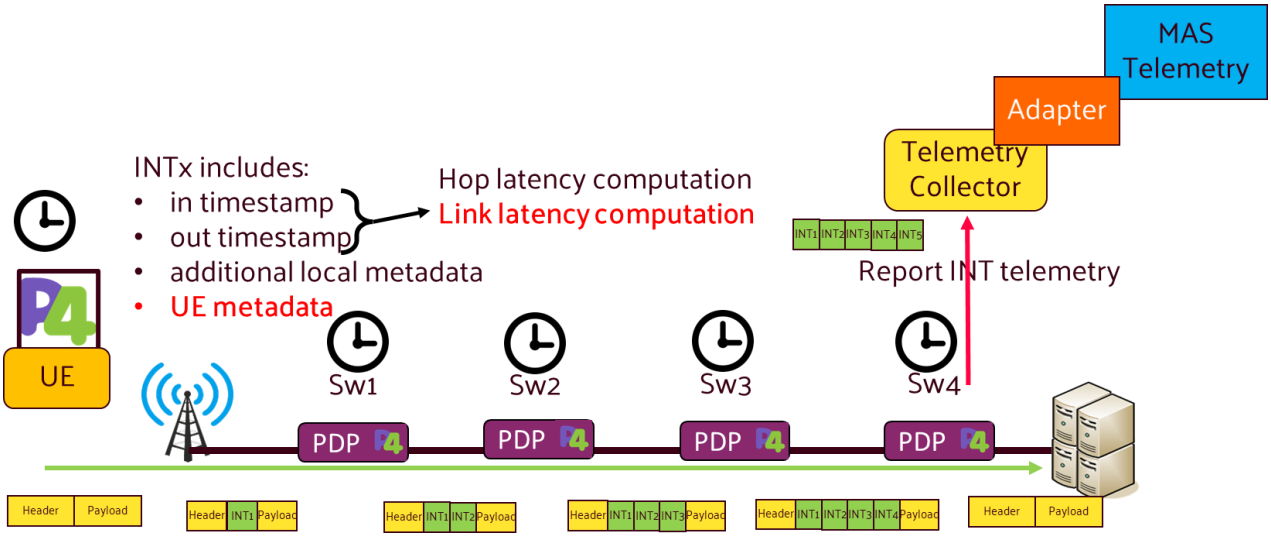

FIGURE 23. P4-ENABLED UE INT SCENARIO WITH CLOCK SYNCHRONIZATION.

In Figure 24 is reported an example of pervasive monitoring applied to a multi-technology scenario. In particular, the case with a programmable data plane and a radio segment is shown. In the programmable segment, the data collection is performed as described in the previous examples, with the adoption of a Telemetry Collector to feed the data to the MAS . In the radio segment, the data retrieval is performed via xApp, feeding the data to the MAS. In fact, by leveraging the near real-time Radio Intelligent Controller (NR-RIC), a specific xApp can be adopted to collect the telemetry data of interest from the slices (per-UEs or aggregated per-slice). At the time of writing, the analysis on the available parameters is ongoing. Some metrics have been identified (i.e., the wideband Channel Quality Indicator) from an experimental testbed, but additional metrics will be considered according to the adopted implementation of a NR-RIC.

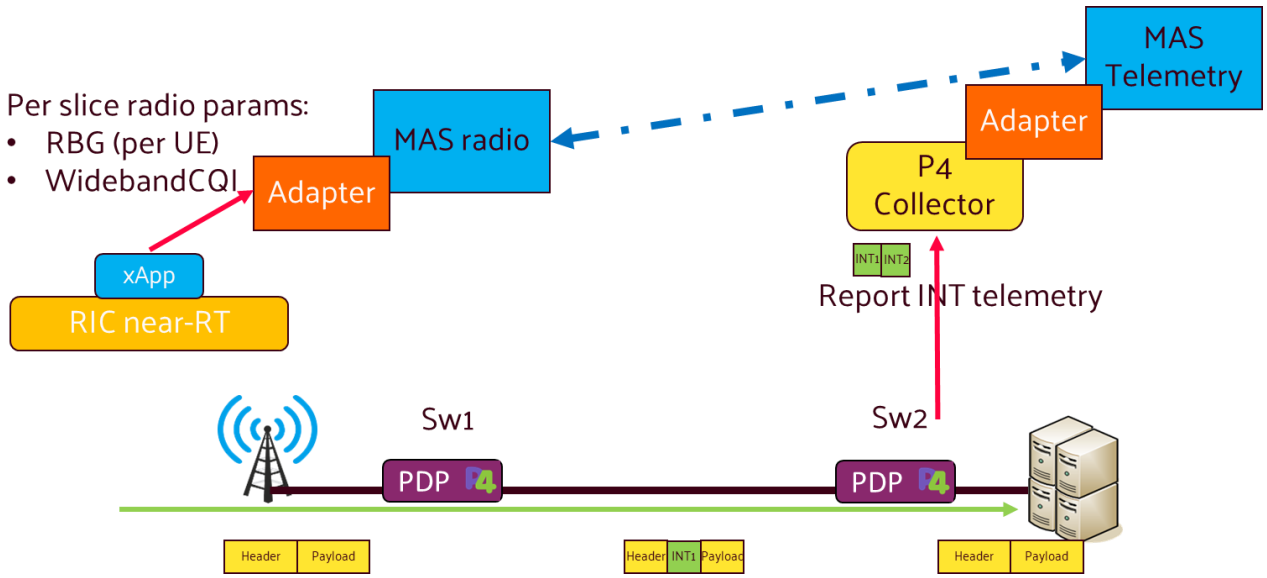

FIGURE 24. EXAMPLE OF MULTITECHNOLOGY END-TO-END SCENARIO WITH RADIO AND TRANSPORT DOMAIN.

The collected radio metrics will be then consumed locally, at the radio MAS agent, for the near-real time control of the service or to trigger its reconfiguration.

Path Tracing: Another metric, particularly significant for supporting mobility and service assurance in the envisioned DESIRE6G architecture is path tracing. This metric helps to track the route that data packets take in the network. Traditional methods of tracking network traffic such as NetFlow are done in the control plane (software) of forwarding devices, are usually resource intensive and provide incomplete information due to time granularity of tracking flow paths [41].

Figure 25 presents the path-tracing scenario, including the telemetry INT headers. In the example, the INT header can include the *switch_id* or the *switch_id, init_TTL* and *hop_number* for the case of DLINT and PLINT respectively. As described in [29] this is the information needed to reconstruct the path on for each approach. Path updates (e.g., due to handover or service degradation) at the 1st level telemetry collector (e.g., P4 collector) can be used by the DESIRE6G multi-agent system in place to update the corresponding forwarding rules at the respective routing NFs (see section 3.1.4). Note that in the DESIRE6G architecture, the infrastructure components fully or partially take over the roles of some 5G functions like UPF. For example, user mapping to network services and the routing to the appropriate cell is solved by the infrastructure. According to this approach, path tracing can ensure the signalling mechanism in DESIRE6G needed for, e.g., mobility support.

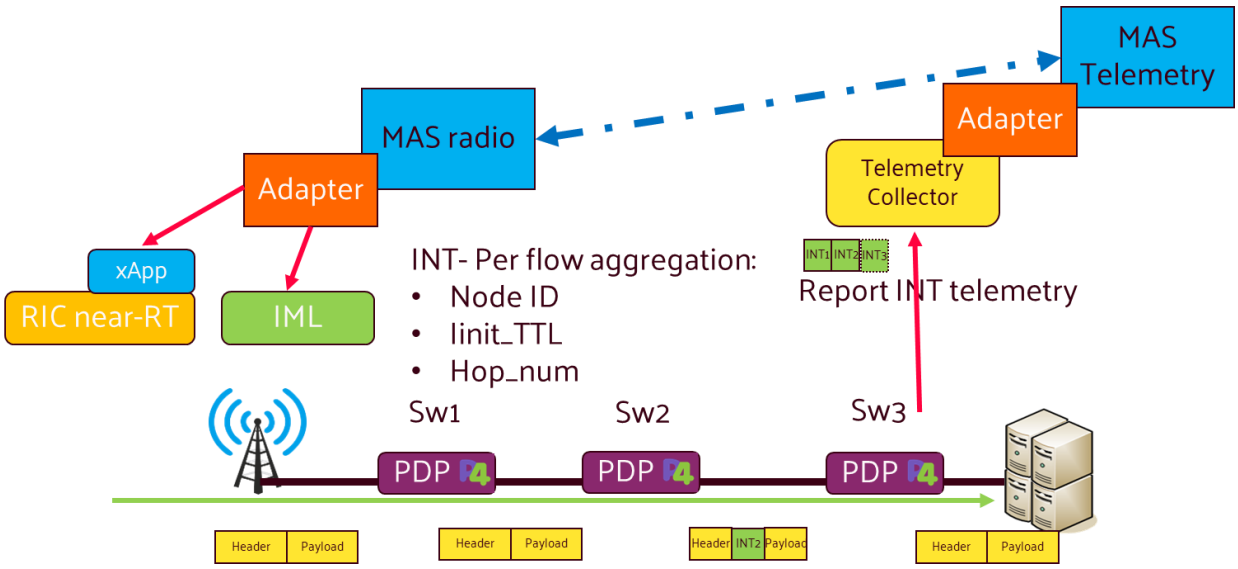

FIGURE 25. P4-ENABLED PATH TRACING SCENARIO

Figure 26 and Figure 27 present the accuracy (correctness) and responsiveness in detecting path updates presented in [29], via a comparison of DLINT and PLINT with a prominent probabilistic INT technique, i.e., PINT packets [37] . Assuming that all flows trigger path update during their lifetime, f indicates the percentage of these updates that have been successfully detected; for DLINT the BF size (memory) is critical for the path update detection, hence different ratios in the number of flows to the BF size are presented. As noted in both figures, more telemetry values than just one can be recorded by the intermediate nodes, affecting the accuracy (correctness) and responsiveness in detecting path updates. PLINT detects the path update much faster (up to 40%) than PINT. In fact, is the most accurate and fast approach among the three, while DLINT is more accurate and equally fast in the case that more than one intermediate node reports their id in a single packet, at the cost of additional transmission overhead.

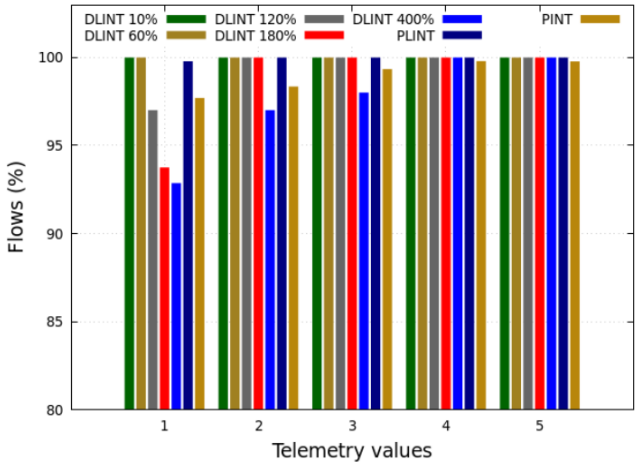

FIGURE 26. PERCENTAGE OF FLOWS WHERE THE PATH UPDATE IS DETECTED.

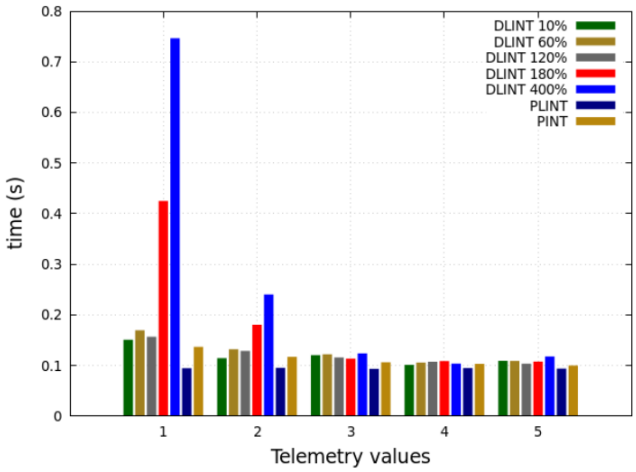

FIGURE 27. TIME FOR PATH UPDATE DETECTION

As with most PFA-INT approaches in the state of the art, the respective experiments for DLINT and PLINT were conducted in a virtual environment (Mininet) using software switches (BMv2). Currently, the solution is validated using the P4-Programmable Tofino ASIC, to identify the challenges of porting the solution to the selected hardware target and investigate the impact of the (PFA-)INT approaches on the utilization of the hardware resources, processing delay and energy consumption. The telemetry data in this experiment validation are embedded to the packet as TCP Options, as depicted in Figure 28. Another option we have investigated in the past is the IPv6 Hop-by-Hop Options header [41].

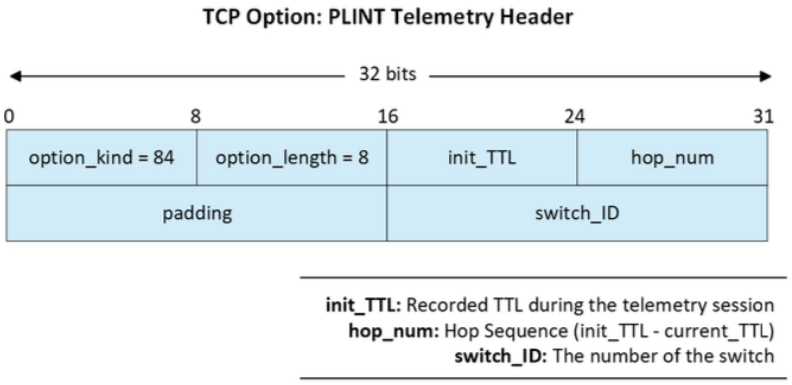

FIGURE 28. STRUCTURE OF A SAMPLE TELEMETRY HEADER FOR PLINT

Indicatively, Table 1 presents the memory utilization (TCAM and SRAM) and number of stages for DLINT and PLINT compared against basic L3 forwarding. The results are obtained using the P4 Insight (p4i) tool1 provided by Intel to inspect P4 code. We note that there is no significant impact on the use of TCAM/SRAM between the two methods and L3 forwarding; the small increase in the use of SRAM is due to the use of by registers (DLINT employs Bloom Filter for keeping telemetry state).

TABLE 1. RESOURCE UTILIZATION

Resource	DLINT	PLINT	L3 FWD
SSTAGES	7	7	2
SRAM	0.5%	0.3%	0.1%
TCAM	0.4%	0.4%	0.4%

5.3.3 Telemetry Collector

The key component, responsible for the connection among PDP and the MAS Telemetry, is the Telemetry Collector. This section describes the general structure and the main set of functionalities of the P4 Collector.

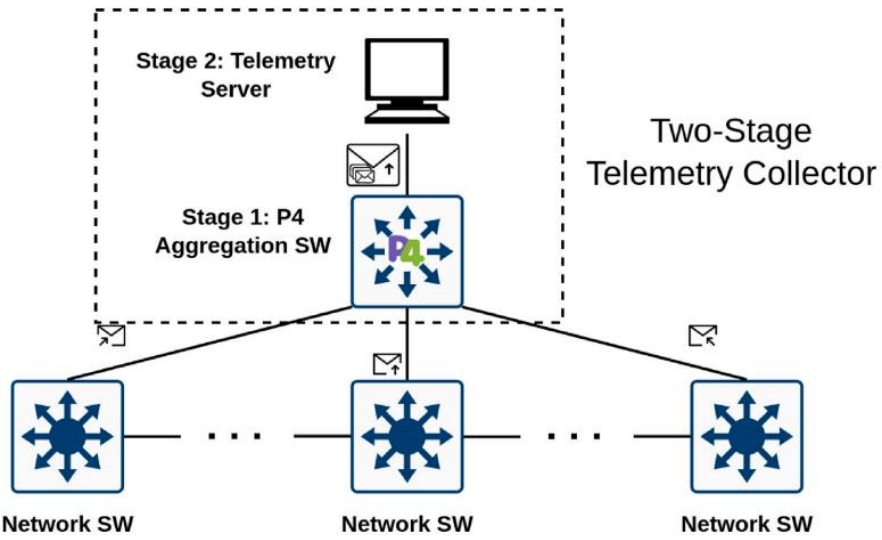

FIGURE 29. TWO STAGE TELEMETRY COLLECTOR USED FOR POSTCARD TELEMETRY IMPLEMENTATION

Figure 29 shows the considered PDP reference network scenario. A Network Switch generates a postcard which is a packet dedicated for carrying the switch's telemetry data (e.g., for every data packet or only for data packets of certain tracked flows). The generated postcard contains the switch metadata related to the packet (e.g., timestamps, queue length, interfaces, ...). The Aggregation Switch receives each postcard then extracts and stores the metadata in a buffer in memory, after this the postcard is dropped as it is no longer needed. When the buffer at the aggregation switch becomes full the aggregation switch builds a new packet that contains the aggregated metadata and forwards it to the Telemetry Server.

In Figure 30, we propose different levels of aggregation that the P4 aggregation switch can perform. The level of aggregation refers to how many telemetry reports are aggregated in one packet and whether further processing is applied to these reports. The proposed aggregation levels are as follows:

Agg(R): This is the simplest level of aggregation, requiring only one generic buffer. In this solution, the P4 aggregation switch extracts and aggregates R telemetry reports in one larger packet and forwards it to the telemetry server, regardless of the flow that triggered the generation of the telemetry report or the switch that originated the report.

$Agg_N(R)$: This solution is slightly more complicated, as it extracts and aggregates telemetry reports based on the network node where they originated. To implement this solution, the aggregation switch would require N buffers, with each buffer storing the telemetry reports from a distinct node.

$Agg_{N,F}(R)$: This is the most complex solution, in which the P4 switch has N buffer banks, with N corresponding to the number of network nodes generating telemetry reports. Each buffer bank consists of F buffers, where F is the number of flows passing through the network node. This solution can significantly reduce subsequent correlations that need to be performed by the telemetry server.

The P4 aggregation switch can perform additional functionality in certain scenarios to reduce the amount of network information collected and processed. This includes providing only peak, minimum, and average values instead of comprehensive data. This helps to decrease bandwidth usage and CPU cycles needed for data processing. Based on the three aggregation levels described earlier, three new solutions can be derived, performing correlations (as shown in Figure 31). These solutions involve performing correlations on telemetry data at the P4 aggregation switch to calculate maximum, minimum, and average values for network and switch metadata parameters such as queue lengths, latency, and traffic.

The proposed correlation levels are proposed as follows:

$Cor(R)$: This solution is based on $Agg(R)$ previously mentioned. The P4 switch performs correlations on telemetry reports from different switches and flows. The information generated by this solution provides a broader view of the entire network or at least the aggregation segment.

$Cor_N\ (R)$: This solution is based on $Agg_N\ (R)$ mentioned earlier. The P4 aggregation switch performs correlations on telemetry reports organized by the originating network node. This solution offers better granularity as the correlation process provides per-switch details.

$Cor_{N,F}\ (R)$: This solution is based on $Agg_{N,F}\ (R)$ and performs correlations on reports organized per-flow and per switch. This solution offers the highest granularity and significantly reduces the workload for the telemetry server.

We can illustrate the difference between different aggregation levels in terms of the number of packets and bandwidth usage in Figure 32. With no aggregation, R telemetry reports are sent in R packets. When performing aggregation, it is possible to remove $R - 1$ set of headers. With Cor, further reduction of the stack is performed whereas, besides headers, one single correlated report is enclosed with statistics extracted from R reports, with a significant gain of bandwidth and CPU processing at the telemetry server.

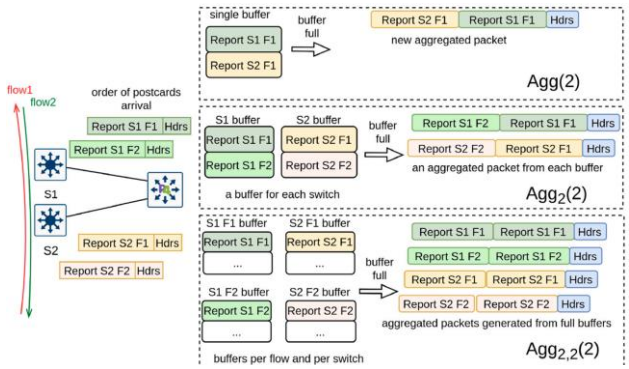

FIGURE 30. P4 COLLECTOR: AGGREGATION MODE

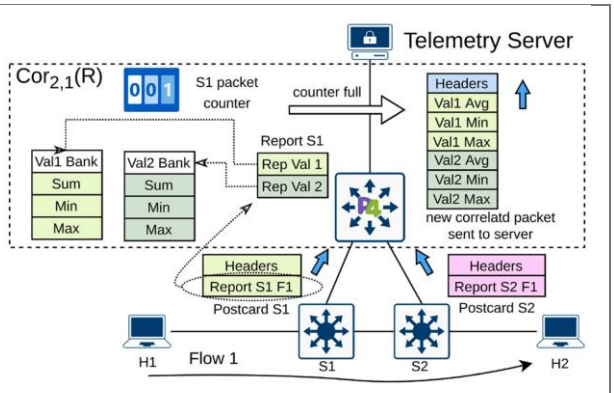

FIGURE 31. P4 COLLECTOR: CORRELATION MODE

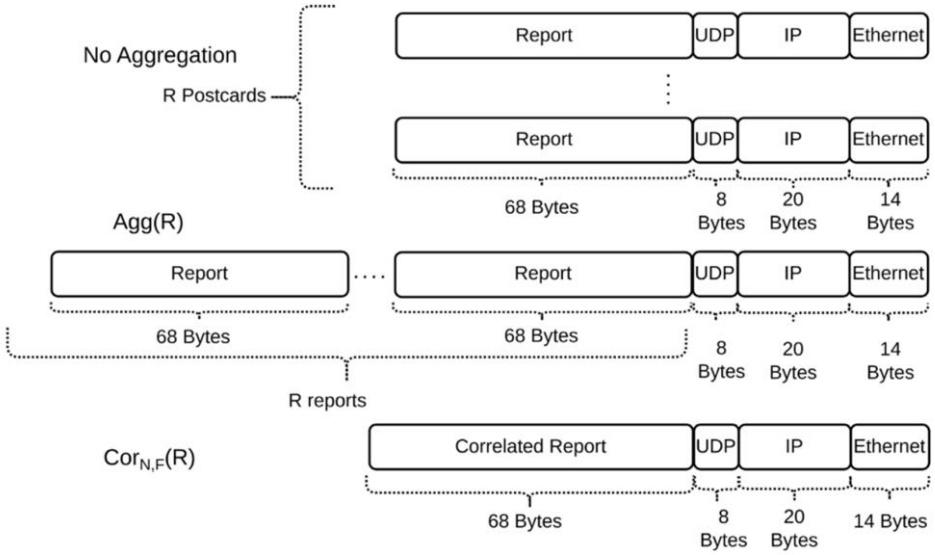

FIGURE 32. DIFFERENT REPORT STACK IN NO AGGREGATION, AGG AND COR MODES.

5.4 Telemetry and Distributed Intelligence

To achieve scalable near real-time operation, the DESIRE6G solution for the next generation of mobile networks integrates: i) pervasive monitoring based on INT; ii) intelligent data aggregation at both data and control plane to deal with scalability issues; and iii) intelligent distributed control for near real-time service operation [49].

Telemetry measurements can be aggregated by telemetry processors targeting dimensionality reduction, as well as reducing data rate at the control plane. Examples include compression of individual

measurements and time series aggregation. In both cases, techniques based on statistics, machine learning (ML), e.g., autoencoders (AE), and data stream mining, can be used.

In addition, intelligence entails, among others: i) adaptation, i.e., dynamically deciding when and how data aggregation needs to be done; ii) consolidating/correlating heterogeneous measurements; and iii) adding value to measurements, e.g., AEs can be used in both forward and backward directions to quantify relevance metrics at the latent feature space and input.

Distributed decision making is the project's choice for network and service operation, not only to relieve the SDN controller / orchestrator from fine grain tasks and increase scalability, but also as an enabler for near real-time control. In such approach, agent nodes are augmented with intelligent algorithms, e.g., based on reinforcement learning that make autonomous decisions as a function of the observed conditions, collected through telemetry. In addition, communication among agents with similar or different capabilities allows creating distributed control systems (e.g., MAS) that can cooperate to achieve some common objective, like ensuring E2E delay for services under changing conditions.

Let us present a use case that combines the solutions described in the previous section for the near real-time control of E2E 6G services. Figure 33 presents the scenario, where a service provides connectivity between an unmanned aerial vehicle (UAV) and VNF providing computing and storage resources for real-time augmented/virtual reality (AR/VR) high quality video reconstruction. Strict E2E and segment latency and jitter requirements imposed by the application, require continuous monitoring of a large number of stream flows, to dynamically make routing and edge computing resource selections that satisfy the committed performance.

For that very reason, INT is used to extend packet headers with metadata added by the UAV and the intermediate network nodes: i) the UAV adds geo-location coordinates; ii) the RAN aggregates signal quality indicators, e.g., the wireless received signal strength indicator (RSSI) for WiFi or the channel quality indicator (CQI) for 4G/5G, and the latency experienced in the access segment; and iii) intra/inter-switch latencies and jitter are collected from the packet nodes. To cope with telemetry scalability issues, two-stage P4 collectors aggregate INT reports in different ways, employing both standard report aggregation and report correlation. The former option aggregates relevant metadata in a single report

packet per configurable time window or number of packets, while the latter provides metadata statistics related to different flows, switches and e2e intents (e.g., maximum/average latency values).

Depending on the INT application mode, telemetry reports are either directly exported by each INT node, from their data plane to the collector switch (P4 aggregation in Figure 33), or they are embedded into the packets along the data path. For the latter, we employ deterministic and probabilistic per-flow aggregation techniques (see section 5.3.1) to reduce transmission overhead thus, spreading the telemetry values across multiple packets of a flow. Finally, the collector switch strips the aggregated metadata and selectively sends them to the telemetry processor.

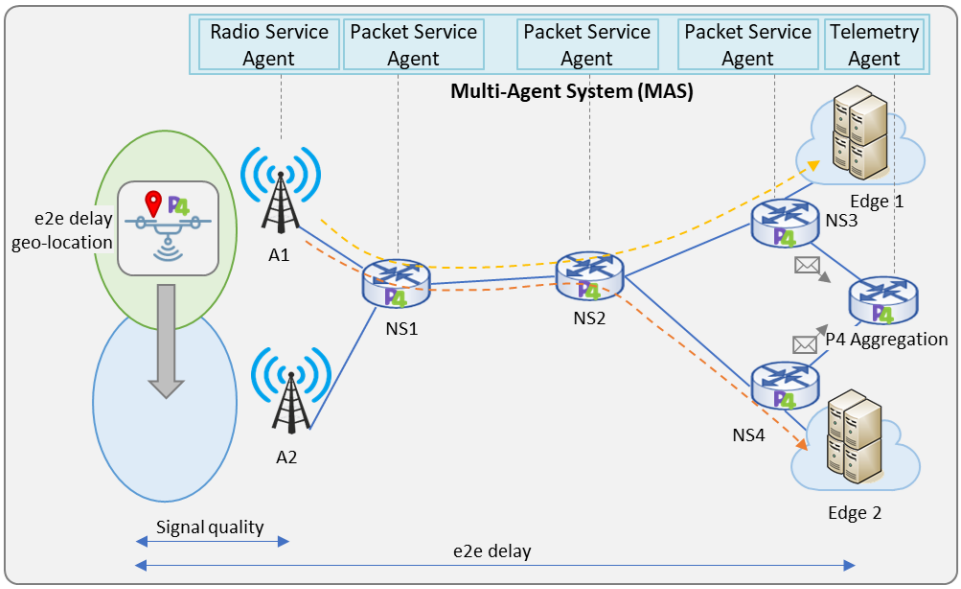

FIGURE 33. ILLUSTRATIVE SCENARIO SUPPORTING AN E2E 6G SERVICE.

The collected data are consumed at the intelligent control plane, where the MAS controls the service near real-time. The MAS consists of a number of heterogeneous service agents running as part of node agents that communicate among them. Service agents might include (Figure 34): i) a telemetry processor to collect from the local node and process telemetry data, including intelligent data aggregation; ii) an inter-agent communication module, which is used for telemetry data distribution and state and model sharing; and iii) technology-specific (i.e., RAN, packet, etc.) intelligence for autonomous decision making based on local and remote observations. The SMO layer provides guidelines to the MAS, while giving freedom to the MAS for operating on the resources allocated to the service.

Figure 34 illustrates this operation; the agents receive from the SMO the resources that can be used for the connectivity service and the max E2E delay ($dmax_{e2e}$) that needs to be ensured. Based on the required performance, let us assume that each segment is assigned with an initial budget delay ($dmax_{Di}$) (labelled 1 or 2 in Figure 34). Once in operation, E2E delay and other performance indicators are measured, and results distributed among the agents in the MAS (2). For instance, in the case of the radio segment, measurements and forecasted metrics are used to enforce the resource block group adaptation with margin in time, reducing the service SLA violations. However, imagine that at some point in time, the radio segment cannot provide the committed delay in its domain. In this case, the RAN agent announces the new delay budget for its segment to the other service agents (3), so packet agents can make decisions, e.g., changing routing, to ensure the new budget delay in their domains.

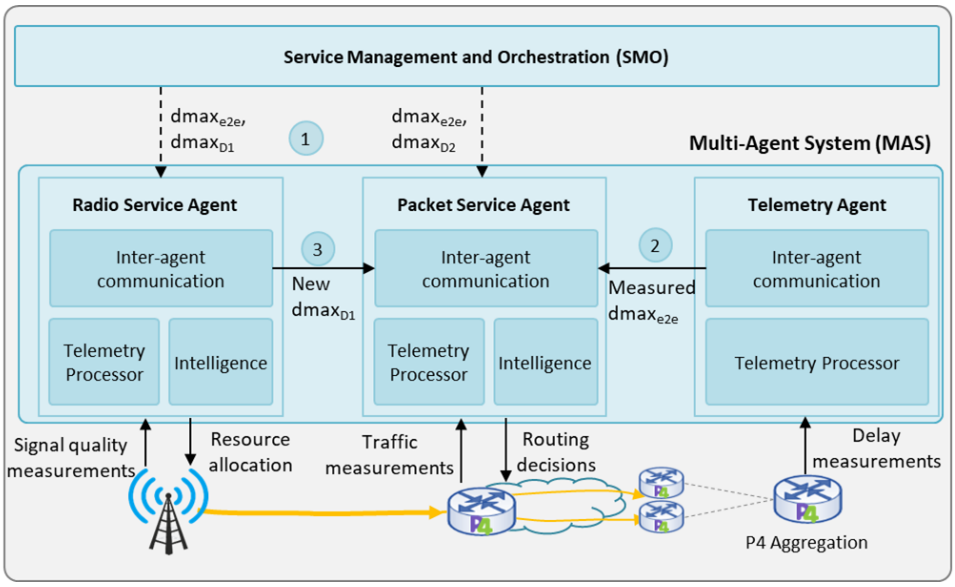

FIGURE 34. PERVASIVE TELEMETRY AND DISTRIBUTED INTELLIGENCE SOLUTION.

5.5 Infrastructure Monitoring

The infrastructure monitoring capability is an essential component that helps organizations to maintain a stable and high-performing network infrastructure, enables proactive issue resolution, enhances security, and supports strategic decision-making through data-driven insights. The choice of specific tools and technologies within the framework may vary depending on the organization's requirements

and infrastructure. Following preliminary considerations on the collection strategies to be applied considering Network, cloud, radio and security aspects is reported.

It is important to mention that the proposed kafka-based monitoring system is very flexible and versatile, offering a wide range of tools and plugins for the data collection, the data adaptation (form different sources) and the data processing. For example, Telegraf is a plugin-driven server agent for collecting and reporting metrics. It performs a metric forwarding/adaptation from an input plugin to an output plugin (datastores, services, and message queues). Telegraf has integrations to a variety of metrics, events, and logs directly from the cloud entities (including the containers), pull metrics from third-party APIs, or even listen for metrics via a StatsD.

5.5.1 Network Monitoring

In this context different metrics and parameters can be observed to maintain a real time network view at the edge (locally at a DESIRE6G site) and in the central cloud. For this perspective, in an SDN network scenario, maintaining up to date the topological view of the network is crucial. This allows to detect link/node failures, triggering the reconfiguration procedures to guarantee the service SLA.

In addition, relevant metrics can be collected, monitoring the device operational parameters including:

- bandwidth utilization,
- latency,
- energy consumption,
- error rates,
- resource usage (CPU, RAM),
- aggregated throughput,
- interfaces status,
- per interface counters.

The knowledge of those metrics can be helpful to avoid bottlenecks in the deployment phase and during the lifecycle of a service.

When and where possible, specific agents will be adopted to collect the infrastructure metrics of interest from the network device. This will allow the continuous collection and transmission of data. The agent

will periodically query the device, streaming the resulting state condition to the Kafka-based monitoring system [27].

In other cases, the monitoring data collection will be performed by already existing monitoring systems (i.e., Nagios [42] , Zabbix [43] , Prometheus [44] ALTO [45] etc.,) where the data is collected and stored. In this conditions, specific plugins will be developed to automate the data collection and the data injection in the DESIRE6G pervasive monitoring system.

In particular, ALTO serves as a network topology provider, as explained in section 2.4 of DESIRE6G D3.1, representing this topology information (e.g., network nodes, links, bandwidth, etc.) in a unified and abstracted way. To do so, ALTO requires to be fed with this information (as it does not monitor network devices), which can be provided by the PDP through various means such as an open configuration script or BGP-LS.

5.5.2 Cloud Monitoring

For the cloud monitoring aspects, some assumption is needed according to the virtualization strategy in place. Considering a Kubernetes environment, a robust ecosystem of tools and practices for monitoring your clusters and applications is already available and integrated.

To monitor the Cluster status, two are the more efficient solutions in literature:

Prometheus [44] Prometheus is a popular open-source monitoring and alerting toolkit. It's highly customizable and can be used to monitor various aspects of your cluster, such as node and pod metrics.

Considering the Prometheus Node Exporter, node-level metrics, (CPU, memory, disk, and network utilization) can be exported and injected in the DESIRE6G pervasive monitoring system.

kube-state-metrics [46] is a tool that exports information about the state of objects in a Kubernetes cluster. It is typically used to monitor the overall cluster status and the application health.

cAdvisor [47]: The Container Advisor provides container-level metrics and is integrated into the kubelet, making it easy to collect metrics on running containers.

5.5.3 RAN Monitoring

The following diagram shows the RAN monitoring aspects and the Telemetry interfaces in DESIRE6G architecture. Figure 35 shows the considered scenario.

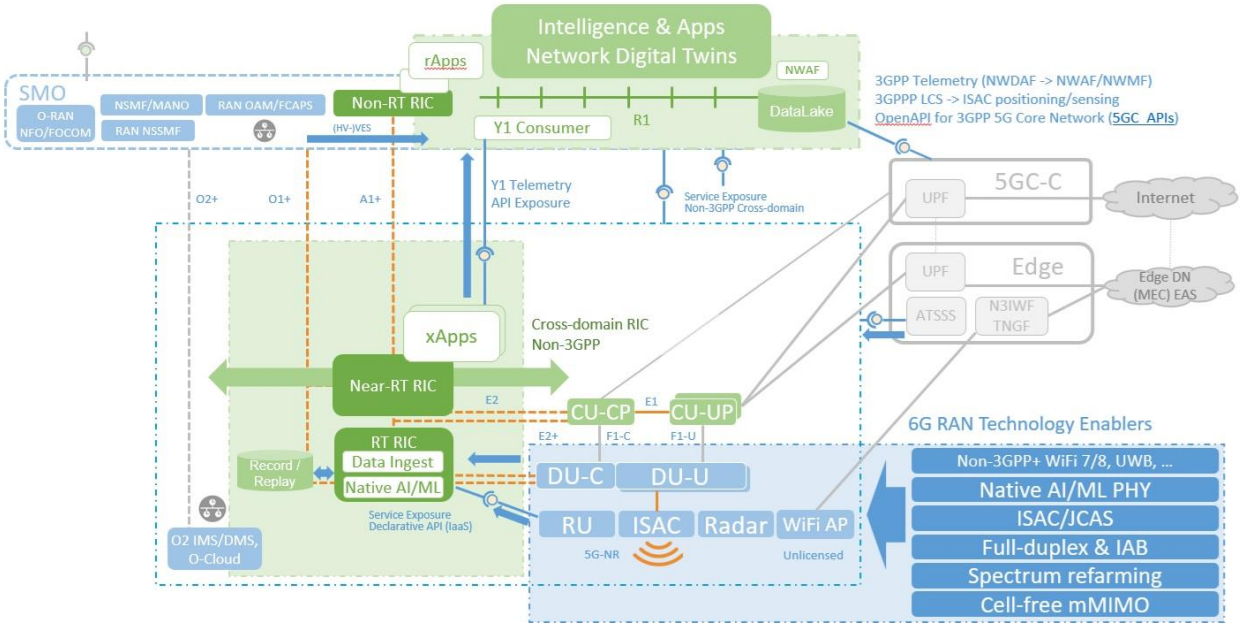

FIGURE 35. DESIRE6G RAN TELEMETRY INTERFACES

The CU-CP and the DU publish 3GPP 28.552 Performance Management (PM) stats on to Kafka data bus and O-RAN E2 interface respectively. The CU also publishes periodic UE air interface measurements. The xAPPs can subscribe to these measurements and implement ML algorithms to optimize the network performance or implement control functions. O1 interface supplies the necessary telemetry to SMO for rAPPs using VES (VNF Event streaming) interface. The VES interface is also used by RAN elements to publish Performance measurement stats and Alarms. These VES events and E2 KPM stats can be monitored by MAS Agents on near-RT RIC to optimize the RAN configuration or perform RAN control commands. The MAS Agents on non-RT RIC can optimise RAN configuration or interact with SDN controller to optimise the transport network resources. The O-RAN Y1 interface will support near-RT RIC telemetry data exposure to non-RT RIC.

Table 2 shows the telemetry metrics that can be made available from the CU-CP and CU-UP to near RT RIC.

TABLE 2. AVAILABLE METRICS AT CU-CP AND CU-UP

CU-CP counters	CU-UP counters
Mean number of RRC Connections	UL PDCP SDU loss rate
Maximum number of RRC Connections	UL F1-U packet loss rate (NOT implemented)
Number of PDU sessions requested to setup	DL PDCP SDU drop rate (NOT completed)
Number of PDU sessions failed to setup	Average delay DL in CU-UP
SS RSRP Distribution per SSB	Distribution of delay DL in CU-UP

Table 3 summarizes the metrics that the DU can report following Telemetry per cell via E2 interface[1]:

TABLE 3. AVAILABLE CELL-BASED METRICS

Cell metrics	Description
RRU.PrbTotDl	3GPP 28.552 section 5.1.1.2.1 DL Total PRB Usage
Maximum number of RRC Connections	3GPP 28.552 section 5.1.1.2.2 UL Total PRB Usage
RRU.PrbTotUl	3GPP 28.552 section 5.1.1.3.1 Average DL UE throughput in gNB
DRB.UEThpDl	3GPP 28.552 section 5.1.1.3.3 Average UL UE throughput in gNB
DRB.UEThpUl	Distribution of delay DL in CU-UP
CARR.WBCQIDist	3GPP 28.552 section 5.1.1.11.1 Wideband CQI distribution
CARR.PDSCHMCSDist	3GPP 28.552 5.1.1.11.1 Wideband CQI distribution
CARR.PUSCHMCSDist	3GPP 28.552 5.1.1.12.2 MCS Distribution in PUSCH
DlUeThroughput-Cell	Downlink UE throughput for whole cell
RACH.PreambleACell	3GPP 28.552 5.1.1.20.1 Received Random Access Preambles per cell
DRB.MeanActiveUe	Number of active UE per cell

[1] This is just for example. Actual telemetry supported will be dependent on the DU finally chosen for the project.

Table 4 reports the per UE[2] metrics that the DU can report.

TABLE 4. AVAILABLE UE BASED METRICS

UE metrics	Description
Effnet.RLC.TxBufferState	RLC tx buffer state of the UE
Effnet.CARR.WBCQIDist.Ue	Wide band CQI distribution for requested UE
Effnet.L1M.ATADist.Ue	Timing advance distribution for requested UE

5.5.4 Software Monitoring

DESIRE6G deep telemetry goes down to the software level with the objective to get confirmation that a software is effectively executing, at the right pace and is integrated during its execution, progressing the state of the art. These novel proofs of software correctness will be remotely monitored and will be derived from automatically appended control flow shadowing (CFS) processing, which deviates the control flow to a security routine before jumping to the next code block. These proofs will be delivered in the form of metadata, constructed in the security routine. Typically, the proof of effective execution can simply derive from the code block jump (which triggers the execution of the security and monitoring routine).

DESIRE6G is working on the elaboration of time series extraction directly operated on binary rewritten protected software, conferring novel forms of correctness validation during the execution. This metric collects call frequency and execution, during each code block of the software control flow graph, with the objectives of regulating runtime integrity verifications and delivering novel performance ratio as well as effective execution evidence. Figure 36 shows the principle of time series extraction during the runtime execution. The relevance of the time series extraction, in terms of quality of the data, impacts on the performance of highly constrained Network Function L2fwd

[2] This is just for example. Actual telemetry supported will be dependent on the DU finally chosen for the project.

According to the achieved results, the retrieval of the locally stored time series could be orchestrated asynchronously and through the DLT and the smart contract as an additional and secondary security service to the MAS inter-agent remote attestation.

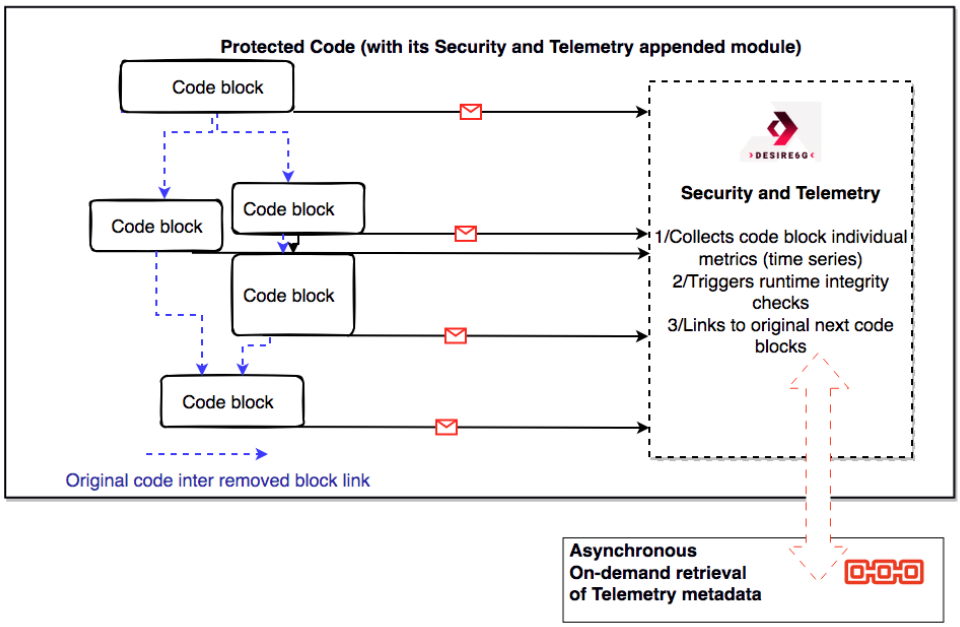

FIGURE 36. GENERAL PRINCIPLE OF THE TIME SERIES EXTRACTION ON THE PROTECTED SOFTWARE CONTROL FLOW GRAPH DURING ITS EXECUTION

Preliminary implementation tests on L2fwd control flow graph hooking by use of E9patch static binary rewriting library is started, using the trampoline insertion at control flow instructions (jump, call, ret). These tests on the impact of the trampoline insertion on L2fwd performance have revealed that performance can be significantly degraded even when the trampoline is empty as shown in Figure 37.

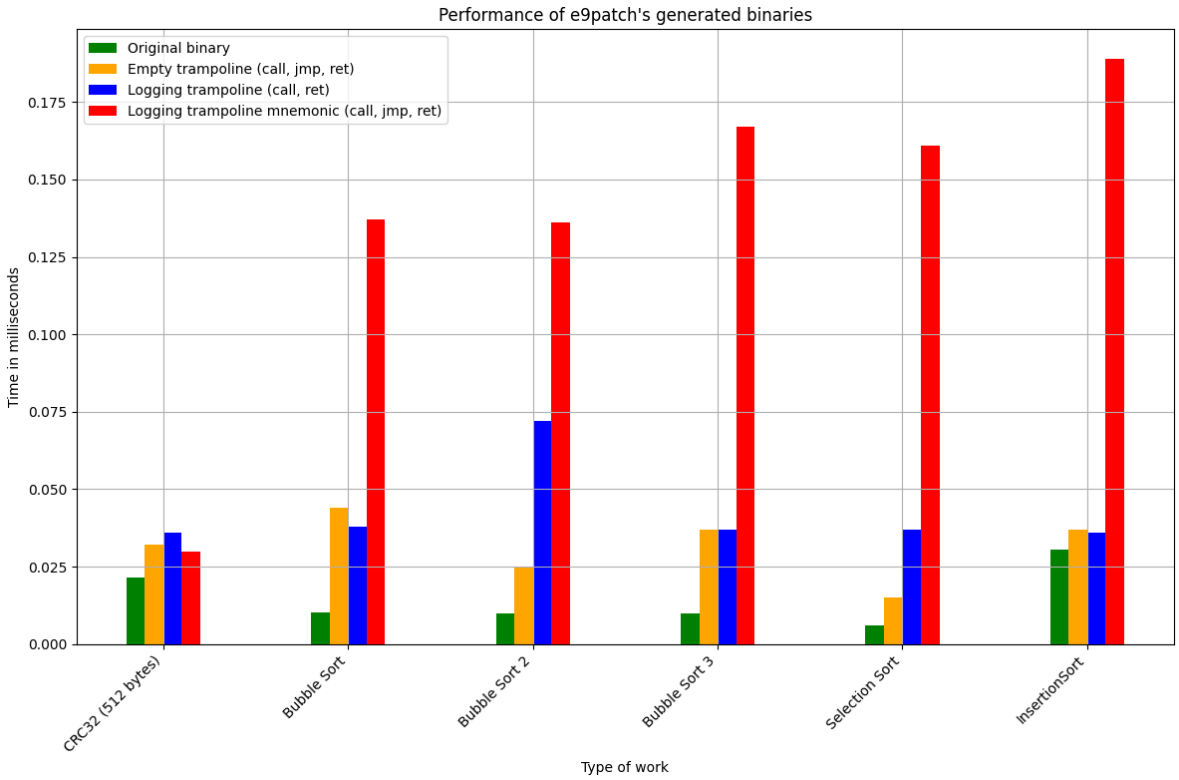

FIGURE 37. TRAMPOLINE INSERTION PERFORMANCE IMPACT ON DEMANDING PRIMITIVES

An optimization with the creation of a buffer of varying size, cancelling *printf* system calls, brings a very consistent performance gain as shown in Figure 38, leading to the conclusion that per block trampoline insertion can be feasible and considered.

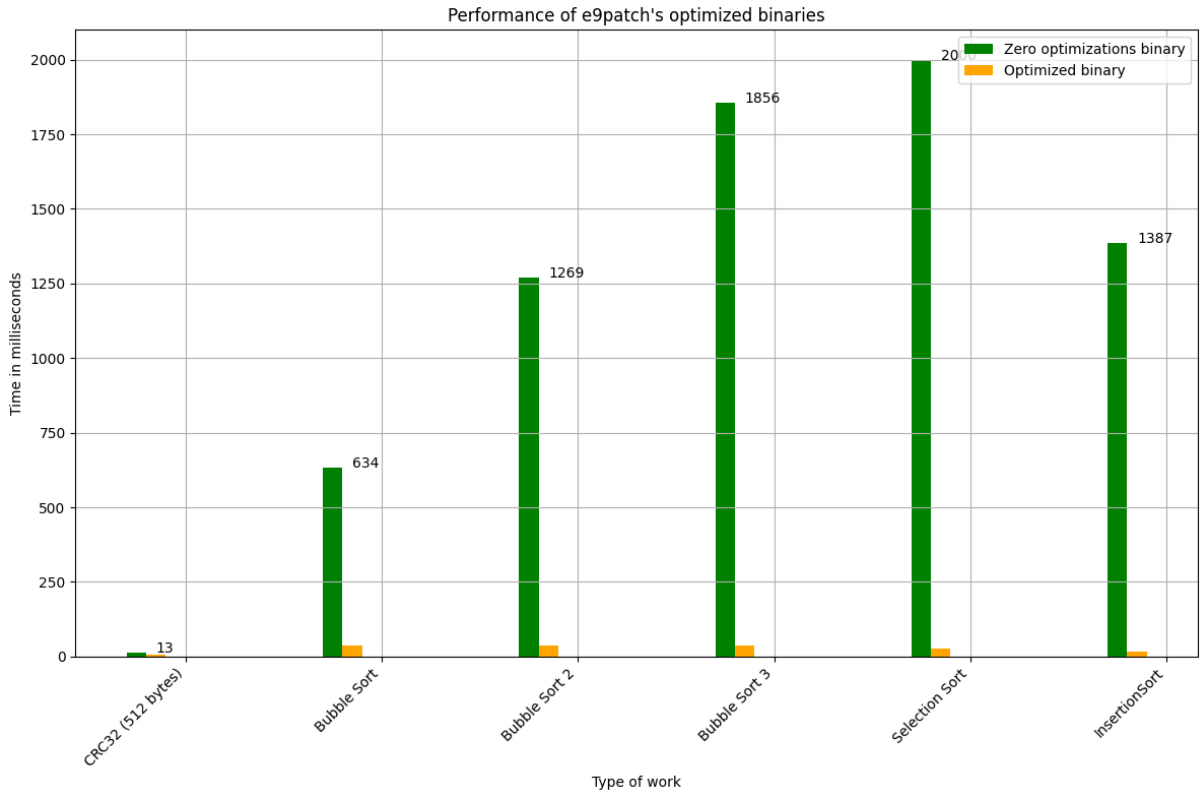

FIGURE 38. GAIN BROUGHT LEVERAGING BUFFERS FOR TRAMPOLINE LOGS

To progress further on the network function, L2fwd, Intel's DPDK Data Plane Development Kit will be integrated and plugged, with the use of representative traffic data.

6. Acceleration functionality

6.1.1 Introduction

The ever-evolving digital landscape is driving the need for advanced solutions in network acceleration and artificial intelligence (AI) performance enhancement. Addressing this technological challenge, DESIRE6G is dedicated to exploring and developing novel solutions for optimizing both network and AI functions.

The aim of DESIRE6G is twofold. Firstly, we are exploring methods to enhance network function performance. With the constant growth of network data traffic and the diversity of services, it is imperative to develop mechanisms that can handle higher capacities and provide more efficient service delivery. Secondly, we are focusing on boosting AI performance, a critical component for many modern applications, ranging from autonomous vehicles to advanced data analytics.

Our research is not limited to a single hardware platform. We acknowledge the diverse hardware ecosystems available today, from programmable P4 switches known for their granular network control, to graphics processing units (GPUs) that serve as the backbone for high-performance computing and AI operations. DESIRE6G aims to propose solutions that are adaptable and scalable across these varying hardware systems.

This section will delve into the innovative research methodologies and solutions being explored and developed under the DESIRE6G project. Our objective is to contribute to the development of a future where network and AI capabilities can be scaled and optimized to unprecedented levels, leading to ground - breaking advancements in digital connectivity and intelligent systems.

6.1.2 HW acceleration

During the DESIRE6G project several accelerated network functions will be developed. So far, we have identified the following functions:

TABLE 5. HW ACCELERATION

Description	Network Function	NF type (selectable)	Relevant service(s)/ UC(s)	On which level it acts (e.g., packet or stream level)	HW type
UE2Service mapper	UPF (see 6.1.5.2)	Mobile NF	all	packet	Tofino
Usage reporting		Mobile NF	all	packet	Tofino
MAS telemetry	INT	Infra NF	UC1	packet	Tofino
Traffic steering	TS	Infra NF	UC1	packet	Tofino
UE telemetry	UET	Mobile NF	UC1	packet	
Softwarization in RAN	CU-UP	Mobile NF	all		Tofino+DPDK /Netronome
NF Routing	NFR	Infra NF	all	packet	Tofino (and optionally other hardware targets)
Load Balancing between packet processing network functions	Load Balancer	Infra NF		packet	Tofino (and optionally other hardware targets)
Heavy hitter offloading to hardware-accelerated data planes	Heavy hitter offloading	Infra NF		packet	Tofino (and optionally other hardware targets)
Transport adapter to interconnect DESIRE6G sites with various tunnelling technologies	Transport adapter	Infra NF		packet	Tofino
QoS-aware traffic management and network resource allocation	QoS Handler	Infra NF		packet	Tofino, software targets (DPDK or eBPF) (and optionally other hardware targets)

6.1.3 Application acceleration

As we approach the dawn of the sixth generation of wireless technology, we're preparing to witness a significant transformation in the operations of sophisticated applications such as autonomous robots, drones, and other intelligent devices. Among the distinguishing features of 6G, the capacity for in-network computation stands out. This feature contrasts with traditional models where computations are predominantly carried out at endpoint devices or within the cloud; instead, 6G encourages computational processing directly within the network infrastructure.

This transformative approach is set to have major implications for the future of advanced applications. With 6G, networks are slated to evolve from simply being conduits for data transmission to becoming proactive participants in data processing and decision-making.

Applications that necessitate real-time responsiveness, like autonomous robots and drones, stand to gain considerably from the ultra-low latency that in-network computation in 6G networks can offer. By enabling necessary data processing closer to the source, these networks can facilitate faster decision-making and more prompt responses, crucial aspects for such systems.

In addition, the incorporation of in-network computation within 6G networks enhances both reliability and resilience, qualities that are critical for operations where safety is paramount. By reducing reliance on potentially remote cloud servers, these networks can avoid potential connectivity issues, thereby providing a more steady and dependable service.

However, resources situated within the network and at the edge are finite, and considering the anticipated surge in resource demand - attributed both to the growing number of potential applications and the increasing complexity of AI models used - there is an acute need to optimize the way these applications operate.

Addressing this need, the DESIRE6G project will focus on developing techniques for the optimization of Neural Network computations. By harnessing the capabilities of 6G networks, DESIRE6G aims to maximize the efficiency of resource utilization and pave the way for the future of in-network computation.

6.1.4 Neural Network Acceleration

In DESIRE6G we use SOL [48] for the acceleration of Neural networks. The mission of the SOL project is to transparently accelerate neural network workloads, with as few computing overhead as possible. We integrate SOL into neural network frameworks and where it attaches to the neural network.

6.1.4.1 The SOL Neural Network Acceleration Concept

SOL takes over the control of neural networks and reshapes their execution process. It analyses the underlying structure and applies a series of optimizations (including operation reordering, loop merging, kernel fusion, etc.) to maximize the data reuse in neural network layers, to utilize caches and other on-chip memories more efficiently and optimize external library parameters to deliver optimal performance. While it alters the computations it ensures that we do not alter the results (i.e., the operations before and after SOL are mathematically equivalent). Therefore, we rely solely on optimizations to instructions, caching, and workflows and do not employ alternative algorithms, approximations, data types with lower accuracy or other methods that could influence the results.

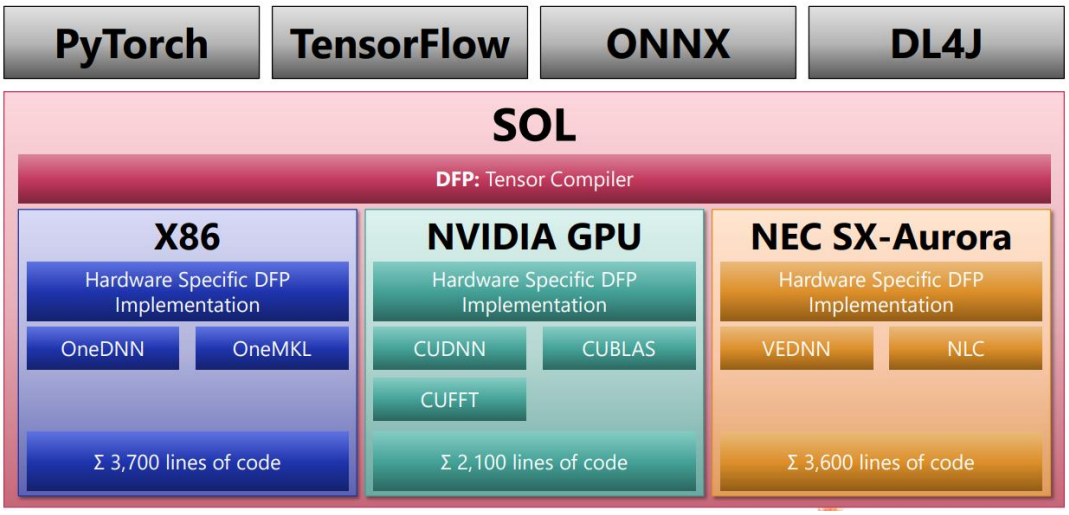

FIGURE 39. SOL NN ACCELERATION ARCHITECTURE

Figure 39 shows the architecture of the SOL compiler. It allows the optimization of models developed in the most popular AI frameworks such as PyTorch and TensorFlow, as well as the optimization for different hardware types, including CPUs, GPUs and specific accelerators such as the NEC SX-Aurora.

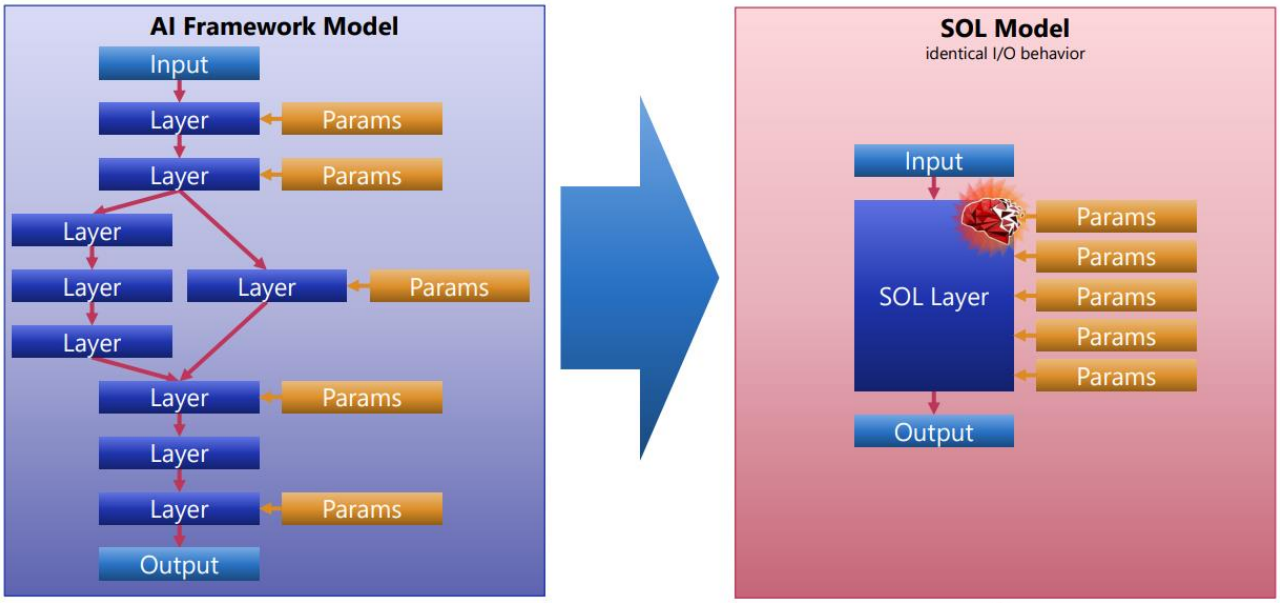

FIGURE 40. SOL OPTIMIZATION EXAMPLE

Figure 40 shows an example of the optimization performed by the SOL compiler. It analyses the structure of the NN model to simplify and optimize it as much as possible, using to this end different techniques and low-level compilers that optimize the performance of the global model utilization.

6.1.4.2 Advances in the SOL engine

During this reporting period, several improvements has been done to the SOL Neural Network acceleration engine. In particular, we are working towards the release of the version 0.5.3 of the engine. Here, we enumerate the main advances:

DNN-RNN Module

In SOL's RNN implementation we added support for non-standard activations, masking and recurrent dropout which are features not available in any DNN library (such as OneDNN or CUDNN) but can be used in the Keras API. However, in these cases TensorFlow falls back to utilize more generic layers, which result in a very inefficient implementation. To not suffer from similar performance issues, SOL's RNN implementation is highly specializable using C++ templates, and works on X86, CUDA and NEC SX-Aurora with the same code base. It achieves performance on par with CUDNN for standard RNN layers, and up to 50x higher performance for non-standard RNNs compared to TensorFlow.

Further, we have been able to reduce peak memory consumption in training by factor 2.5x in our testcase, which enables to run this test case on GPUs with limited memory (8GB in our testserver), while the same RNN fails within TensorFlow.

Last, we have improved numerical stability for cases where activations such as Softmax use large input values (e.g. exp(90)) which results otherwise in inf values, which don't allow to properly compute the Softmax result.

Framework-Identical Pseudo Random Number Generation

To enable identical results during training, we implemented the identical random number generation algorithms. During this process we ran into several issues which we reported to PyTorch and TensorFlow. E.g., PyTorch's `torch.bernoulli(...)` function returns different results, depending if the input is a Tensor or a Scalar. Further, in PyTorch prior v2.0 different random numbers have been generated for Intel and AMD CPUs. In TensorFlow random numbers using the identical local seed value produce different results in eager and graph mode.

We implemented this behaviour (if possible) also in SOL to guarantee identical random numbers and will dynamically enable the correct versions based on the framework's version, once they get fixed.

During out implementation we identified that esp. PyTorch does not use a single random algorithm but is based on different implementations. While TensorFlow always uses the identical PHILOX algorithm for all devices (CPU and GPU), PyTorch uses different algorithms for:

- X86 Bernoulli(Tensor): a modified version of MT19937
- X86 Bernoulli(Scalar) (e.g., ReLU): MKL's VSL Bernoulli Algorithm
- CUDA Bernoulli(Tensor and Scalar): CURAND's Philox, with the exception that the thread order is different in both cases.
- X86 Rand uses the same MT19937 as X86 Bernoulli(Tensor)
- CUDA Rand uses the same PHILOX as CUDA Bernoulli(Scalar)

This required to have in total four different algorithms implemented for PyTorch, that don't allow to compare any results between devices. Even changing the `p` of the Bernoulli from Tensor to Scalar switched to a different algorithm or implementation, which make the results impossible to reproduce on another device.

Further, the previously mentioned behaviour of different random numbers between Intel and AMD CPUs have been fixed in PyTorch 2.0, making it impossible to reproduce numbers from older versions.

Framework Compatibility

We have further improved the compatibility of SOL with the frameworks. To be able to interface with these, SOL needs to a) connect to the framework's memory allocator and b) to inject a custom layer that

enables to execute SOL within the framework. Due to changing symbols within the libraries and compiler flags (e.g., version of C++ standard, or CXX_ABI) we now compile bridging modules directly on the target machines, that adjust themselves to the correct version of the frame and selecting the appropriate compiler flags. This enables us to support nearly all past and future versions (if the APIs don't get changed), without updating SOL.

6.1.4.3 The SOL Server

In order to use SOL in the Desire6G project we will develop a Server that will allow us to use SOL as a Service. That is, the network, would allow the optimization of NN models that are used on it. To this end, the SMO will be able to call the SOL Server and send to it the different pretrained NN models. Those models will then be optimized and stored in the Model Catalog, a special version of the Service Catalog in the SMO.

6.1.5 Hardware Accelerated DP NFs

6.1.5.1 Acceleration of Infra NFs with P4 data planes

DESIRE6G introduces different Infra NFs (see Table 1) needed for traffic steering, load optimization among NFs and supporting QoS-aware traffic management. All these network functions are needed to provide a unified data plane abstraction for operators to deploy and operate their network service as simply as cloud-native applications run in data-centres. Since these Infra NFs must handle large traffic volume and thus line-rate performance is required, all of them will be implemented in hardware-accelerated data planes like Intel Tofino-based switches.

For example, one realization of the load balancer block of LO-DP as a packet processing digital circuit design is described in Section 3.1.1. The key objects of the digital circuit (managed by the LO-CP) are the following:

- GetInstanceGroup is a longest prefix match lookup table (requiring TCAM memory space in the digital circuit) that can provide the instance group id (IGid) and instance group size (IGsize), provided that multiple different instance groups may be configured by the LO-CP
- GetInstanceAddress is an exact lookup table (hash table in the SRAM) that maps IGid and a number from 0 to IGsize-1 (this is the instance sequence number within instance group IGid) to the address of the given data plane instance.

We assume that each packet is tagged with a key field representing the packet group it belongs to. The configuration in the LO-DP consists of an instance group id (IGid) and the number of instances (IGsize) belonging to the group. At packet arrival, the key carried by the packet is hashed to an instance number (iid) between 0 and IGsize-1. The iid is then mapped to the address of the instance (iaddr) to be used for serving the packet and the packet will be forwarded to the given iaddr.

There are still some issues to be solved during the later phases of the project, i.e., how to handle instance up-or downscaling in an efficient way.

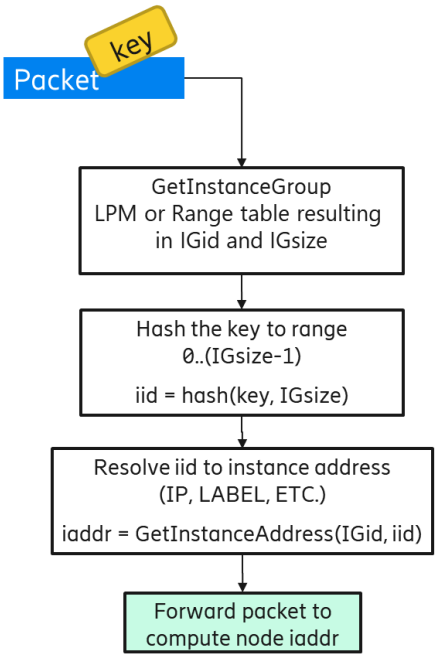

FIGURE 41. LOAD BALANCING SCHEMA IN THE LO-DP PACKET PROCESSING DIGITAL CIRCUIT

6.1.5.2 Acceleration of UPF with P4 data planes

The UPF functionality will consist of a UPF-CP and several UPF-DP elements, interconnected via a PDP aggregation proxy to ensure multi-tenant capability for the UPF-DP element(s) as shown in Figure 16.

The UPF-CP will be in charge of communicating with the 3GPP CN CP elements (e.g., SMF in 5G) in case when events triggering UP changes happen, e.g., when a new UE attaches to the network (and needs to be mapped to a service) or if an existing UE changes its position (and "access site" has been changed).

The DP functionality of the UPF will be achieved by a number of programmable components, some of them being the business logic components of the UPF, others being infrastructure NFs, controlled by the IML. The advantage of this setup is that the components of a specific UPF realizing a specific service

may differ from the components of an UPF of a different service both in terms of the business logic applied and the infrastructure NFs used. The different components may be accelerated separately, if needed. A list of example PDP components is given below:

- UE2Service mapper: its main role is to (based on info from UPF-CP) map the traffic from a specific user to a specific service and processing site. In downlink direction, it maps the destination IP address of incoming packets to a service id and RAN site location id. There are multiple options on how to do it, e.g.,
    - generate a DESIRE6G header with service id and RAN site location id and the id of the next NF of UPF-DP
    - if IPv6 address space is used, the upper 64bits may be used as UE identifier, and the lower 64 bits are then mapped to a private IP prefix describing the service, the RAN site location of the UE, and next NF to be used for processing.
- QoS handler (intra-slice): it ensures the proper resource sharing between the different users using the same service. Note that, as all other UP nodes, the UPF should implement the traffic management functionality to ensure proper QoS.
- Transport adapter: it is the first (and last) component of the UPF. It performs encapsulation/decapsulation if a specific tunnelling protocol is used between the sites. A selected set of tunnelling protocols may be supported. Note that in homogeneous DESIRE6G domains it may not be needed.
- Usage reporting: it sends usage reporting information to UPF-CP .

6.1.5.3 Acceleration of gNodeB (CU-UP) functionalities with P4 data planes

Next-generation NodeB (gNodeB) is located in the 5G Radio Access Network (RAN) close to the base station. In the downlink direction, 5G UPF forwards subscribers' traffic to the appropriate gNodeB node encapsulated into GPRS Tunnel Protocol for user data (GTP-U). The adoption of GTP tunnels will be maintained for 5G compliancy, while it will be investigated for the effective adoption in final DESIRE6G ecosystem. The gNodeB first decapsulates the downlink packets, ciphers the payload, adds PDCP (Packet Data Convergence Protocol) and RLC (Radio Link Control) headers, and then forwards the packet towards the user equipment. The gNodeB uses ARQ for reliable transport, meaning that all the downstream packets need to be stored in a packet buffer until an acknowledgment arrives. If the acknowledgment does not arrive within a timeout period, the packet is re-transmitted, and the appropriate acknowledgment timer is restarted. In the upstream pipeline, it has to handle

acknowledgments (RLC control packets) that are not forwarded, and uplink user data encapsulated in RLC/PDCP packets. Upstream user data must be deciphered and then forwarded via a GTP-U tunnel to a UPF instance in the 5G-Core.

Parts of the gNodeB functionality can be accelerated by P4 capable devices with a combination of hardware accelerator chips and software services. Conversely, the direction choice and header operations (encapsulation/decapsulation) can be implemented in P4, some operations such as encryption/decryption, or compression require specific hardware still can be handled in P4. Buffering, and time-based retransmission (ARQ) are not supported by P4 therefore the involvement of additional hardware or software solutions become necessary. Packet processing in the gNodeB branches along with the upstream and downstream directions, as shown in Figure 42. Note Figure 42 depicts our initial gNodeB implementation using a Tofino hardware.

The downstream direction receives GTP encapsulated packets. It saves the GTP TEID field, decapsulates the GTP header, and creates the RLC and PDCP headers with the PDCP serial number (SN) calculated from the TEID value. The newly constructed PDCP packet is sent to the encryption service, from where the encrypted packet is received asynchronously later. As a next step, the encrypted packet is cloned. One copy goes through the appropriate downstream port, while the other is transmitted to the buffering service.

The upstream direction handles incoming PDCP packets. PDCP protocol distinguishes status messages (acknowledgments) and data packets. Acknowledgments are forwarded to the buffering service without modification. While for data packets, the gNodeB has to rebuild the GTP header. After the PDCP and RLC header removal and the proper GTP encapsulation, the packet goes to the upstream port.

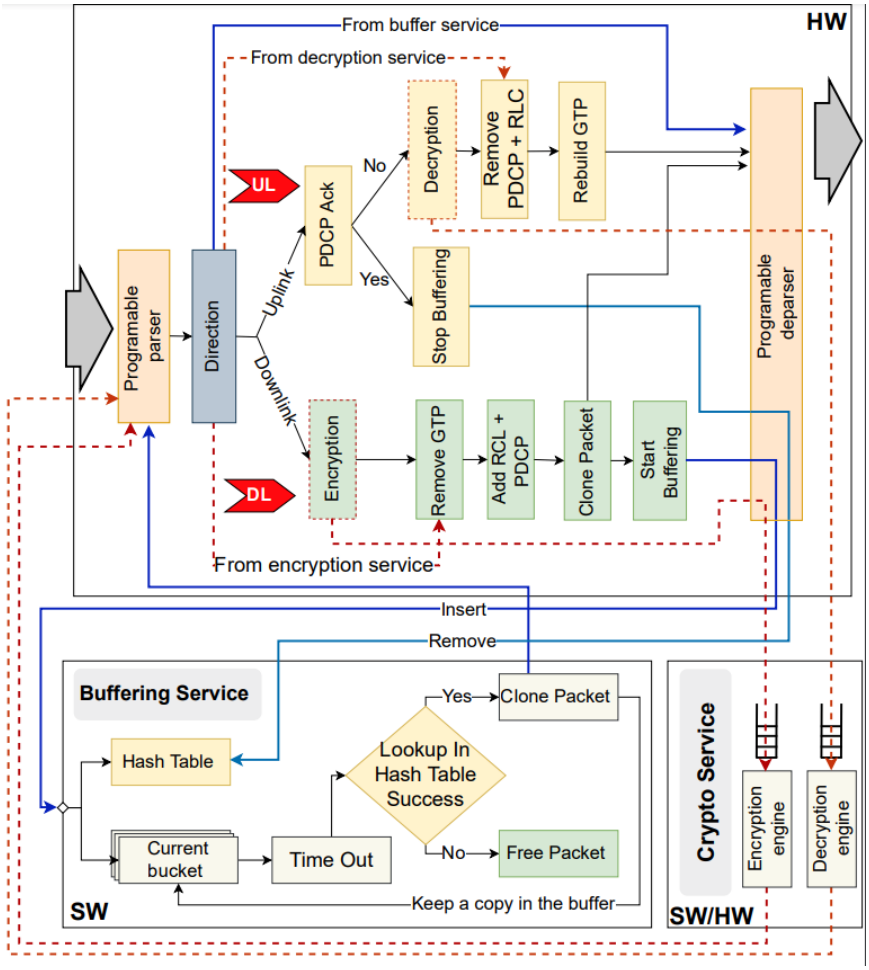

FIGURE 42. HYBRID GNODEB MODEL USING P4 PROGRAMMABLE HARDWARE AND SW COMPONENTS

A possible implementation of the algorithm for the applied Buffering Service can be reported in Figure 43. It was implemented as a DPDK-based software component and is responsible for storing packets until they are acknowledged, managing timers for acknowledgement reception and initiate retransmission if needed.

Algorithm 1 Buffering

```
1: i := 0; n := resubmission interval in ms
2: for j IN 1...n do
3:     bucket[j] := empty bucket
4: end for
5: while True do
6:     if 1 ms passed then
7:         i := (i + 1)%n
8:         for pkt IN bucket[j] do
9:             if hashtable.contains(pkt.id) then
10:                pkt2 = clone(pkt)
11:                send_out(pkt2)
12:            else
13:                bucket[j].remove(pkt)
14:                free(pkt)
15:            end if
16:        end for
17:    end if
18:    pkt := receive_packet()
19:    if pkt is PDCP_STATUS then
20:        hashtable.remove(pkt.id)
21:    else
22:        bucket[j].add(p)
23:        hashtable.insert(pkt.id)
24:    end if
25: end while
```

FIGURE 43. ALGORITHM FOR BUFFERING SERVICE.

6.1.5.4 Acceleration of CU-UP in smart NICs

The diagram below shows the hardware accelerated CU-UP architecture planned to be implemented in NFP-4000 SmartNIC card. This implementation uses XDP for hardware acceleration of CU-UP. The CU-UP uses the SmartNIC's crypto accelerator to accelerate PDCP ciphering and deciphering. The control plane or E1 interface part of CU-UP is terminated in the Arm11 core of the SmartNIC . The CU-UP data plane pipeline is implemented in the eBPF core. The eBPF flow processing cores can implement PDCP and SDAP protocols in the user plane pipeline.

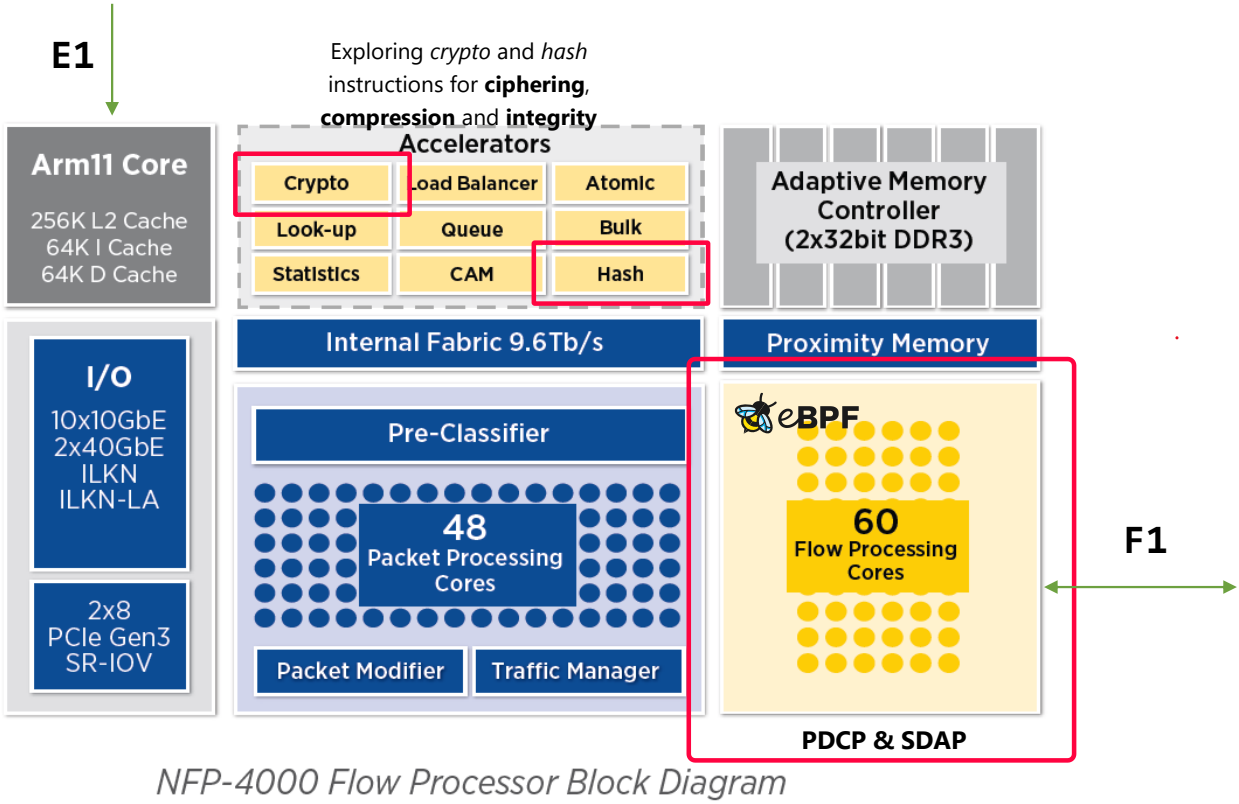

FIGURE 44. ACCELERATION ARCHITECTURE.

7. Conclusions

The deliverable summarizes the main achievements of the work done in WP4 in the first phase of the study, where the main target is to develop the unified programmable data plane (PDP) layer of DESIRE6G architecture. The main goal for this first phase was to develop the key concepts related to the PDP. Besides this, the study has also identified and documented some open issues that are mostly related to the implementation of the functionality proposed. In the next phase, the plan is to address the problems identified, and perform a first software implementation of the mandatory modules of the DESIRE6G unified programmable data plane layer.

8. References

[1] G. Pongrácz, A. Mihály, I. Gódor, S. Laki, A. Nanos, C. Papagianni, Towards extreme network KPIs with programmability in 6G MobiHoc '23: Proceedings of the Twenty-fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing, October 2023, https://doi.org/10.1145/3565287.3617610

[2] DESIRE6G Deliverable D2.1: "D2.1 Definition of Use Cases, Service Requirements and KPIs/KVIs"

[3] Sivaraman, V., Narayana, S., Rottenstreich, O., Muthukrishnan, S. and Rexford, J., 2017, April. Heavy-hitter detection entirely in the data plane. In Proceedings of the Symposium on SDN Research (pp. 164-176).

[4] Basat, R.B., Chen, X., Einziger, G. and Rottenstreich, O., 2020. Designing heavy-hitter detection algorithms for programmable switches. IEEE/ACM Transactions on Networking, 28(3), pp.1172-1185.

[5] Kumar, A., Jain, S., Naik, U., Raghuraman, A., Kasinadhuni, N., Zermeno, E. C., ... & Vahdat, A. (2015, August). BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing. In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (pp. 1-14).

[6] PIFO Sivaraman A, Subramanian S, Alizadeh M, Chole S, Chuang ST, Agrawal A, Balakrishnan H, Edsall T, Katti S, McKeown N. Programmable packet scheduling at line rate. InProceedings of the 2016 ACM SIGCOMM Conference 2016 Aug 22 (pp. 44-57)

[7] AIFO Yu Z, Hu C, Wu J, Sun X, Braverman V, Chowdhury M, Liu Z, Jin X. Programmable packet scheduling with a single queue. InProceedings of the 2021 ACM SIGCOMM 2021 Conference 2021 Aug 9 (pp. 179-193).

[8] RIFO Mostafaei H, Pacut M, Schmid S. RIFO: Pushing the Efficiency of Programmable Packet Schedulers. arXiv preprint arXiv:2308.07442. 2023 Aug 14.

[9] PPV Fejes F, Nádas S, Gombos G, Laki S. DeepQoS: Core-Stateless Hierarchical QoS in Programmable Switches. IEEE Transactions on Network and Service Management. 2022 Feb 16;19(2):1842-61.

[10] RFC 9330 - https://www.rfc-editor.org/rfc/rfc9330.txt (last access: 27/11/2023)

[11] Nádas S, Turányi ZR, Rácz S. Per packet value: A practical concept for network resource sharing. IEEE Global Communications Conference (GLOBECOM) 2016 Dec 4 (pp. 1-7).

[12] Fejes F, Nádas S, Gombos G, Laki S. DeepQoS: Core-Stateless Hierarchical QoS in Programmable Switches. IEEE Transactions on Network and Service Management. 2022 Feb 16;19(2):1842-61.

[13] Harkous H, Papagianni C, De Schepper K, Jarschel M, Dimolianis M, Pries R. Virtual queues for P4: A poor man's programmable traffic manager. IEEE Transactions on Network and Service Management. 2021 May 3;18(3):2860-72.

[14] S. S. Kunniyur and R. Srikant, "An adaptive virtual queue (AVQ) algo- rithm for active queue management," IEEE/ACM Trans. Netw., vol. 12, no. 2, pp. 286–299, Apr. 2004

[15] A. Eryilmaz and R. Srikant, "Fair resource allocation in wire- less networks using queue-length-based scheduling and congestion control," IEEE/ACM Trans. Netw., vol. 15, no. 6, pp. 1333–1344, Dec. 2007.

[16] Iannone L, Saucez D, Bonaventure O. Implementing the locator/id separation protocol: Design and experience. Computer Networks. 2011 Mar 10;55(4):948-58.

[17] Samdanis K, Taleb T. The road beyond 5G: A vision and insight of the key technologies. IEEE Network. 2020 Feb 19;34(2):135-41.

[18] Kubernetes (k8s): https://kubernetes.io/ (last access: 27/11/2023)

[19] P4 runtime specifications: https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html (last access: 27/11/2023)

[20] networkservicemesh.io: https://networkservicemesh.io/ (last access: 16/10/2023)

[21] https://vaccel.org. (last access: 27/11/2023)

[22] https://github.com/nubificus/qemu-vaccel (last access: 27/11/2023)

[23] https://github.com/nubificus/firecracker (last access: 27/11/2023)

[24] https://katacontainers.io (last access: 27/11/2023)

[25] DESIRE6G Deliverable D2.2: "D2.2 Functional Architecture Definition"

[26] DESIRE6G Deliverable D3.1: "D3.1 Intelligent and secure management, orchestration, and control platform"

[27] Kafka: https://kafka.apache.org/ (last access: 16/10/2023)

[28] The P4.org Applications Working Group.[n.d.]. In-band network telemetry data-plane specification 2.1. https://p4.org/p4-spec/docs/INT_v2_1.pdf

[29] Papadopoulos K, Papadimitriou P, Papagianni C. Deterministic and Probabilistic P4-Enabled Lightweight In-Band Network Telemetry. IEEE Transactions on Network and Service Management. 2023 Aug 3.

[30] F. Alhamed, et al., "P4 Telemetry collector", Computer Networks, Vol 227,2023, 109727, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2023.109727.

[31] Youngho Kim, Dongeun Suh, and Sangheon Pack. 2018. Selective in-band network telemetry for overhead reduction. In 2018 IEEE 7th International Conference on Cloud Networking (CloudNet). IEEE, 1–3.

[32] Shaofei Tang, Jiawei Kong, Bin Niu, and Zuqing Zhu. 2020. Programmable Multilayer INT: An Enabler for AI-Assisted Network Automation. IEEE Communications Magazine 58, 1 (2020), 26–32.

[33] Dongeun Suh, Seokwon Jang, Sol Han, Sangheon Pack, and Xiaofei Wang. 2020. Flexible sampling-based in-band network telemetry in programmable data plane. ICT Express 6, 1 (2020), 62–65.

[34] Shaofei Tang, Deyun Li, Bin Niu, Jianquan Peng, and Zuqing Zhu. 2019. Sel-INT: A runtime-programmable selective in-band network telemetry system. IEEE transactions on network and service management 17, 2 (2019), 708–721.

[35] Goksel Simsek, Doğanalp Ergenç, and Ertan Onur. 2021. Efficient network monitoring via in-band telemetry. In 2021 17th International Conference on the Design of Reliable Communication Networks (DRCN). IEEE, 1–6.

[36] Dandan Mo, Zhiruo Liu, Du Chen, and Deyun Gao .2021.C-INT:An Efficient Cluster Based In-Band Network Telemetry. In 2021 4th International Conference on Hot Information-Centric Networking (HotICN). IEEE, 129–134.

[37] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: Probabilistic in-band network telemetry. In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. 662–680.

[38] Enge Song, Tian Pan, Chenhao Jia, Wendi Cao, Jiao Zhang, Tao Huang, and Yunjie Liu. 2021. INT-label: Lightweight In-band Network-Wide Telemetry via Interval-based Distributed Labelling. In IEEE INFOCOM 2021 - IEEE Conference on Computer Communications. 1–10. https://doi.org/10.1109/INFOCOM42981.2021.9488799

[39] Abdulkadir Karaagac, Eli De Poorter, and Jeroen Hoebeke. 2020. In- Band Network Telemetry in Industrial Wireless Sensor Networks. IEEE Transactions on Network and Service Management 17, 1 (2020), 517–531. https://doi.org/10.1109/TNSM.2019.2949509

[40] D. Scano, et al. "Extending P4 in-band telemetry to user equipment for latency- and localization-aware autonomous networking with AI forecasting," in JOCN, vol. 13, no. 9, pp. D103-D114, September 2021, doi: 10.1364/JOCN.425891.

https://ieeexplore.ieee.org/abstract/document/9481776

[41] Knossen S, Hill J, Grosso P. Hop recording and forwarding state logging: Two implementations for path tracking in p4. In2019 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS) 2019 Nov 17 (pp. 36-47). IEEE.

[42] Nagios https://www.nagios.org/ (last access: 27/11/2023)

[43] Zabbix https://www.zabbix.com/ (last access: 27/11/2023)

[44] Prometheus https://prometheus.io/ (last access: 27/11/2023)

[45] S. Shalunov et al., «Application-Layer Traffic Optimization (ALTO) Protocol», Internet Engineering Task Force, Request for Comments RFC 7285, sep. 2014. doi: 10.17487/RFC7285.

[46] kube-state-metrics KSM https://github.com/kubernetes/kube-state-metrics (last access: 27/11/2023)

[47] cAdvisor https://github.com/google/cadvisor (last access: 27/11/2023)

[48] https://sol.neclab.eu/ (last access: 27/11/2023)

[49] Velasco L, Ruiz M, Gonzalez P, Paolucci F, Sgambelluri A, Valcarenghi L, Papagianni C. Pervasive monitoring and distributed intelligence for 6G near real-time operation. In European Conference on Networks and Communications (EUCNC) 2023.