Computer Networks 246 (2024) 110397

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

# Leveraging Data Plane Programmability to enhance service orchestration at the edge: A focus on industrial security

Gaetano Francesco Pittalà [a,*], Lorenzo Rinieri [b], Amir Al Sadi [b], Gianluca Davoli [a], Andrea Melis [b], Marco Prandini [b], Walter Cerroni [a]

[a] Department of Electrical, Electronic, and Information Engineering "Guglielmo Marconi" (DEI), University of Bologna, Italy
[b] Department of Computer Science and Engineering (DISI), University of Bologna, Italy

ARTICLE INFO

*Keywords:*
Service orchestration
Edge Computing
Data Plane Programmability
Industrial security
P4

ABSTRACT

The Edge Computing paradigm is increasingly gaining traction in modern telecommunication scenarios, as it enables the offloading of computational tasks from end devices to a variety of nodes located in close proximity to them. This approach is essential for meeting the ever-stricter Quality of Service requirements imposed by modern applications. Concurrently, the advent of Data Plane Programmability allows for unmatched flexibility on the networking plane, supporting processing of multiple protocols in a logically centralized fashion with simple in-line computation, and offering the possibility to offload additional services to networking equipment. Reaping those benefits necessitates heedful management of resources and infrastructure. This, in turn, calls for the introduction of a service orchestration entity, capable of taking advantage of device heterogeneity to enable efficient and swift service provisioning. This work delves into the potential of introducing an orchestration system able to cope with the challenges of offloading security tasks at the Edge. This effort involves developing and implementing novel architectural components that capitalize on the heterogeneous nature of the Edge infrastructure as well as of the Programmable Data Plane as a potential tool for service offloading. To establish the feasibility and performance of this approach, an industrial scenario is considered, where the integrity of data from legacy devices must be ensured. Following an evaluation of the hashing performance of the Programmable Data Plane in comparison to general-purpose devices, a simulation study is conducted on the overall orchestration system, demonstrating the viability of the proposed approach.

## 1. Introduction

In recent years, due to the increasing pervasiveness of data processing capabilities across the whole network, service instantiation has been moving from core networks towards the elements that generate the data itself, located at the edge of the infrastructure, leading to the emergence of the Edge Computing (EC) paradigm [1]. User devices as well as data sources can greatly benefit from this transition, as they are in constant demand for real-time and latency-sensitive processing [2]. Along with meeting these requirements, the adoption of EC also promises an optimization of network traffic, an improvement of the user experience, and an enhancement of privacy and security for a number of applications, including industrial automation. Indeed, EC can be pivotal in augmenting services offered by the network with new processing capabilities, making up for the scarcity of computing power or software compatibility typically displayed by specialized equipment, e.g., by providing low-power legacy devices with packet encryption functionalities. In doing that, Data Plane Programmability (DPP) can be of great help, as devices in the Programmable Data Plane (PDP) are very flexible when it comes to performing simple and quick data processing on the traffic they handle, opening up to a large set of functionalities that can be offered to network services.

In industrial environments, clusters of sensors and actuators are typically deployed across the scenario, in arrangements often referred to as Industrial Internet of Things (IIoT) networks. They present specific architectural challenges and weaknesses [3], stemming from their peculiarities. Indeed, such networks are usually heterogeneous, unsuitable for computation-intensive operations, and prone to security issues, due to their devices needing to always be connected to the Internet while lacking energy-demanding security software modules. DPP-based solutions can help tackle these problems, in multiple ways. The Programming Protocol-Independent Packet Processors (P4) language has emerged as a powerful tool to control PDP devices, allowing network operators and service providers to define the behavior of the network devices at a packet-level granularity, enabling them to create

* Corresponding author.
*E-mail address:* francesco.pittala@unibo.it (G.F. Pittalà).

https://doi.org/10.1016/j.comnet.2024.110397
Received 31 October 2023; Received in revised form 15 February 2024; Accepted 3 April 2024
Available online 9 April 2024

custom forwarding and processing pipelines tailored to specific applications [4]. By using P4 to include cross-level headers, packet processing can be offloaded to a single or few network devices provisioned with more resources, supporting simple encryption or integrity checking. Moreover, DPP can facilitate the adoption of security countermeasures without adding complexity to the IIoT devices.

In this paper we explore the potential of employing DPP as a tool to enhance service offloading at the Edge, focusing on industrial security applications. Including DPP in a framework for flexible service provisioning, we can enable the customization and adaptability needed to improve services offered at the Edge. This calls for the introduction of a service orchestration entity, capable of taking advantage of a pool of heterogeneous (computing and networking) resources to enable efficient and swift service provisioning. In other words, we abstract the functionalities of PDP devices and make them comparable to those of computing nodes, then we apply policy-based placement strategies to pick the most suitable resource to provide a given service. As a case study, we refer to an industrial environment, where the security of remote maintenance services can be enhanced by activating proper services offered at the Edge. We emulate the use case scenario to evaluate the performance of implementing data integrity functions on PDP devices in comparison to general-purpose computing nodes. Finally, leveraging on the results of the emulation, we assess through simulation how the availability of P4 switches can improve the efficiency, performance, and scalability of service offloading. In the context of this work, we refer as *service* to the composition of functionalities (e.g., data processing, traffic steering, etc.) that may be deployed in a distributed way over the network and EC infrastructures, and typically offered to the user as a cohesive bundle. *Service components* (i.e., the single parts being composed) are the abstracted elements representing the underlying physical or virtual resources that can be configured to perform those functionalities.[1]

The remainder of the paper is structured as follows. In Section 2 we offer a recollection of state-of-the-art security challenges for industrial environments, as well as solutions for service orchestration at the Edge. In Section 3 we introduce the architecture and working principles of the service orchestration framework we employed. In Section 4 we present the scenario in which this work is articulated, outlining the typical structure of an industrial network, its main challenges, and what issues we aim to solve with our proposed solution. In Section 5 we describe the topology for our emulated environment and provide its evaluation, along with a commentary of preliminary results that assess the potential of employing PDP devices in the delineated orchestration context. In Section 6 we showcase through simulation results the benefits offered by our approach in the offloading of services at the Edge, focusing on the described use case. In Section 7 we draw conclusions and highlight future research directions.

## 2. Related work

In this section, we first review research efforts that highlight the potential of employing DPP to cope with shortcomings of IIoT networks, with a focus on processing offloading to network devices. We then provide a summary of state-of-the-art solutions for the orchestration of services at the Edge. In doing that, we use the terms EC and Fog Computing (FC) interchangeably, as their definition may overlap in scenarios such as the one depicted in this work. Indeed, from the point of view of cloud service providers, EC resources are typically referred to as "near edge", whereas FC resources are classified as "far edge", justifying the terminology we adopt here [5].

[1] In the following, depending on the context, for the sake of readability we may refer to "the function performed by a single service component" simply as a *service*.

### 2.1. Data plane programmability for in-network offloading

DPP enables reshaping packet processing procedures to smartly exploit the hardware capabilities of networking resources. This architectural advantage can be leveraged to implement custom packet programming for multiple purposes. PDP devices can be employed to support security and ease communication within IoT environments.

Some work analyzing this matter can be found in the literature. P4 can be employed in IIoT scenarios, as argued in [6] and in [7], which propose a framework to describe in-network computation tasks, respectively to support AI-based scenarios and event detection over a publish/subscribe architecture (using hardware targets like FPGAs and SmartNICs).

P4 can also enable communication over complex wireless networks, as argued in [8], as well as [9,10], which provide examples of how to exploit P4 devices to enable peer-to-peer communication, manage link load and enable WiFi communication over distributed Internet of Things (IoT) wireless networks. The common denominator of these approaches is that P4 targets introduce little to no communication overhead.

One of the few DPP-based IoT architectural solutions for multi-access networks is proposed in [11], employing P4 and In-band Network Telemetry (INT) [12] over a P4 target to dynamically offload tasks on the network.

Hence, in-network offloading can help service management systems drastically cut some operational overhead and allow for finer-grained network inspection. This is especially relevant in environments revolving around IIoT, usually composed of nodes with limited computational power. Moreover, these networks use heterogeneous means of communication and require ad hoc clients and servers to send and receive information. Thus, DPP and P4 can be helpful to potentially offload part of the server-side computation since programmable targets can be used and programmed like regular nodes. In fact, a P4 switch can handle multiple or even custom protocols, and hence be used as gateways for communications inside the plant, as well as run simple in-line computation when processing packets. These features allow for service offloading on the data plane, so the network devices can be instrumented to perform network services computation at line rate. In this paper, we show the advantages of such offloading features in an IIoT context focusing on security applications.

### 2.2. Service orchestration at the edge

The EC paradigm enables efficient and distributed execution of computational tasks closer to the network's edge, bringing significant benefits in terms of latency, privacy, and bandwidth utilization. However, effective resource orchestration in EC environments remains a challenging task, due to the dynamic and heterogeneous nature of the resources present in such systems.

While substantial steps have been made regarding the standardization of service orchestration systems, such as OpenFog [13] and Mobile Edge Computing (MEC) [14], open-source implementations of fully-compliant orchestration systems are still lacking. However, we present here a comprehensive overview of state-of-the-art orchestration systems, to give context to the system we relied on for our evaluations.

In [15] the authors propose a novel orchestration architecture for FC environments. Such architecture is divided into three tiers, namely "cloud tier", "edge cloudlets", and "edge gateways", and arranged by the distance from the user to the requested resources (e.g., the edge gateway nodes are positioned closer to the user than the edge cloudlets nodes, offering applications with better latency performance). The workload placement is regulated to meet the demands of fog applications.

The authors of [16] created a prototype EC orchestrator, addressing the heterogeneity of the IoT environment as well as the capabilities of involved devices and the imposed constraints. The orchestrator is

logically centralized, and an agent needs to run in every controlled node, to manage local virtual instances (i.e., containers). They focused their evaluation on two critical performance factors, namely the time performance of the system while orchestrating different services in different configurations (e.g., varying image size, image location, etc.), and the success rate of the system in instantiating the services as requested.

A hybrid architecture to manage resources in the Fog-to-Cloud continuum is presented in [17]. It suggests a solution for the distributed management of applications and services in the IoT and Fog domains, while it recommends using a centralized strategy for the orchestration in the Edge and Cloud domains, to benefit from global awareness of the resources present in the network.

In [18], the authors provide a layered and modular architecture with containerized services and microservices that operate on the Fog-to-Cloud continuum. A hierarchically lower layer is responsible for performing sensing and operations, while a middle layer handles intermediate computing resources and routing, and an upper layer deals with wider-view operations such as long-term global storage.

The aforementioned works propose new architectures for service orchestration in EC scenarios, emphasizing the distributed character of these systems and the demand for accurate monitoring and data on the availability of resources for both nodes and services. These solutions, however, only combine the information on available resources to pick a node on which to deploy the service, without considering different service provisioning models, nor considering the specific performance users expect of services after their activation.

On the contrary, the authors of [19] propose a service-centric approach, that leverages the inherent flexibility offered by the cloud-native Everything-as-a-Service (XaaS) model to provision services in a more dynamic way. This solution still makes use of data gathered from available resources to decide where and how to deploy services, but in doing so it also considers the nature of the requested service, including the possibility of deploying it in multiple ways, along with the present state of the system. This orchestrator was subsequently extended in [20] and in [21], making it a suitable choice for our purposes here.

The novelty of this paper resides in (i) the adoption of PDP devices for the offloading of security-critical network service functions in an IIoT scenario, and (ii) the inclusion of DPP as a tool to enhance availability and flexibility of services in a XaaS-aware service orchestration framework.

## 3. Orchestrating services leveraging programmable data plane devices

Orchestrating services over a heterogeneous set of resources requires cooperation between monitoring, management, and decision processes. In this section, we describe the structure, functional elements, and working principles of the service orchestration framework we employed. Such framework is based on that introduced in [19,20]. It retains the original XaaS approach, which makes the system aware of different service deployment models, borrowing from the XaaS deployment paradigm typical of Cloud Computing scenarios, and extends it with the support for services offered by networking resources. Specifically, the orchestrator can instantiate services according to five different service provisioning paradigms:

- Infrastructure-as-a-Service (IaaS), for the deployment of a generic virtualization engine (e.g., Docker) on a computing resource;
- Platform-as-a-Service (PaaS), to provide the user with a software framework (e.g., the Python SDK) including tools, libraries, and interpreters, for the execution of generic programs;
- Software-as-a-Service (SaaS), to offer a specific application (e.g., a Web-based one) that users may access through a dedicated interface;

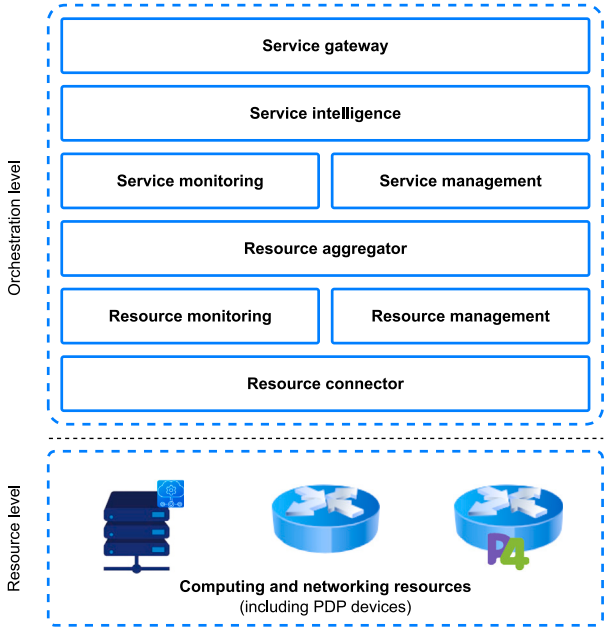



**Fig. 1.** Service orchestration framework architecture.

- Function-as-a-Service (FaaS), to provide the user with a lightweight service component, reduced to a single function (e.g., real-time video transcoding), handled entirely by the computing resource on which it is deployed in an event-driven, serverless manner;
- Programmable-Data-Plane-as-a-Service (PDPaaS), exposing to the user packet-level functionalities of programmable networking resources (i.e., PDP devices), e.g., to steer traffic or process packets as they cross the network, with potentially significant performance advantages.

Leveraging the flexibility offered by this approach, the orchestration system is able to combine different paradigms to obtain the desired service in an efficient and effective way. For instance, if a service is not natively available on a computing resource, but such resource is capable of hosting a compatible version of it, then the requested service can be deployed there. In other words, the orchestrator may decide to deploy an application (i.e., a SaaS-native element) on top of a virtualization engine (i.e., a IaaS-native element) running on a computing resource that did not previously offer that specific application. This introduces great adaptability, albeit at the expense of greater service activation complexity that may entail drawbacks such as an increased service activation time.

The architecture of the orchestration system is represented in Fig. 1. At a macroscopic level, the framework consists of two layers, namely the *Orchestration level* and the *Resource level*. The former comprises the functional elements of the logically centralized orchestrator, while the latter encompasses all the distributed and dynamic resources that are available in the infrastructure for the orchestrator to activate services.

The functional elements of the service orchestrator are structured in a way that reflects the need for abstraction of the service activation process, while also facilitating a modular implementation. Starting from the top of the figure and moving downwards, the point of contact between the orchestrator and the external world is provided by the *Service gateway*, which allows users and other systems to access the functionalities of the orchestrator. Immediately below, it comes the *Service intelligence* element, which implements the core functionalities of the orchestration process; it is tasked with making service activation decisions to satisfy external requests while enforcing predefined policies. The monitoring information is made available to the decision-making module by the *Service monitoring* element, which handles the

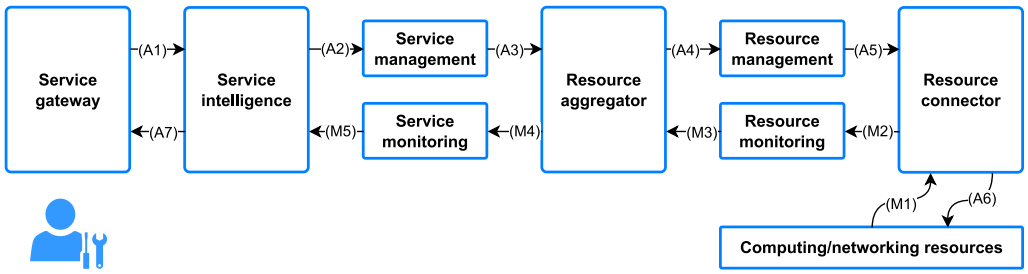



**Fig. 2.** Interactions between functional elements of the orchestration system.

processed telemetry data that describes the status of active services and the availability of components to instantiate new ones. Its functions are complemented by the *Service management* element, which handles the lifecycle of services, ensuring that their deployment and decommissioning is operated in compliance with the decisions coming from the *Service Intelligence*. The role of the *Resource aggregator* element is that of providing the abstractions that allow the elements above to work with abstracted service components that lack any technology-specific details. Similarly to their overlaying counterparts, the *Resource monitoring* and *Resource management* elements are in charge of collecting telemetry data and handling the deployment processes, respectively, but at a lower level of abstraction, interacting with resource domains in a way that is specific to their technological characteristics. Lastly, the *Resource connector* element is in charge of the communication between the service orchestrator and the underlying infrastructure, facilitating the integration of diverse domains into a pool of resources visible to the orchestration processes.

For this work, we introduced the support for some additional services that may leverage DPP, depending on the availability of PDP devices. We make the practical assumption that the code needed in order to provision the newly introduced services is already available in the PDP devices, so as to simplify their deployment by just activating them when needed, without running a re-configuration of the pipeline.

The procedure of activating a service through the orchestration system typically starts with the intended user requesting a list of the offered services. This list reports information in a symbolic format, representing the available services as well as those that the orchestrator can deploy. The user may then request the activation of a specific service. The activation process is entirely handled by the cooperation among the functional elements in the service orchestrator. The steps of such process are displayed in Fig. 2, where the interactions pertaining to the activation of services are labeled with *A*, whereas those related to the monitoring process are labeled with *M*. The former chain of actions is triggered by the request coming from the user, while the latter happens periodically. The numbers on the labels represent the order in which those interactions take place. In summary, when receiving a service activation request, the orchestrator will leverage the periodically refreshed monitoring information on the available resources to determine how to activate the requested service. It will then trigger the required steps to configure the underlying resources for the provisioning of the service to the user. At the end of the procedure, the orchestrator will inform the user of the outcome, allowing it to access the service.

To showcase the support for the PDPaaS paradigm, we extended the orchestrator with the ability to interact with the PDP. From the point of view of the orchestrator, and specifically of its *Service intelligence* element, all resources are comparable, as they are described in terms of their features by means of abstractions provided by the *Resource aggregator* element. This makes it so that PDP devices are regarded in the same way as any other resource, allowing PDP resources to be included, alongside computing resources, in the same pool from which the placement algorithm picks for the activation of the requested service. Also consistently with the implementation of the other instances of the XaaS paradigm, the same principles for the monitoring and management of computing resources are applied to PDP resources. In practice, this translates to the introduction of a REpresentational State Transfer (REST) interface on the control plane of the programmable switches, enabling the interaction with the orchestrator for monitoring and management purposes.

The principles and mechanisms described in this section can be leveraged to address significant issues in the reference scenario, as detailed in the following.

## 4. Industrial security: a focus on legacy devices

In this section, we outline the Industrial Control Systems (ICS) context to which the use case discussed in this paper pertains. We specifically describe some of the most important challenges on the orchestration and management of services in this context, and how our solution can help tackle some of those challenges, such as those related to remote maintenance. ICS are composed of interconnected Cyber and Physical components that monitor and manage physical processes. They are responsible for the safety and operations of the industrial process, which implies the management of heterogeneous hardware and software. They include devices such as sensors, actuators, supervisory control and data acquisition (SCADA) systems, Human Machine Interfaces (HMI), and dedicated subsystems such as programmable logic controllers (PLC) [22]. This heterogeneity obviously translates into system complexity, which implies more effort to manage and prevent anomalies. The Purdue Enterprise Reference Architecture [23] is the reference networking architecture for ICS systems, adopted in the ANSI/ISA-95 standard, and we can use it to analyze each segment of a typical ICS.

As depicted in Fig. 3, the Purdue Architecture divides the ICS network into six layers which are arranged into three logical segments: the layers from 0 to 3 constitute the Manufacturing Zone, while levels 4 and 5 constitute the Enterprise Zone, with a Demilitarized Zone of convergence between them. The Enterprise Zone, also referred to as Information Technology (IT) network, incorporates traditional IT devices and systems where the primary business functions of the enterprise occur, including the orchestration of manufacturing operations and services. On the other hand, the Manufacturing Zone is known as Operational Technology (OT) network because it contains systems and devices responsible for the control, monitoring, and automation of physical processes. At level 0 of the Manufacturing Zone, sensors and actuators are deployed to interact directly with the physical process while level 1 is composed of PLCs which implement systems control logic by observing sensor readings and by consequently updating actuators signals. Level 2 (SCADA, HMI) and level 3 devices are responsible for control, data acquisition, and monitoring in order to manage plant operations. Edge nodes at level 3 are also responsible for running applications that need to interact with OT devices and for providing security for the Manufacturing network. In addition, devices belonging to those two levels can communicate with the Enterprise Zone through the demilitarized zone (DMZ), which manages the connection between the IT and the OT networks while maintaining the

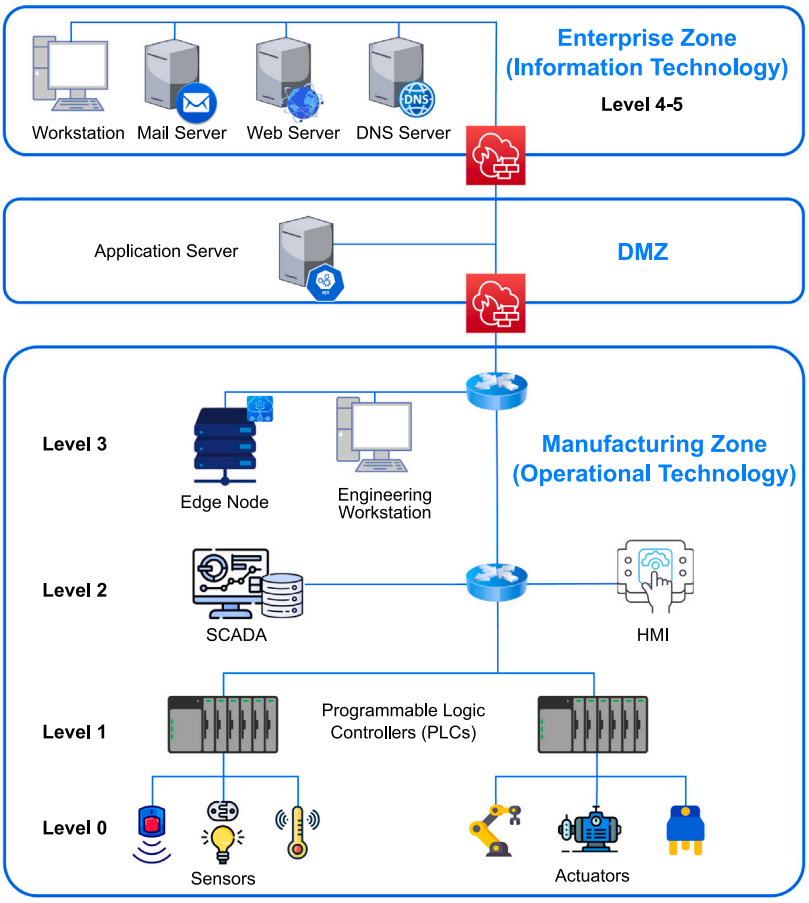



**Fig. 3.** ICS Purdue Reference Architecture.

two worlds isolated from each other. The DMZ serves as a controlled buffer zone, enabling secure data exchange and access management while maintaining distinct security protocols suited to each network's priorities: IT focuses on data integrity and confidentiality while OT prioritizes system availability and physical safety. Additionally, the DMZ architecture aids in regulatory compliance by establishing a clear boundary that can be audited according to industry standards [24]. The reliability of the OT network is paramount: failures cannot be acceptable due to the critical nature of the physical processes monitored and faults would imply shutting down the entire industry, leading to potential economic losses. Finally, it is important to highlight that the risk impact is also different considering that, in the IT network, the principal risk is the loss or unauthorized alteration of data. Instead, in the OT environment, a security breach can jeopardize both production and equipment, while in the worst case, can cause a loss of lives or environmental damage [22]. This alignment with cybersecurity best practices underscores the DMZ's vital role in safeguarding industrial infrastructure, making it an essential component of modern network design

### 4.1. Threats on legacy devices: the OPC UA protocol

ICS employs a wide range of protocols, depending on the specific objectives of each system. Real-time constraints and legacy hardware are two of the most important challenges that industrial protocols are specifically made to address. Legacy components in particular are one the main source of issues in modern ICS. Legacy devices usually lack proper security measures. At the same time, protecting and managing these devices becomes necessary because replacing them is often a complex and expensive process. These devices typically show limited flexibility and frequently lack the capability to be updated to meet modern standards. As a consequence, these devices may exhibit shortcomings such as the absence of mechanisms for firmware management. This often leads to issues derived from outdated software (SW) and firmware (FW) such as limited connectivity, insecurity of channel communication, unverified data integrity, uninsured data confidentiality, or lack of access control monitoring policies.

In this context, a new building model, OPC UA [25] has emerged as the de-facto standard for machine-to-machine communication because, compared to other common industrial features, it enables platform-independent and secure communication by design. It has become popular with the advent of Industry 4.0, a paradigm that aims to create new business logic and markets while opening the old OT industrial segment to the Internet, legacy devices included.

Adopting this protocol in an industrial environment enables the integration of heterogeneous hardware, which is instead a constraint brought by proprietary protocols (like Siemens S7). At the same time, the advanced security-by-design capabilities of the protocol reduce some of the typical security risks of ICS (such as lack of message authentication or encryption). Two communication strategies are possible, namely client–server and publisher–subscriber. Even though OPC UA does not strictly enforce the use of security mechanisms, both communication models allow messages to be signed to ensure authenticity and encrypted to add confidentiality. Actually, OPC UA messages can be exchanged in one of three Security Modes: *None* for unprotected communication, *Sign* for authenticated communication, *SignAndEncrypt* for authenticated and encrypted communication [26].

Despite the benefits of the protocol, supporting the security features of OPC UA by product vendors, libraries implementing the standard, and end-users remains challenging, preventing companies from adopting proper security mechanisms. Currently, roughly 14.6% of OPC UA

device vendors do not support security features at all, while 64.6% of them present issues or errors in the Trustlist management, enable Rogue Client, Rogue Server, and Man-in-the-middle attacks, and only 20.8% of them correctly implement the security features offered by the OPC UA protocol [27].

We argue that, despite its potential, the OPC UA architecture needs to be backed by additional mechanisms when the adoption of its security features is limited or nonexistent.

### 4.2. Use case: Remote maintenance of industrial plants

With the ever-growing need for operational efficiency and negligible downtime, remote maintenance is becoming a critical tool for industrial plant maintainers. In fact, the possibility of connecting with the industrial plant remotely (e.g., from the premises where the IT is located) enables quick ordinary reconfiguration and prompt reaction to unexpected events, at all times. In this section, we describe a typical remote maintenance use case and outline the risks that this work method involves.

In our use case scenario, a technician needs to operate on equipment situated in the industrial plant, but from an external location, through a remote connection. Such connection may carry text-based data (e.g., for command line operation on a terminal) or multimedia data (e.g., for visual control of machinery).

The technological requirements to establish this kind of remote session are mostly already met in modern industrial environments. However, their activation is often hard to automate, since the OT is generally managed in local networks, and it may include specialized hardware (e.g., an industrial gateway that performs security duties) [28]. Moreover, it is prone to security and privacy threats, arising from the exchange of potentially critical information outside of the private network of the company. Furthermore, in order to guarantee maximum efficacy, the Quality of Service (QoS) perceived by the user (i.e., the technician, in this example) should be high enough to support the workflow without impediments or delays.

The traffic generated by the remotely controlled equipment needs to cross multiple network segments. For instance, the data may be generated on a device connected to a server (e.g. OPC-UA) located on a PLC in the OT segment. From there, it needs to cross the OT network, which is composed of devices such as PLCs, SCADAs, and actuators, which typically have low computation capabilities. Then it goes through the IT network, which usually hosts most of the computation power. Finally, it crosses the Internet, which exposes it to a number of potential threats [29]. All things considered, the technology implied is not enough to grant confidentiality and integrity of the data, while of course availability depends on the switching and firewall configuration. More specifically, we can analyze the Confidentiality, Integrity, Availability (CIA) risks of each of the traversed domains. Based on computation capability, the crossed path can be divided into two portions, namely OT-IT (low capability route) and IT-Internet (high capability route). In the OT-IT portion, confidentiality is not threatened, as data travels within the private network of the company. Its integrity, however, can be undermined by the fact that legacy devices do not usually support integrity checks. On the other hand, in the IT-Internet portion, data is prone to confidentiality threats, since more personal or company data are shared in such portion as well as integrity issues that may cause data flow disruption or alteration. In both portions, data availability is threatened by out-of-domain connections that can lead to denial-of-service attacks.

Companies often leverage Virtual Private Network (VPN) technology to establish a private end-to-end connection between the client and the IT (which usually comprises a VPN server), ensuring Confidentiality and Integrity of the transmitted data. All things considered, we can still identify two main points that need to be addressed to achieve automated deployment of safe remote maintenance sessions in the industrial plant. The first is the lack of CIA in the OT-IT portion, due to the scarce computation capabilities of the involved devices. The second is the automatic deployment of remote service components to support the maintenance session on client request, keeping CIA requirements into account.

For the purpose of this work, we have defined and implemented instances of such support services, which we refer to as Maintenance Services (MSs), and their goal is to enhance the reliability of the remote maintenance routine, consistent with the use case presented in this section. This is achieved by applying hashing to the data flow, according to one of four different hashing algorithms, namely CRC32, XXH64, MD5, and SHA256. The data flow may be either text-based (e.g., for remote terminal operation) or video-based (e.g., for remote inspection), resulting in a total of eight new services offered to the user.

In Section 5, we provide a proof of concept to tackle the aforementioned concerns by leveraging service orchestration and PDPaaS through the activation of MSs.

## 5. Proof of concept: Emulation of an industrial environment

This section must be intended as a motivation chapter to introduce the evaluation on Section 6. In fact, the testbed used to draw the results shown in this section is emulated. Thus, the results must be interpreted as a simple prototype of our solution, which we believe can be reasonably scaled up to a real-world scenario.

Fig. 4 shows the emulated industrial environment used for the tests, which is inspired by the architecture described in Section 4.

The IT and the OT are separated by a DMZ: on the IT side, it is accessed via a traditional router, while the OT devices are reached through a P4 programmable switch. The P4 switch can be managed by the orchestration framework to provide specific PDPaaS services. Since the goal is to establish a remote maintenance connection, our system provides a remote VPN connection to the industrial network for a client located on the Internet. The VPN server is placed in the IT network, alongside the workstations and the orchestrator presented in Section 3. In particular, the orchestrator is responsible for deploying services both in the IT and the OT. The types of services that can be managed by the orchestrator are also described in Section 3.

When compared to the Purdue reference architecture, the OT network is simplified to only one level, in which two main elements are located, namely OPC UA servers and Edge Nodes. Edge Nodes supply computational power to the Manufacturing Zone and can support the deployment of PDPaaS services to provide integrity along the OT-IT path. OPC UA servers, instead, are legacy devices with low computing and networking capabilities. Considering this aspect, we assume that they can only operate in Security Mode None. In addition, OPC UA servers are connected to monitoring cameras, as described in [30], which observe the assembly line and allow the remote maintainer to discover faults in the production processes.

### 5.1. Experimental setup

We implemented the emulated industrial testbed using the Kathará framework [31]. Our Kathará topology and all the test scripts are open-source and public at [32]. All elements of the testbed are implemented as Docker containers, except for a separate node on IT premises hosting the Service Orchestrator. The remote client hosts an OPC UA client process capable of connecting to a VPN server located in the IT network. We built a specific Docker container image published to Docker Hub to implement these functionalities, employing the opcua-asyncio[2] Python library for the OPC UA client functionalities. Edge nodes and IT workstations are general-purpose Debian-based Docker containers; edge nodes have higher computing capabilities, in terms of RAM and CPU power, than the others. OPC UA servers, on the other hand, are

[2] https://github.com/FreeOpcUa/opcua-asyncio

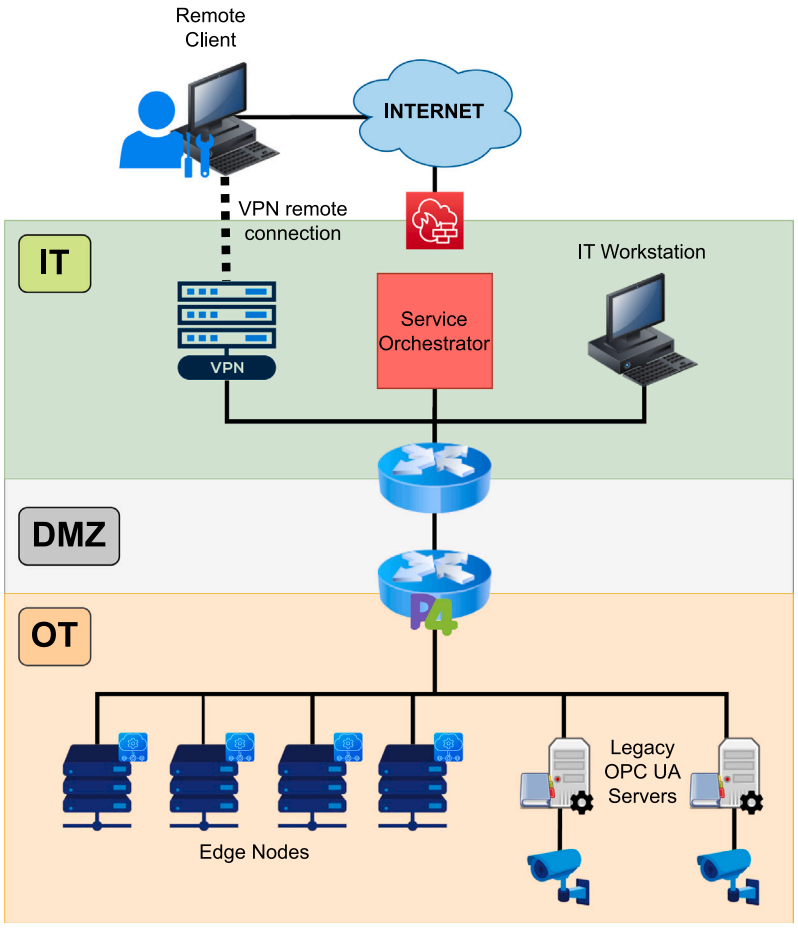



**Fig. 4.** Topology of the Proof-of-Concept implementation.

equipped with less RAM and less CPU power, to account for the fact that they represent legacy devices, and they are based on the same image as the ones that represent the remote client. The VPN server is an OpenVPN server that pushes the route to reach the OT private subnets to the remote client. Between the OT and the IT segments, there is a traditional Open vSwitch [33] which acts as a firewall by means of IPTables rules, with the goal of allowing only VPN traffic. A container emulating the P4 switch is placed between the IT nodes and the Open vSwitch connecting the OT and the IT networks. It is equipped with a simple level 2 forwarding pipeline[3] and configured to only forward traffic to the end hosts: the P4 code is compiled for the reference virtual target bmv2.[4] On top of this, we developed a P4 program for each payload size and hashing function, i.e., 256 and 1024 bytes, for a total of 8 pipelines (i.e., 2 different payload sizes and 4 hashes). To develop the hash functions, we exploited the idea of P4 extern.[5] An extern is an API that uses an external dependency, which can be queried by the target. Each hash function is implemented in the form of

$$hash(output, input[])$$

where the output is the hash produced by hashing the concatenation of the input payload chunks, each chunk being 256 bytes since bmv2 only allows for variables with a size up to 2048 bits. Each hash extern leverages standard C++ implementations of the hashing functions.

[3] https://github.com/UniboSecurityResearch/P4-Forch_KatharaTopo/blob/master/router1/root/p4/program.p4

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

[5] https://p4.org/p4-spec/docs/P4-16-working-spec.html#sec-external-units

Each pipeline hashes the payload in the ingress queue control, by simply calling the extern and adding the hashed payload in the custom field at the start of the payload, as summarized in Algorithm 1.

**Algorithm 1:** Hashing in the ingress pipeline

**Input** : a packet $packet$ containing
$H$ packet header, $hash$ payload hash custom field,
$P$ payload;
**Output:** $hashed_P$ = hashed $P$
1 $hash \leftarrow hash(hashed_P, P)$: we set the custom hash field in the packet, which is then carried in the network.

We tested the topology on a Ubuntu 20.04 LTS Server with 14 GB of RAM and 3 CPU cores KVM machine.

### 5.2. Hashing performance comparison

To evaluate our solution we compared the performance of the programmable nodes and traditional edge nodes in performing the CRC32, XXH64, MD5, and SHA256 hashing functions. To do so, we ran some tests calculating the effectiveness of the two different options to hash OPC UA packet payloads. The results obtained can then be used by the orchestrator when deciding on whether a switch or an edge node should be chosen as a resource to deploy the integrity hashing function on the plant.

As we can see from Fig. 5 , the payload processing time of the switch increases depending on the type of hash function.

We chose a simple metric to calculate the added hashing overhead, the Total Time Increment (TTI). Given the average processing time to process an OPC UA packet by a switch ($T_{baseline}$) over a number $N$ of

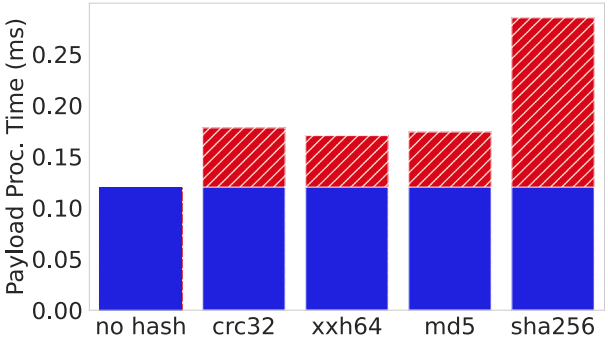



**Fig. 5.** Payload Processing Time for 1024 bytes payloads. In blue, is the time to process the OPC UA payload without having it, and in red is the added time to hash it with different functions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

analyzed packets with a processing time $T_i$

$$T_{baseline} = \frac{\sum_{i=1}^{N} T_i}{N}$$

and given the time to hash an OPC UA packet by a switch or an edge node ($T_{hash}$) over a number $N$ of analyzed packets with a processing time $H_i$

$$T_{hash} = \frac{\sum_{i=1}^{N} H_i}{N}$$

The Total Time Increment (TTI) is defined as the percentage increase of the hashing time with the baseline, calculated as:

$$TTI(\%) = \frac{T_{hash} - T_{baseline}}{T_{baseline}} \cdot 100$$

We calculated the TTI for three different payload sizes: 256 bytes (Fig. 6) and 1024 bytes (Fig. 7), for different traffic rates. We gathered data for traffic lower than 4 Mbps since in our configuration the topology has packet loss for traffic above that threshold. These limitations come from the use of a virtualized environment and bmv2.

Figs. 6 and 7 show the TTI for OPC UA payloads of 256 bytes, comparing switch and edge node hashing to the baseline. By looking at the graphs, we can summarize two main outcomes:

1. Computing hashes on the edge nodes entails a TTI that is 4 orders of magnitude bigger than hashing on the switch.
2. The TTI of the switch varies depending on the hashing function.

Statement 1. is explained by the fact that the processing time of the edge node is calculated between the ingress and egress interfaces of the container. In fact, this time is the composition of the time to move the packet sniffed with the Scapy[6] library from kernel to user space, the time to hash the payload with a Python script, and the time to move back the packet from user to kernel space. Moreover, the node TTI depends more on the traffic rate than the hashing function.

The switch TTI results roughly confirm the expected behavior in terms of hashing complexity [34]. In fact, the SHA256 hash calculation takes more steps than CRC32, XXH64, and MD5.

Fig. 8 groups the TTI curves for 1024 bytes payload, using a logarithmic TTI scale. Each set of curves is graphically grouped with a colored circle, based on the type of device that performs the hash. The logarithmic scale outlines the difference between the node TTIs and switch TTIs.

As an overall comparison, Fig. 9 sums up how the average processing time for a packet on the switch completely outperforms the respective one on the edge node. This suggests that, if choosing between an edge node and a switch for service placement purposes, the obvious choice is the programmable switch.

[6] https://scapy.net/

## 6. Evaluation process and results

As implied by the results presented in Section 5.2, offloading the computation required for MSs to PDP devices allows to achieve better performance compared to using conventional computing devices for the same task, while also requiring no replacement of legacy devices that are already deployed in the scenario. In this section, we evaluate the feasibility and the performance of the orchestration of those services in an environment such as the one described in Section 5. To do so, we implemented a Python-based discrete event simulator.

In the simulator, the orchestration system is actually reduced to its *Service intelligence* element, as no real telemetry data is collected from – and no management action is applied to – underlying resources. The intelligence of the simulated orchestrator is in charge of making placement decisions while enforcing a given policy, and, as the simulation progresses, the availability of resources is affected by the decisions made up to that point. For performance evaluation purposes, we implemented three policies, so as to have the orchestrator prioritize different aspects while making its decisions.

The first policy is denoted as *Random* (R), and instructs the orchestrator to make a completely random decision on the action to apply to any service request, including the choice of where and how to activate the service or block the service activation altogether. Such actions are uniformly distributed and independent of one another, meaning that each service request might be served by a computing resource, a networking resource (if available), or be blocked, with the same probability. This placement algorithm is represented in Algorithm 2.

**Algorithm 2:** Service placement with *Random* policy

**Input :** set $R^c$ of computing resources $R^c = \{r_i^c \mid i \in \mathbb{N} \wedge i \leq \text{amount of computing resources}\}$; set $R^n$ of networking resources, $R^n = \{r_j^n \mid j \in \mathbb{N} \wedge j \leq \text{amount of networking resources}\}$; combined set $R$ of available resources $R = \{r \in R^c \cup R^n \mid r \text{ is not completely busy}\}$

**Output:** service placement decision $p$

1 $p \leftarrow choice(\{r \in R\} \cup \{\text{block}\})$, where *choice* represents a random selection of an element from the input set, with uniform probability

The second policy is called *Load balancing* (LB), according to which the orchestrator is expected to balance the service placement between computing and networking resources, with a particular focus on network utilization. In other words, when enforcing this policy, the orchestrator tries to balance the overall load due to service placement across the resources. This favors the choice of a networking resource over a computing one when the network load grows, leveraging the better ability of networking resources in providing specific services. As detailed in Algorithm 3, when applying this policy, the orchestrator sorts the available resources based on the collected metrics (CPU, RAM, disk, network usage, etc.), from the least to the most loaded. It then picks the resource with the lowest network occupation between the first computing resource and the first networking resource in the sorted list.

The third and final policy considered is referred to as *User QoS* (UQoS) because it aims at optimizing all aspects perceived by users, including the service activation time and overall latency, as discussed in 5. The rationale behind this is in the consideration that activating a service on a PDP device with a pre-configured pipeline is much faster than doing it on a computing resource, as will be further argued later on in this section. Additionally, the performance in terms of service fruition of PDP devices have been shown to be superior to that of computing resources. For these reasons, if the objective is to enhance the overall user experience as much as possible, networking resources should be favored over computing ones at all times. In line with what is

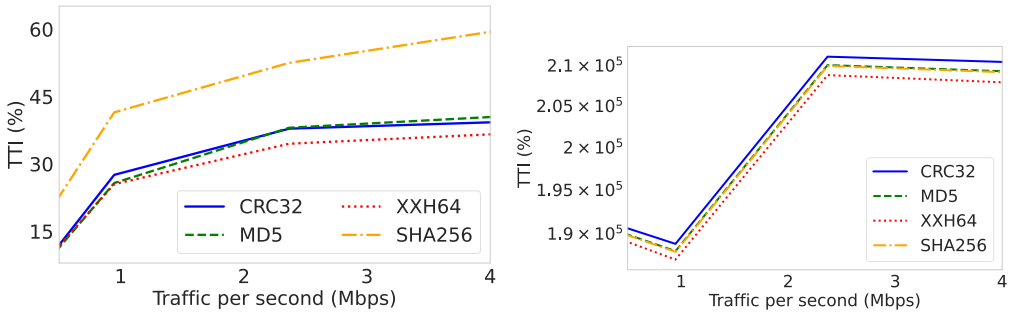



**Fig. 6.** Total Time Increment values for 256 *bytes payload* packets measured on a switch (a) or an edge node (b), for each hashing function (CRC32, XXH64, MD5, SHA256) with variable traffic volume.

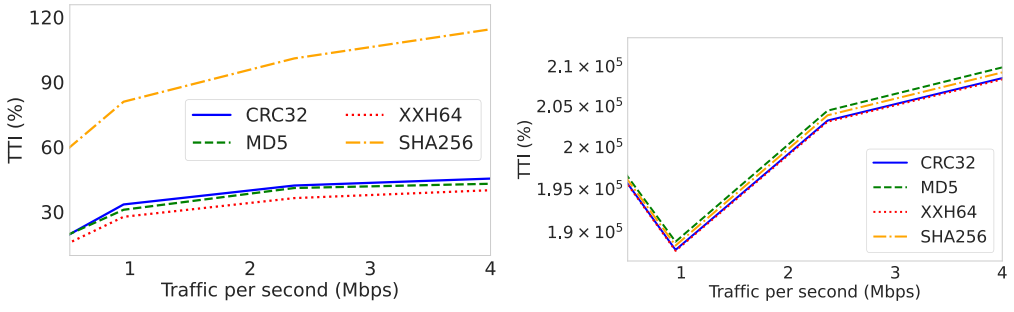



**Fig. 7.** Total Time Increment values for 1024 *bytes payload* packets measured on a switch (a) or an edge node (b), for each hashing function (CRC32, XXH64, MD5, SHA256) with variable traffic volume.

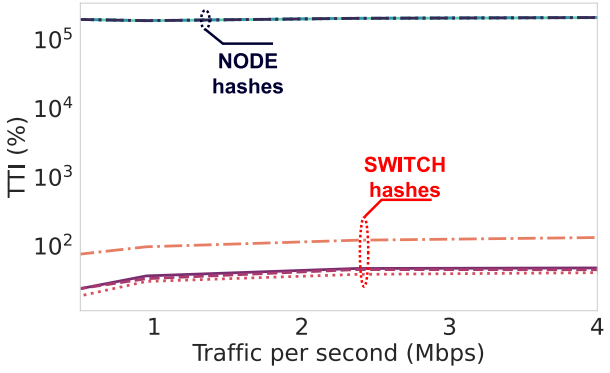



**Fig. 8.** TTI resulting from performing the hashing on the switch or on the edge node.

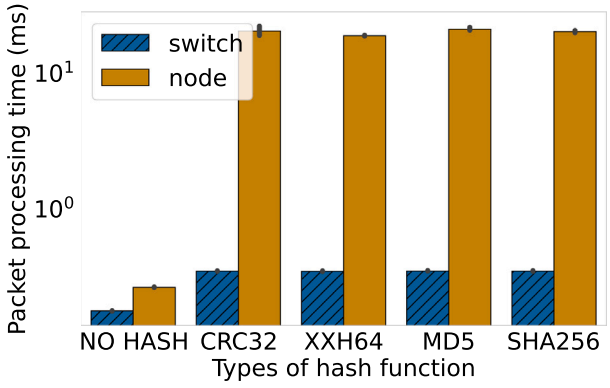



**Fig. 9.** Edge node and switch average delay comparison for different hashing functions.

represented in Algorithm 4, when activating a service, the orchestrator will always select the P4 switch if it is available, or the least occupied edge node otherwise, blocking the request if none of these resources is available.

In each simulation run, the orchestrator receives a sequence of service requests, which may require any of the supported services to be activated, including the newly-implemented MSs, that may leverage DPP. The service requests are generated following a Poisson process, with the service duration being exponentially distributed. Specifically, service requests are randomly generated out of a pool of 50 different services, of which 8 are MSs and 42 belong to the other service models with the following proportion: 10% IaaS, 20% PaaS, 30% SaaS, and 40% FaaS. Each request may pose different requirements in terms of computing power, memory, storage, and network capabilities, as well as in terms of service provisioning mechanisms (referred to the XaaS paradigm detailed in Section 3). Based on the policy to be enforced, the details specified in the request, and the current status of the underlying resources, the orchestrator needs to make a decision on how and on which resource to activate the service, or to block the request in case no suitable resource is available. In the simulation, no actual telemetry data is collected, so all resources are considered to be fully available at the beginning of each run, while at each simulation step they are assigned to an incoming service request, and released at its completion.

We estimate the service activation delay, defined as the additional time required by the orchestrator to activate the service (not including network time), by counting the amount of times that a MS was activated according to each of the different models supported, and multiply that amount by the average time required to activate a service in that way, as evaluated in [20].

The simulation scenario stems from the topology depicted in Fig. 4, instantiated in the simulator in three different configurations, with either 1, 4, or 9 compute nodes and a P4 switch available, for a total

**Algorithm 3:** Service placement with *Load Balancing* policy

**Input :** set $R$ of available resources $r$ as defined in Alg. 2; current status $s$ of each resource as a collection of metrics, $s(r) = \{\text{CPU, RAM, etc.}\}, \forall r \in R$;
**Output:** service placement decision $p$

```
if R = ∅ then
    p ← block
    return
if |R| = 1 then
    p ← the only available resource
    return
Create the ordered set R_O by sorting elements of R by their metrics, starting from the least busy one
if no networking resource in R_O then
    p ← the first element of R_O
    return
if network load of least busy networking resource < network load of least busy computing resource then
    p ← the least busy networking resource
else
    p ← the least busy computing resource
```

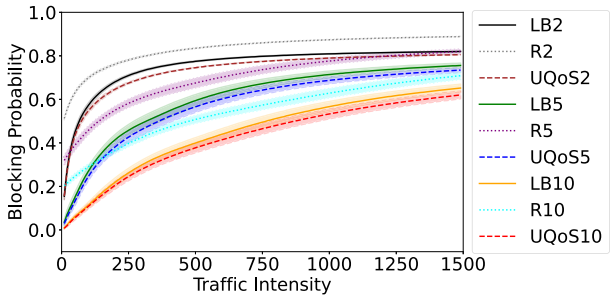
**Algorithm 4:** Service placement with *User QoS* policy

**Input :** set of available resources $R$ as defined in Alg. 2; current status $s$ of each resource as defined in Alg. 3
**Output:** service placement decision $p$

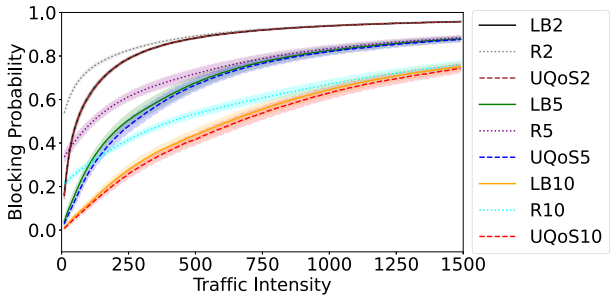
```
if R = ∅ then
    p ← block
    return
if |R| = 1 then
    p ← the only available resource
    return
Create the ordered set R_O by sorting elements of R by their metrics, starting from the least busy one
if ∃ networking resource ∈ R then
    p ← the least busy networking resource
    return
p ← the least busy resource
```

of 2, 5, or 10 available items in the resource pool the orchestrator can choose from. Industrial environments are very diverse, in that the amount and distribution of computing and networking resources can vary widely among different premises. In such environments, we expect to have a limited amount of computing resources in the IT and at most one programmable switch (along with potentially multiple non-programmable ones) between OT and IT. We argue that the different scenarios we showcased represent real-world industrial ones, both from the point of view of the technological solutions involved (i.e., P4) [22], as well as from that of the infrastructural network models [35]. Based on the same topology, we further distinguish two cases, differentiated by whether the P4 switch is available or not — in other words, whether the orchestrator can rely on heterogeneous (computing and networking) resources or not.

We simulated each different scenario given by the combination of configuration of nodes, switch availability, policy, and traffic intensity. In particular, available nodes were allowed to vary freely between simulations, but at least one of them was required to support the IaaS model. Moreover, we let the traffic intensity (the ratio between arrival rate and service rate) vary from 10 to 1500, in steps of 10. For each scenario, we run the simulation 20 times, with different seeds, to obtain




(a) PDP device available




(b) PDP device unavailable

**Fig. 10.** Probability of the orchestration system in blocking a generic service request due to insufficient available resources. The curves represent the performance of different service placement policies with different amounts of available resources, as indicated by acronyms and numbers in the legend.

different incoming service requests at each run. We generated and submitted $10^5$ service requests for each run. We then computed the sample mean over the 20 runs and reported them in the graphs as the solid and dashed lines, surrounded by shaded areas depicting the 95% confidence interval.

To begin with, we assessed the probability of a service request being blocked by the orchestration system. This was computed as the ratio between the number of blocked requests and the number of offered ones. As shown in Fig. 10, the results show an Erlang-B formula behavior, as can be expected, given the distributions of the service request arrival rate (Poisson) and of the service duration (exponential), as well as the fact that the orchestrator behaves like a multiple-server system without queuing space. We can also confirm the intuitive expectation that the blocking probability is higher for scenarios with a lower amount of available resources, and for those using the *Random* service placement policy. The difference between Figs. 10(a) and 10(b) is in the availability of the P4 switch. In the former case, the switch can be leveraged to provision services. In the latter case, the switch is assumed pre-loaded with other tasks, making it unavailable for the orchestrator. The policy that is most influenced by this difference is the *User QoS*, as it prioritizes the use of the PDP device. Also the *Load Balancing* policy shows some sensitivity to the availability of networking resources for a very small amount of overall resources, as in that case the policy can no longer reduce the load on the compute nodes.

As a further evaluation, we assessed the impact of the activation of MSs on the workflow of the user. In other words, we wanted to evaluate the delay caused by the additional deployment time of the service components needed to apply hashing to the text or video stream that the user employs to perform remote operations and compare the impact of different service policies. We defined a metric, called Mean Activation Time (MAT), that is computed at the end of the simulation of each scenario, as the ratio between the total time taken by the activation of MS instances over the number of such instances activated in the simulation run, where all kinds of service requests are generated.

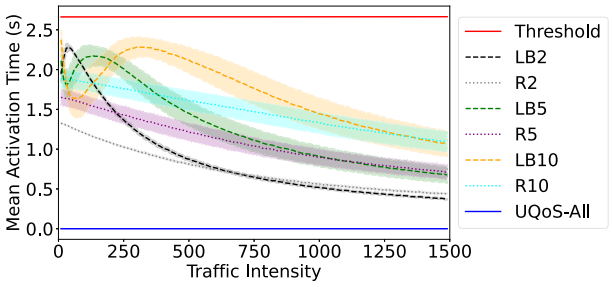
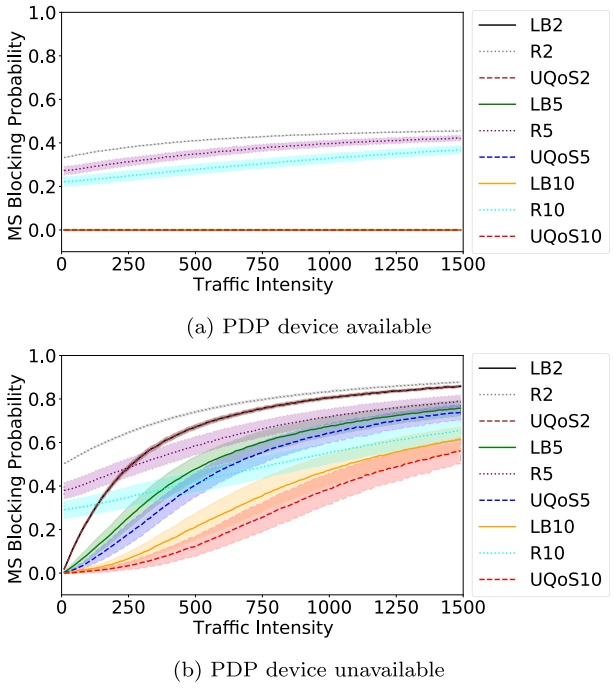



**Fig. 11.** Mean Activation Time of Management Services for different service placement policies and amounts of available resources.

In practical terms, if MAT is zero, the activation of services by the orchestrator did not introduce any tangible delay in the remote operations of the user. For instance, this is possible when the MS is deployed on a P4 switch pre-loaded with the code needed to perform the service, requiring only a signal (e.g., a REST HTTP request) to activate the additional traffic processing, without the need for a reconfiguration of the pipeline. To be precise, this would still inevitably require a small time, but we can reasonably approximate it to zero, as it would be seamless to the workflow of the user. Any value of MAT larger than zero means that the user is subjected to a small delay while the hashing mechanisms are activated.

This MAT is represented in Fig. 11, focusing on the case when a PDP device is available. Such switch is consistently leveraged by the policy *User QoS* for the deployment of services, allowing that policy to achieve a negligible MAT. The MAT related to the other policies depends on the traffic intensity. In particular, the decrement of the MAT with larger values of traffic intensity is due to the decreasing availability of edge nodes, forcing the two policies to choose between the PDP device or blocking the request. To fully understand this result, one must consider that the activation of services on compute nodes takes longer for the first MS instances deployed on them, due to the need to download the required software (e.g., the Docker image) and set it running. The activation time decreases as further requests for the same services are served, as the software components will already be on the node. When considering small values of traffic intensity, the impact of the first requests (i.e., the ones taking longer to be activated) will still be relevant with respect to the following ones (i.e., those served in a shorter time), leading to a higher value of MAT. In line with this rationale, one would expect all curves to exhibit a maximum for the lowest value of traffic intensity. However, that is not so for the *Load balancing* case, as this policy places services on the resource that is least involved in handling network traffic, generally favoring compute nodes in an initial phase, until the point where it can no longer keep up with the increasing traffic intensity as resources are saturated, resulting in choosing the PDP device. The small number of requested MSs in comparison to other generic services (on average 8 out of 50) are typically served by the PDP device. On the other hand, the *Random* policy exhibits a trend compliant with its working principle of randomly selecting a service placement action, and such actions are restricted to activating the service on the PDP device or blocking the request. The horizontal line denoted as "Threshold" represents the mean of the MAT value across all the policies when the PDP device is completely busy, thus unavailable to the service placement process. The fact that this line is never crossed is due to the availability of the PDP device, showing that, when the nature of the services allows it, leveraging DPP always leads to better performance compared to using compute nodes alone.

As a last evaluation, we assessed the blocking probability again, but limiting the requested services only to those that can be deployed on the programmable switch, i.e., to MSs. In Fig. 12(a) we can see that if the switch is available, the only policy experiencing blocking is the




(a) PDP device available

(b) PDP device unavailable

**Fig. 12.** Probability of the orchestration system in blocking a request for a Management Service due to insufficient available resources. The curves represent the performance of different service placement policies with different amount of available resources, as indicated by acronyms and numbers in the legend.

*Random* one, as the other policies always pick the switch to provide the service. This is the reason for the apparent better performance of the *Random* policy for low values of traffic intensity in Fig. 11: the policy is actually blocking numerous requests. Conversely, when the switch is not available, the situation is comparable to when the switch is available but overloaded, making Fig. 12(b) resemble Fig. 10(a).

## 7. Conclusion

In this work, we demonstrated how DPP can be employed to enhance service orchestration over a heterogeneous infrastructure, consisting of a diversified set of resources. We focused on an industrial environment, which is particularly prone to security threats, and suggested solutions to mitigate the issue. We considered a number of different methods to improve data integrity, implementing services employing them, and making them available to a service orchestration system that is able to activate them at need over heterogeneous resources. The evaluation consisted of two parts, the former focused on the data integrity mechanisms, and the latter on the service orchestration solution.

We demonstrated how the use of PDP devices can be largely beneficial to the execution of data integrity mechanisms compared to regular container-based services on the edge nodes. In fact, the programmable switches completely outperform the edge nodes in terms of processing time since the integrity calculation is placed inside the network nodes and performed on the flowing traffic. We evaluated the orchestration solution with the use case of remote maintenance in mind, addressing various technical, policy, and performance aspects. We focused on the offloading of computation for MS to PDP devices, showing that this approach offers superior performance compared to conventional computing devices without requiring the replacement of legacy equipment. We considered multiple distinct service placement policies that allow the orchestration systems to make intelligent decisions in line with their objectives and resource availability. We aimed at replicating real-world conditions, where MS orchestration must handle a dynamic sequence

of service requests with varying requirements. We measured the likelihood of service requests being blocked due to resource constraints, as well as the delay introduced by MS activation, showing that the presence and availability of PDP devices may have a significant impact on the achieved user QoS.

In conclusion, this research contributes to a deeper understanding of how to optimize the orchestration of management services in industrial settings. By addressing technical, policy-driven, and resource-specific aspects, our study provides guidance for achieving better resource utilization, reduced service activation delays, and improved user experiences.

## CRediT authorship contribution statement

**Gaetano Francesco Pittalà:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft. **Lorenzo Rinieri:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Writing – original draft, Visualization. **Amir Al Sadi:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft. **Gianluca Davoli:** Conceptualization, Investigation, Methodology, Project administration, Supervision, Writing – original draft, Writing – review & editing. **Andrea Melis:** Conceptualization, Project administration, Supervision, Writing – original draft, Writing – review & editing. **Marco Prandini:** Conceptualization, Project administration, Supervision, Writing – review & editing. **Walter Cerroni:** Conceptualization, Project administration, Supervision, Writing – review & editing.

## Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Gaetano Francesco Pittalà reports financial support was provided by European Union. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

Data will be made available on request.

## Acknowledgments

This work was partially supported by the European Union under the Italian MUR National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnerships on “Telecommunications of the Future” (PE00000001 - program “RESTART”) and on “SEcurity and RIghts In the CyberSpace” (PE00000014 - program “SERICS”).

## References

[1] M. Caprolu, R. Di Pietro, F. Lombardi, S. Raponi, Edge computing perspectives: Architectures, technologies, and open security issues, in: 2019 IEEE International Conference on Edge Computing, EDGE, IEEE, 2019, pp. 116–123.

[2] W. Yu, F. Liang, X. He, W.G. Hatcher, C. Lu, J. Lin, X. Yang, A survey on the edge computing for the internet of things, IEEE Access 6 (2018) 6900–6919, http://dx.doi.org/10.1109/ACCESS.2017.2778504.

[3] E. Schiller, A. Aidoo, J. Fuhrer, J. Stahl, M. Ziörjen, B. Stiller, Landscape of IoT security, Comp. Sci. Rev. 44 (2022) 100467.

[4] E.F. Kfoury, J. Crichigno, E. Bou-Harb, An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends, IEEE Access 9 (2021) 87094–87155.

[5] I. Sittón-Candanedo, R.S. Alonso, J.M. Corchado, S. Rodríguez-González, R. Casado-Vara, A review of edge computing reference architectures and a new global edge proposal, Future Gener. Comput. Syst. 99 (2019) 278–294.

[6] G.C. Sankaran, K.M. Sivalingam, H. Gondaliya, P4 and netfpga based secure in-network computing architecture for ai-enabled industrial internet of things, IEEE Internet Things J. (2021).

[7] J. Vestin, A. Kassler, S. Laki, G. Pongrácz, Toward in-network event detection and filtering for publish/subscribe communication using programmable data planes, IEEE Trans. Netw. Serv. Manag. 18 (1) (2020) 415–428.

[8] M. Uddin, S. Mukherjee, H. Chang, T. Lakshman, SDN-based multi-protocol edge switching for IoT service automation, IEEE J. Sel. Areas Commun. 36 (12) (2018) 2775–2786.

[9] C. Györgyi, P. Vörös, K. Kecskeméti, G. Szabó, S. Laki, Adaptive network traffic reduction on the fly with programmable data planes, IEEE Access 11 (2023) 24935–24944.

[10] P. Engelhard, A. Zachlod, J. Schulz-Zander, S. Du, Toward scalable and virtualized massive wireless sensor networks, in: 2019 International Conference on Networked Systems, NetSys, IEEE, 2019, pp. 1–6.

[11] T. Tachibana, K. Sawada, H. Fujii, R. Maruyama, T. Yamada, M. Fujii, T. Fukuda, Open multi-access network platform with dynamic task offloading and intelligent resource monitoring, IEEE Commun. Mag. 60 (8) (2022) 52–58.

[12] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, N. Li, In-band network telemetry: A survey, Comput. Netw. 186 (2021) 107763.

[13] OpenFog Consortium Architecture Working Group, OpenFog reference architecture for fog computing, 2017, p. 162, OPFRA001 20817.

[14] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: The communication perspective, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2322–2358.

[15] D. Santoro, et al., Foggy: A platform for workload orchestration in a fog computing environment, in: 2017 IEEE International Conference on Cloud Computing Technology and Science, CloudCom, IEEE, 2017, pp. 231–234.

[16] M.S. De Brito, et al., A service orchestration architecture for fog-enabled infrastructures, in: 2017 Second International Conference on Fog and Mobile Edge Computing, FMEC, IEEE, 2017, pp. 127–132.

[17] K. Velasquez, et al., Service orchestration in fog environments, in: 2017 IEEE 5th International Conference on Future Internet of Things and Cloud, FiCloud, 2017, pp. 329–336, http://dx.doi.org/10.1109/FiCloud.2017.49.

[18] M. Alam, J. Rufino, J. Ferreira, S.H. Ahmed, N. Shah, Y. Chen, Orchestration of microservices for IoT using docker and edge computing, IEEE Commun. Mag. 56 (9) (2018) 118–123, http://dx.doi.org/10.1109/MCOM.2018.1701233.

[19] G. Davoli, et al., A fog computing orchestrator architecture with service model awareness, IEEE Trans. Netw. Serv. Manag. 19 (3) (2021) 2131–2147.

[20] G.F. Pittalà, G. Davoli, D. Borsatti, W. Cerroni, C. Raffaelli, Function-as-a-service orchestration in fog computing environments, in: 2022 18th International Conference on Network and Service Management, CNSM, IEEE, 2022, pp. 364–366.

[21] N. Di Cicco, G.F. Pittalà, G. Davoli, D. Borsatti, W. Cerroni, C. Raffaelli, M. Tornatore, DRL-FORCH: A scalable deep reinforcement learning-based fog computing orchestrator, in: 2023 IEEE 9th International Conference on Network Softwarization, NetSoft, IEEE, 2023, pp. 125–133.

[22] M. Conti, D. Donadel, F. Turrin, A survey on industrial control system testbeds and datasets for security research, IEEE Commun. Surv. Tutor. 23 (4) (2021) 2248–2294.

[23] T.J. Williams, The Purdue enterprise reference architecture, Comput. Ind. 24 (2–3) (1994) 141–158.

[24] ISO/IEC 62443-3-3: Industrial communication networks - network and system security - part 3-3: System security requirements and security levels, 2013, International Organization for Standardization, Available from ISO, https://www.iso.org/standard/65696.html.

[25] OPC Foundation, OPC unified architecture specification part 1: Overview and concepts, 2022, URL https://reference.opcfoundation.org/Core/Part1/v105/docs/.

[26] OPC Foundation, OPC unified architecture specification part 2: Security, 2022, URL https://reference.opcfoundation.org/Core/Part2/v105/docs/.

[27] A. Erba, A. Müller, N.O. Tippenhauer, Security analysis of vendor implementations of the OPC UA protocol for industrial control systems, in: Proceedings of the 4th Workshop on CPS & IoT Security and Privacy, 2022, pp. 1–13.

[28] T. Honda, T. Hamaguchi, Y. Hashimoto, OPC UA information transfer via unidirectional data diode for ICS cyber security, in: Computer Aided Chemical Engineering, vol. 49, Elsevier, 2022, pp. 1459–1464.

[29] M.R. Asghar, Q. Hu, S. Zeadally, Cybersecurity in industrial control systems: Issues, technologies, and challenges, Comput. Netw. 165 (2019) 106946.

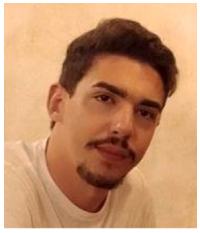
[30] OPC Foundation, OPC 40100-1: Machine vision - control, configuration management, recipe management, result management, 2019, URL https://reference.opcfoundation.org/MachineVision/v100/docs/.

[31] G. Bonofiglio, V. Iovinella, G. Lospoto, G. Di Battista, Kathará: A container-based framework for implementing network function virtualization and software defined networks, in: NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2018, pp. 1–9.

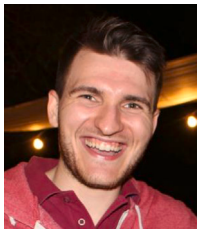
[32] L. Rinieri, A. Al Sadi, P4-forch_KatharaTopo, 2023, http://dx.doi.org/10.5281/zenodo.8376994, URL https://github.com/UniboSecurityResearch/P4-Forch_KatharaTopo.

[33] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, et al., The design and implementation of open {vSwitch}, in: 12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, 2015, pp. 117–130.

[34] S. Richter, V. Alvarez, J. Dittrich, A seven-dimensional analysis of hashing methods and its implications on query processing, PVLDB 9 (3) (2015) 96–107.
[35] C. Dong, X. Xiong, Q. Xue, Z. Zhang, K. Niu, P. Zhang, A survey on the network models applied in the industrial network optimization, Sci. China Inf. Sci. 67 (2) (2024) 121301.



**Gaetano Francesco Pittalà** received the master's degree in Telecommunications Engineering from the Alma Mater Studiorum—University of Bologna, Bologna, Italy, in 2022, where he is currently pursuing the Ph.D. degree in Electronics, Telecommunications, and Information Technologies Engineering. His research interests focus on Service Orchestration, Edge Computing and Serverless Computing.



**Lorenzo Rinieri** received the master's degree in computer engineering from the Alma Mater Studiorum—Università degli Studi di Bologna, Bologna, Italy, in 2022, where he is currently pursuing the Ph.D. degree in computer science and engineering. His research interests focus on industrial control system security, network security and SDN.

**Amir Al Sadi** (Student Member, IEEE) Amir Al Sadi is a Ph.D. candidate in Computer Science and Engineering at Alma Mater Studiorum Università degli Studi di Bologna, Bologna, Italy. He received a Master's Degree in Computer Science at the same University in 2021. His research interests focus on Programmable Data Planes, P4 and Network Security.

**Gianluca Davoli** is a Researcher and Assistant Professor at the University of Bologna, Italy, and his research interests revolve around communication networks, focusing on the new approaches to programmability, management, and monitoring of software-based network infrastructures.

**Andrea Melis** is a Junior Assistant Professor with the Department of Computer Science and Engineering, University of Bologna. His research focuses on aspects of computer security related to innovative software architectures. In particular, his actual research activity aims to study, design, and implement innovative solutions that allow to improve the safety and operational robustness of connected production industrial network and devices for cybersecurity contexts.

**Marco Prandini** is an associate professor at the University of Bologna, Italy, where he received his Ph.D. degree in electronic and computer engineering in 2000, and adjunct associate professor of cybersecurity at the University of Southern Denmark. His research activities started in the field of public-key infrastructures and later moved to subjects related to the security of microservice-based architectures, software-defined networks, IoT, and industrial control systems.

**Walter Cerroni** [M'01, SM'16] (walter.cerroni@unibo.it) is an associate professor of communication networks at the University of Bologna, Italy. Previously, he was a junior researcher at Alcatel, Dallas, Texas, a research associate at the National Inter-University Consortium for Telecommunications (CNIT), Italy, an assistant professor at the University of Bologna, Italy, and a visiting assistant professor at the School of Information Sciences, University of Pittsburgh, Pennsylvania. His recent research interests include software-defined networking, network function virtualization, edge/fog/cloud computing, service function chaining, intent-based networks, and service management. He has coauthored more than 140 articles published in the most renowned international journals, magazines, and conference proceedings. He serves as Lead Series Editor for the IEEE Communications Magazine Series on Network Softwarization and Management. He served as Associate Editor for the IEEE Communications Letters and as Technical Program Co-Chair for IEEE-sponsored international workshops and conferences.