Design and Implementation of a Slice as a Service Architecture on the Edge Cloud with Resource Constraints

Rodrigo Ferraz Azevedo
Department of Computer Science
Federal University of São Carlos
Sorocaba, Brazil
Email: eu@rodrigoazevedo.com

Luciano Bernardes de Paula
Federal Institute of São Paulo
Bragança Paulista, Brazil
Email: lbernardes@ifsp.edu.br

Fábio Luciano Verdi
Department of Computer Science
Federal University of São Carlos
Sorocaba, Brazil
Email: verdi@ufscar.br

***Abstract*—The Cloud Network Slicing (CNS) is a new concept that describes a mechanism to provide computing, networking, and storage as a virtual slice entity, enabling new approaches to IoT applications and structuring resources at the edge of the network. In this paper, the architecture defined in the NECOS Project is adopted and the functions for creating CNS in resource-constrained edge devices were designed and implemented. The implementation was evaluated on Single Board Computers (SBCs), using lightweight virtualization solutions (microservices) and the results achieved show that it is possible to instantiate CNSs on those hardware, however also show some limitations of multiple slice support on resource-constrained devices.**

***Keywords*-cloud network slicing; microservices; single board computers; cloud computing; edge computing; IoT.**

I. Introduction

Slicing is not a new concept and has been mentioned in several initiatives such as [1] and [4]. However, the emergence of a new concept, called Cloud Network Slicing (CNS), as defined in [2], involves not only the concept of slicing resources of network, but also computing and storage. This concept also allowed a new proposal for the IoT architecture, based on the isolated end-to-end (E2E) network abstraction, in which an IoT device becomes an element of the distributed network that hosts the CNS.

Additionally, in the context of CNS, it is not recommended that edge devices be dedicated to a single slice, especially in cases of lack of computational power, which implies, for example, in the absence of resources such as GPUs, which are specially used in image processing applications. Thus, it is interesting that a single device hosts more than one slice in specific cases [10].

One of the recent architectures that meet all these requirements is the one defined in the context of the NECOS Project (Novel Enablers for Cloud Slicing) [2]. Thus, the possibility of CNS instantiation in resource-constrained edge devices, through NECOS components, is one of the approaches that can make IoT services more secure, scalable, and reliable with the efficient use of scarce resources.

In NECOS, an architecture for CNS was designed and a minimalist implementation was developed in the context of the project. However, the implementation did not consider scenarios with low-resource equipment on the edge. The slices created essentially contemplate the use of computational devices with high processing power, typically physical servers located in data centers. Since the architecture was not instantiated in this type of scenario, its application in this project required the adaptation of some functionalities for scenarios focused on IoT.

The main purpose of this paper is to describe the design and implementation of a minimalist system for creating CNS in resource-constrained edge environments through NECOS components. It is shown how the components of the NECOS architecture were extended through microservices based on containers to support the resource-constrained hardware, whose implementation was validated through a proof of concept using a data collection service as the application, which can be used as an example of an environmental monitoring system.

Single Board Computers (SBCs) were used, which refer to computers that aggregate all electronic components on a single printed circuit board. The performance evaluation of the SBCs during the creation of one or more slices was performed using monitoring software to capture the CPU and memory consumption and temperature reached during this process. Although the results achieved show the possibility of supporting multiple slices in these resource-constrained devices, some limitations were found.

The remainder of this article is organized as follows. Section II details the implementation, the experiments performed and the results obtained. Finally, the conclusions and suggestions for future work are presented in Section III.

II. Implementation and evaluation

In this section it is described the extensions and evaluation of components of the NECOS architecture for creating CNS in resource-constrained network edge environments. A proof of concept was implemented to validate the new features using real hardware.

As NECOS depends on virtualization to support slice and SBC are incompatible with hypervisors, it was proposed in this paper to adapt the NECOS architecture to support multiple slices by changing the form of virtualization and reorganization of some network edge components.

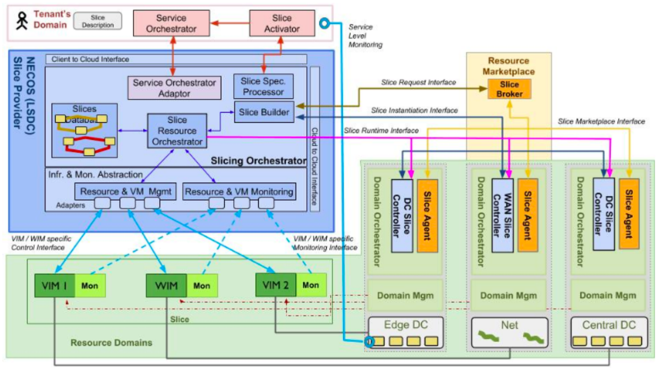


Figure 1: NECOS architecture.

The components were implemented following the NECOS architectural specification highlighted in Figure 1. However, the components of the Resource Provider (RP) subsystem needed to be extended to interact with the containers located in the SBCs, namely the DC Slice Controller (DCSC) and WAN Slice Controller (WSC).

The DC Slice Controller runs inside the provider and creates DC slices with computing and storage resources. The DCSC is also responsbile for instantiating a VIM (Virtual Infrastructure Manager) complying with the tenant's specification. The WAN Slice Controller is equivalent to the DCSC, but for the network part of the slice. The WSC runs inside a network provider and is responsible for interconnecting two DC slices. For more details about the NECOS architecture, see [2].

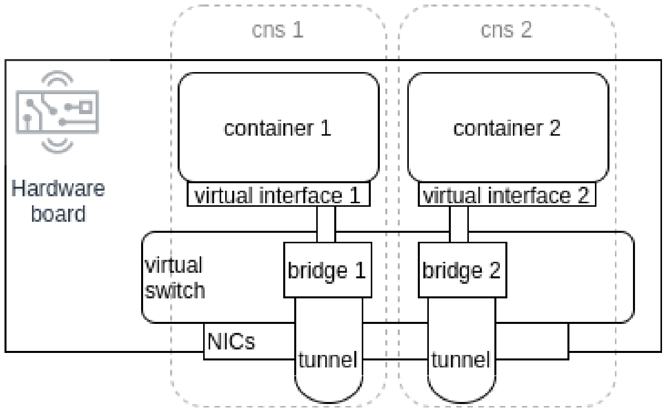


Figure 2: Software architecture at SBC.

The necessary extension occurred in the DCSC and WSC component algorithms to support the instantiation of containers directly in SBCs using the architecture highlighted in Figure 2. The DCSC and WSC were also moved to a new equipment as described below.

Although NECOS allows the entire edge structure (RP) to be instantiated virtually and on the same hardware as the final system, in the IoT context this scenario is not recommended, as such a situation would consume resources that are already scarce by nature. Thus, in this work, a new hardware layer was considered, in the form of a gateway, to receive the NECOS management components at the edge. Such approach is common for IoT scenarios and used in similar works [10]. Typically, as this is a low-processing hardware, deploying the architecture artifacts in the device would use the resources even more, which is not recommended given the hardware restrictions. In these cases, it is recommended to use a gateway, inside the edge, that acts as an intermediary node.

The SBCs chosen for the experiments were: Dragonboard 410c (ARM processor with 1.2 GHz, memory 1 GB and GPU), Raspberry Pi 3 (ARM processor with 1.2 GHz, memory 1 GB and GPU), Raspberry Pi 4 (processor 1.5 GHz ARM, 4 GB memory and GPU) and NVIDIA Jetson Nano (1.43 GHz ARM processor, 4 GB memory and GPU). These devices were chosen because they are widely used in the prototyping of IoT devices [5].

In this sense, the gateway received the three components described below, which interact with the SBCs via SSH.

- DCSC: responsible for isolating the network of each CNS through virtual network interfaces on the device and instantiating Docker containers associated with the appropriate CNSs;
- WSC: responsible for connecting the distributed slice parts, either in edge or cloud, through GRE tunneling ;
- Slice Agent (SA): responsible for updating the Slice Broker with the list of available physical devices to compose the CNS.

However, given that some edge computing hardware tend to have high acquisition and maintenance costs, it is also expected that they will support multiple slices. Consequently, together with the difficulty in offering minimal computational resources to support hypervisor, there was also a need to replace typical NECOS virtualization solutions. In this sense, two approaches to lightweight virtualization were considered in this project: containers [3] and unikernel [6].

However, limitations were encountered during the implementation and deployment of unikernel based solutions on the selected devices. This solution was not compatible with ARM devices, not allowing direct compilation on these, in addition to the dependence on hypervisor, making its orchestration unfeasible. The MirageOS, OSv and Unikraft projects were tested, without success. Therefore, such technology should be analyzed in the future, when solutions based on unikernel are more mature and applicable to this hardware.

To provide connectivity between the SBCs, the WSC was adapted to connect Docker instances in the edge with the slice parts through virtual tunneling using the GRE protocol and, for this functionality of tunneling, the OpenVSwitch

(OVS) [9] was installed in each of the SBCs to manage the tunnels.

The performance during the creation of one or more CNSs was captured using the Netdata monitoring tool [7], obtaining the CPU, memory consumption, and temperature achieved during this process. The results obtained confirmed the hypothesis of supporting multiple slices in these resource-constrained devices, with some limitations.

The results were divided in three aspects for analysis, as follows:

- Network: refers to the time in which the DCSC takes to slice the local network into subnets to isolate the CNSs;
- Container: refers to the time in which the DCSC takes to instantiate the CNS containers in the SBCs;
- WAN: refers to the time the WSC takes to connect the slice part to the CNS.

The benchmarks were performed by instantiating the CNS and then deleting it after its creation. The results are shown in Figures 3, 4 and 5. Values are related to CPU, memory consumption and temperature of operating system processes for managing Docker containers (Network and Container) and OpenvSwitch [9] (WAN).

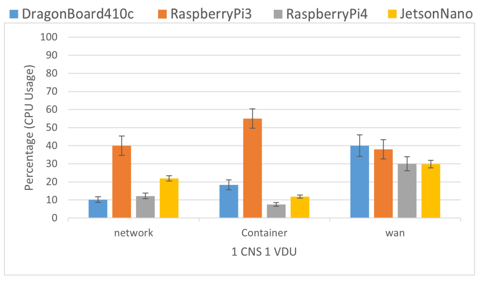
In a first evaluation, the average CPU consumption in different equipments was analyzed. For this, the amount of CNSs and VDUs in each slice was varied as follows: 1 CNS with 1 VDU (Figure 3a), 1 CNS with 2 VDUs (Figure 3b), 1 CNS with 3 VDUs (Figure 3c), 2 CNSs with 1 VDU (Figure 3d), 2 CNSs with 2 VDUs (Figure 3e) and finally 2 CNSs with 3 VDUs (Figure 3f) in each SBC. The process was repeated 5 times for each scenario.

The SBCs with less resources, i.e., Raspberry Pi 3 and Dragonboard, did not support a higher number of repetitions in a short period of time which resulted in loss of communication with the RP (Resource Provider) after the CPU usage was drained in the devices.

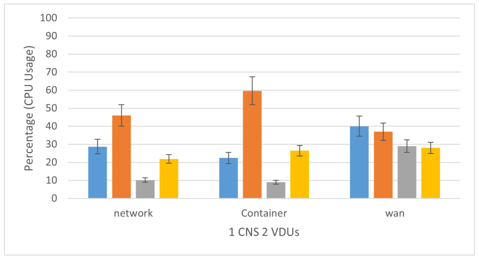
The maximum number of 2 CNSs with 3 VDUs was achieved in the Dragonboard and Raspberry Pi 3 devices. Those hardwares were not able to support more CNSs with more VDUs, something noticeable in Figures 3c and 3f where it is possible to observe the high CPU consumption.

On one hand, this behavior was already expected in hardwares with few resources for managing virtual environments. On the other hand, it is possible to observe that the Raspberry Pi 4 consumed less CPU in practically all scenarios, something that can be justified by its 1.5 GHz frequency, well above the 1.2 GHz of the two more restrictive SBCs and 1.43 GHz of the Jetson Nano. It was also found that the Raspberry Pi 3 had higher CPU consumption, although Dragonboard increased its consumption considerably as new VDUs were added.

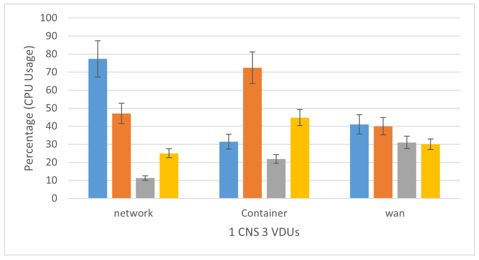
Regarding the memory consumption, shown in Figure 4, it is possible to observe that Dragonboard consumed more memory when instantiating the CNSs, but even so



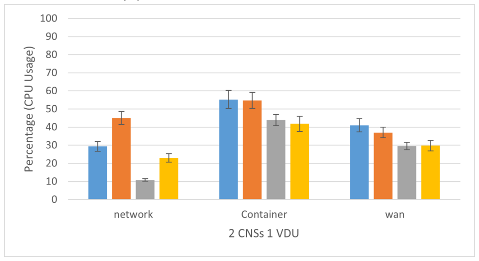
(a) 1 CNS with 1 VDU



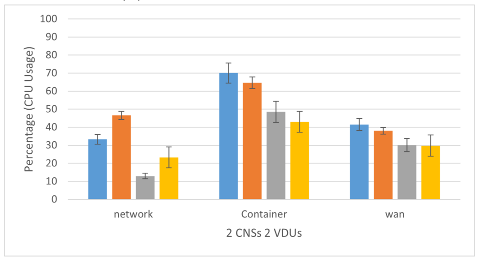
(b) 1 CNS with 2 VDUs



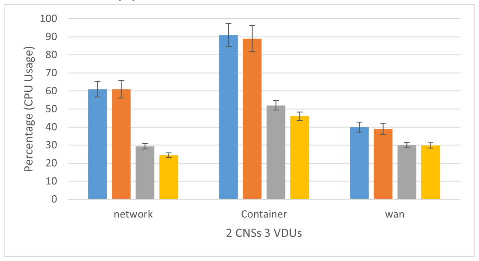
(c) 1 CNS with 3 VDUs



(d) 2 CNSs with 1 VDU



(e) 2 CNSs with 2 VDUs



(f) 2 CNSs with 3 VDUs

Figure 3: Average CPU usage during the creation of one and two CNSs varying the number of VDUs per CNS.

the consumption remained low compared to the amount of memory available in each SBC. The Raspberry Pi 4 was the SBC that consumed the least memory. The Jetson Nano used the resource comparatively above the expected if compared to its amount of memory available with other SBCs.

Regarding temperature, it is known that this characteristic is a limiting factor for this type of hardware due to the use of sensitive components. Both, the Rasberry Pi and the Dragonboard quote in their documentation that up to 70 degrees Celsius is the ideal operating temperature. The Jetson Nano presents in its official documentation the limit of 80 degrees Celsius for perfect operation [8]. Then, as shown in Figure 5, it is possible to observe that these limits were practically reached, limiting the number of CNSs instantiated and operating in these devices.

The average time of CNS instantiation in each SBC was also analyzed as shown in Figure 6. It is possible to notice the fast response of Jetson Nano, which justifies its higher CPU consumption when compared to the Raspberry Pi 4. Thus, the Jetson Nano is a good option for use in contexts that require instantiation or fast retrieval of virtualized elements at the edge of the network. The inferior hardware of the other SBCs reflected negatively on their times.

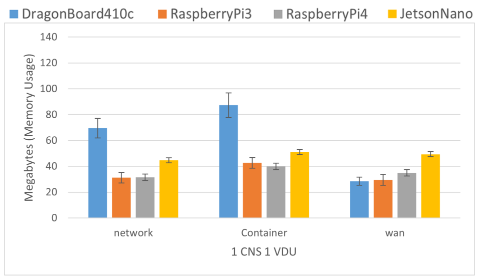
Jetson Nano was the SBC that performed best when instantiating each CNS in less time, followed by Raspberry Pi 4, Raspberry Pi 3, and lastly Dragonboard, which may justify its non-standard memory usage.

From the data presented, it is possible to confirm the support of these SBCs to the CNS. However, the limited hardware does not allow for the traditional NECOS approach that foresees devices without such restrictions and with constant connection, as in data centers. It is also verified that the minimalist structure of this hardware makes them susceptible to factors already controlled in other environments, such as the impact of ambient temperature and device operation in the applications it supports.

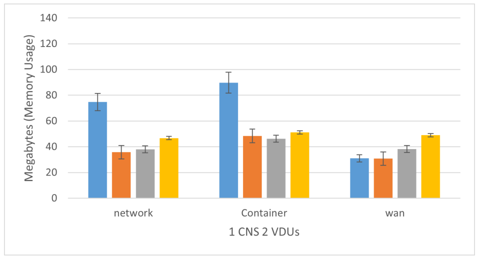
III. Conclusion and Future Work

In this paper, a proposal for a minimalist system using the components of the NECOS project for resource slicing and orchestration of IoT devices operating at the edge was presented. NECOS components were implemented and evaluated to support CNS creation in resource-constrained edge environments using lightweight virtualization through containers. A proof of concept was implemented to validate the new features using real hardware.

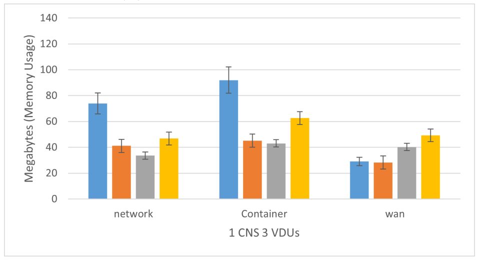
The results obtained confirmed the hypothesis of supporting slices in these devices, however there were limitations in the number of slices instantiated in the same hardware due to resource constraints in IoT devices. Therefore, the presented solution can meet solutions that demand exclusive hardware, but in a limited way in cases where the hardware needs to be shared among tenants.



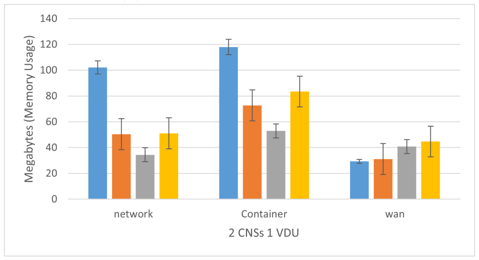
(a) 1 CNS with 1 VDU



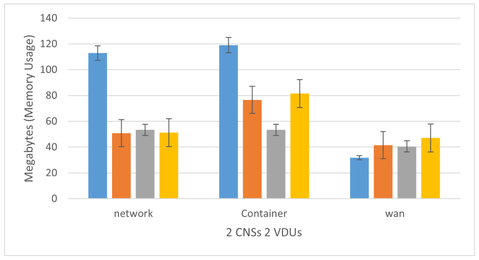
(b) 1 CNS with 2 VDUs



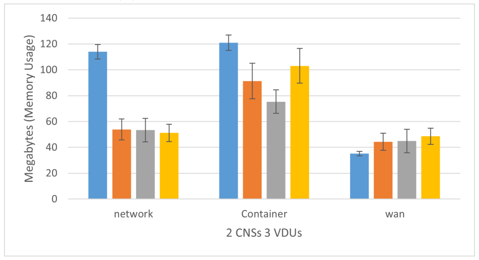
(c) 1 CNS with 3 VDUs



(d) 2 CNSs with 1 VDU

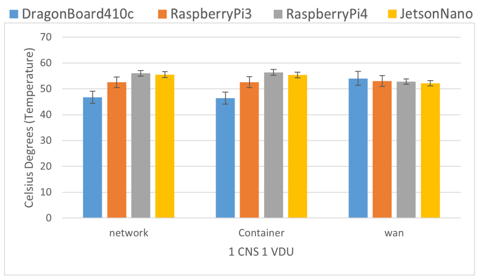


(e) 2 CNSs with 2 VDUs

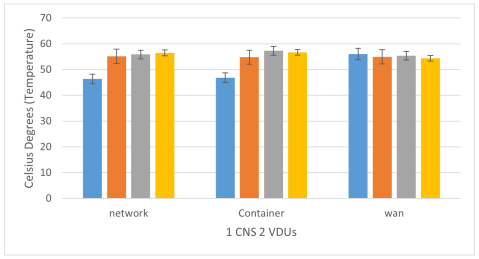


(f) 2 CNSs with 3 VDUs

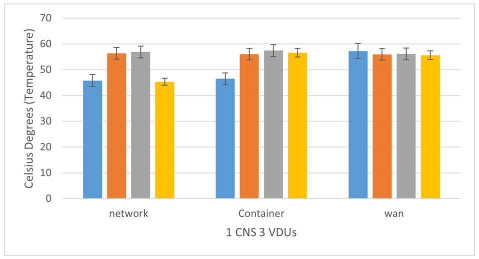
Figure 4: Comparison of average memory consumption when creating 1 and 2 CNSs with up to 3 VDUs per CNS.



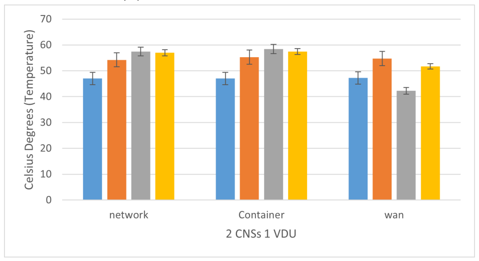
(a) 1 CNS with 1 VDU



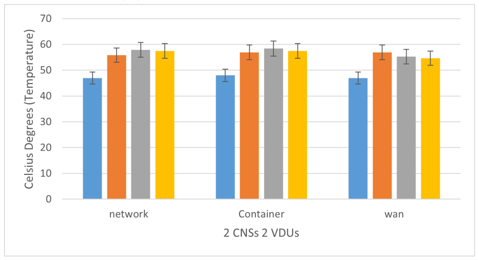
(b) 1 CNS with 2 VDUs



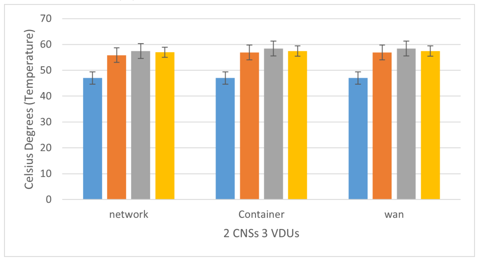
(c) 1 CNS with 3 VDUs



(d) 2 CNSs with 1 VDU



(e) 2 CNSs with 2 VDUs



(f) 2 CNSs with 3 VDUs

Figure 5: Comparison of the temperature reached during the creation of 1 and 2 CNSs with up to 3 VDUs per CNS.

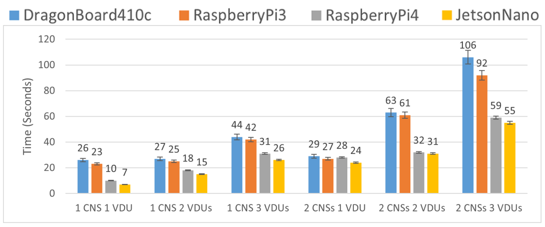


Figure 6: Average CNS instantiation time per SBC.

The analyzes performed in this work serve as a basic foundation for further studies on SBC devices. We used a lightweight virtualization solution (microservices), but even so, instantiating multiple slices was limited, given the need for isolation required by the conceptual nature of slices. In this sense, future investigations should move towards other lighter solutions such as unikernel and FaaS.

REFERENCES

[1] N. Alliance. Ngmn 5g white paper. *Next generation mobile Networks, white paper*, pages 1–125, 2015.

[2] S. Clayman, A. Neto, F. Verdi, S. Correa, S. Sampaio, I. Sakelariou, L. Mamatas, R. Pasquini, K. Cardoso, F. Tusa, C. Rothenberg, and J. Serrat. The necos approach to end-to-end cloud-network slicing as a service. *IEEE Communications Magazine*, 59(3):91–97, 2021.

[3] Docker. Runtime options with Memory, CPUs, and GPUs. https://docs.docker.com/config/containers/resource_constraints, 2021. Accessed: 2021-02-22.

[4] H. Flinck, C. Sartori, A. Andrianov, C. Mannweiler, and N. Sprecher. Network slicing management and orchestration. *Internet Engineering Task Force, Tech. Rep*, 2017.

[5] S. J. Johnston, M. Apetroaie-Cristea, M. Scott, and S. J. Cox. Applicability of commodity, low cost, single board computers for internet of things devices. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 141–146, 2016.

[6] A. Madhavapeddy and D. J. Scott. Unikernels: the rise of the virtual library operating system. *Communications of the ACM*, 57(1):61–69, 2014.

[7] I. Netdata. Github - netdata/netdata: Real-time performance monitoring, done right! https://www.netdata.cloud. https://github.com/netdata/netdata, 2021. Accessed: 2021-02-02.

[8] NVIDIA. Nvidia jetson nano thermal design guide 1.6. https://developer.nvidia.com/embedded/downloads, 2021. Accessed: 2021-02-22.

[9] OpenvSwitch. Open vswitch. https://www.openvswitch.org/, 2021. Accessed: 2021-02-02.

[10] F. Xhafa, B. Kilic, and P. Krause. Evaluation of iot stream processing at edge computing layer for semantic data enrichment. *Future Generation Computer Systems*, 105:730–736, 2020.