A Virtual Object Stack for IoT-Enabled Applications Across the Compute Continuum

GEORGE PAPATHANAIL, University of Macedonia, Greece
LEFTERIS MAMATAS, University of Macedonia, Greece
TRYFON THEODOROU, University of Macedonia, Greece
ILIAS SAKELLARIOU, University of Macedonia, Greece
PANAGIOTIS PAPADIMITRIOU, University of Macedonia, Greece
NIKOS FILINIS, National Technical University of Athens, Greece
DIMITRIOS SPATHARAKIS, National Technical University of Athens, Greece
ELENI FOTOPOULOU, National Technical University of Athens, Greece
ANASTASIOS ZAFEIROPOULOS, National Technical University of Athens, Greece
SYMEON PAPAVASSILIOU, National Technical University of Athens, Greece

5G and cloud computing are set out to address various business needs, often related to the deployment of IoT-enabled applications across the compute continuum. However, the provision of such hyper-distributed applications is far from trivial, due to various interoperability and convergence aspects. In this respect, we introduce a software stack (VOStack) for IoT-oriented virtual objects in order to remove existing openness and interoperability barriers in the convergence of IoT technologies.

We elaborate on physical convergence aspects with an emphasis on networking functionalities in order to address communication issues between IoT devices, IoT gateways, and compute nodes that host virtual objects. In particular, we present control plane functionalities, within the VOStack, for time-sensitive networking (TSN) and reactive routing in order to ensure the timely delivery of time-sensitive data and also alleviate any potential implications due to intermittent connectivity or mobility.

1 INTRODUCTION

Internet of Things (IoT) technologies are evolving at a rapid pace over the last years, leading to the development of various solutions to enable the efficient, secure and reliable deployment of IoT applications [14]. Such solutions include the development of IoT platforms in the form of middlewares to tackle interaction with IoT devices and data fusion aspects, the development of data modeling solutions for the representation of the entities in the IoT world and the collected data, and the development of virtual counterparts of IoT devices and systems in the form of Digital Twins (DTs) [20].

All the aforementioned types of software are provisioned over resources with increased levels of heterogeneity. These resources include -among others- IoT devices deployed usually at the endpoint of provision of the applications, IoT gateways deployed at the extreme edge part of the infrastructure, compute nodes and network management boxes (*e.g.,* Software Defined Networking – SDN switches) deployed at the edge and cloud part of the infrastructure. The term computing continuum is introduced to represent the aforementioned resources and the deployment of IoT applications over resources that span across it.

A set of challenges are identified for managing the deployment of IoT applications across resources in the computing continuum, mostly associated with technologies openness, interoperability and convergence aspects [22]. These challenges include the need for interoperability with different communication protocols, the difficulty to maintain a consistent level of Quality of Service (QoS) in dynamic environments with varying connectivity and high mobility, the need to consider various communication patterns from point-to-point to mesh networks, the need to achieve low-latency communication to serve various IoT applications, and the need to incorporate edge computing capabilities and manage distributed computing resources effectively.

In our work, we consider that a promising way to tackle such challenges is the adoption of virtualization technologies and the development of a software stack that can support the provision of virtual counterparts of IoT devices [20, 23]. The virtual counterparts have to be able to support various IoT device management, networking and security functionalities, while providing open and interoperable interfaces for interacting with both the IoT devices and the edge/cloud computing platforms. We introduce the concept of the Virtual Object (VO) that acts as the virtual counterpart of the IoT device or a virtual counterpart of a group of similar IoT devices (*e.g.,* by interacting with an IoT gateway over an IoT cluster) [23]. This concept is further extended in cases that there is a need to manage information coming from more than one type of IoT devices. In this case, a composite VO (cVO) is considered, able to interact with more than one VOs, fuse information coming by them, as well as enforce joint policies and behaviours to them. The functionalities supported by the (c)VOs are based on the development of a software stack, termed as VOStack [23].

In this paper, we focus on the presentation of the networking mechanisms supported by the VOStack. Such mechanisms are necessary in different real-world scenarios, for instance, managing information flow in a large commercial port. In such a case, IoTs located on transport trucks and/or cargo containers need to maintain

Authors' addresses: George Papathanail, papathanail@uom.edu.gr, University of Macedonia, Greece; Lefteris Mamatas, emamatas@uom.edu.gr, University of Macedonia, Greece; Tryfon Theodorou, theodorou@uom.edu.gr, University of Macedonia, Greece; Ilias Sakellariou, iliass@uom.edu.gr, University of Macedonia, Greece; Panagiotis Papadimitriou, papadimitriou@uom.edu.gr, University of Macedonia, Greece; Nikos Filinis, nfilinis@netmode.ntua.gr, National Technical University of Athens, Greece; Dimitrios Spatharakis, dspatharakis@netmode.ntua.gr, National Technical University of Athens, Greece; Eleni Fotopoulou, efotopoulou@netmode.ntua.gr, National Technical University of Athens, Greece; Anastasios Zafeiropoulos, tzafeir@cn.ntua.gr, National Technical University of Athens, Greece; Symeon Papavassiliou, papavass@mail.ntua.gr, National Technical University of Athens, Greece.

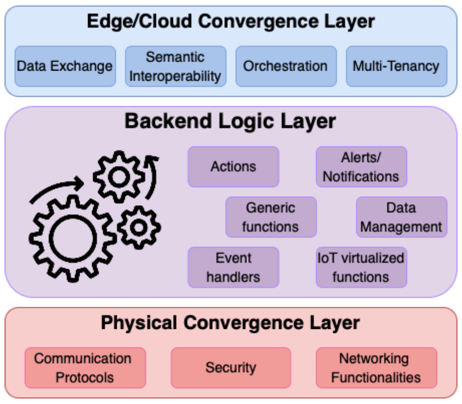

Fig. 1. VOStack layers.

efficient communication in a challenging environment to support a diversity of applications. Mechanisms to tackle such challenges include time-sensitive networking (TSN) functionalities targeted mainly in cases where latency-sensitive application components have to be provisioned at the far-edge part of the infrastructure, and reactive routing functionalities targeted mainly in the case the communication among a group of IoT nodes has to be dynamically managed through their virtual counterparts.

2 VIRTUAL OBJECT STACK

The VOStack is a software stack that aims to enable the convergence of IoT, edge and cloud computing technologies and facilitate the development of distributed IoT applications that can be manageable across resources in the computing continuum [23]. The VOStack aims at addressing challenges related to the convergence, interoperability, and unified abstraction of IoT technologies and mechanisms, as well as challenges related to the convergence of IoT technologies with edge and cloud computing technologies.

With the term IoT interoperability, we refer to both protocols and semantic interoperability. Protocol interoperability aims at supporting (in an agnostic way) various network protocols for the communication between Virtual Objects (VOs) and IoT devices (*e.g.,* HTTP, MQTT, CoAP). Semantic interoperability refers to the support of different data modeling schemes (*e.g.,* W3C WoT, NGSI-LD, OMA LwM2M) for the representation of the data collected by the IoT devices [21]. Semantic interoperability is mostly targeted to IoT application developers.

The VOStack is based on a layered approach with discrete functionalities and objectives per layer. It consists of three basic layers, namely the *Physical Convergence* layer, the *Edge/Cloud Convergence* layer, and the *Backend Logic* layer. A high-level layered view of the VOStack is depicted in Fig. 1.

The *Physical Convergence* layer is responsible for managing the connectivity and communication aspects of the VOs with the different type of IoT devices. It tackles registration, bootstrapping, authentication aspects and management of the network mechanisms applied for the interaction with the IoT devices. A set of network-oriented functionalities are available to facilitate intermittent connectivity of the devices, manage dynamic routing protocols, support TSN mechanisms, and tackle mobility aspects.

The *Backend Logic* layer encompasses a set of functionalities to support the operation of the VOs and cVOs. Each functionality is represented in the form of a virtual function. A virtual function may include IoT device's operational behaviors, enhanced functionalities, and services that the Object/Device can perform. A virtual function may support both processing of the collected data from the IoT device, as well as enforcement of actions or policies to the IoT device. Subset of the virtual functions are going to be generic enough to be applicable in a variety of IoT devices, while another subset may be IoT-device specific.

The *Edge/Cloud Convergence* layer is responsible for supporting mechanisms and interfaces to enable the integration of (c)VOs within application graphs and also render such components orchestratable by edge/cloud computing orchestration platforms. As such, end-to-end orchestration mechanisms can be made available, able to manage the deployment of application components across resources in the computing continuum from the IoT to the edge to the cloud.

3 NETWORKING SUPPORT IN THE VOSTACK

As stated in Section 2, the networking functionalities supported by the Virtual Object (VO) are accommodated within the *Physical Convergence* layer of the VOStack. For instance, in the context of a smart port use case, a network of IoT devices is deployed in the area, ranging from cameras to sensors monitoring storage parameters (*e.g.,* temperature, humidity) mounted on cargo containers. In the former case of IoT devices, minimal latency and jitter are crucial metrics, whereas in the case of cargo container sensors, dynamic routing is crucial, since cargo container stacking, a usual practice in large ports, can create signal interference that may lead to low quality or even connectivity loss, in a rather dynamic mode as these containers are being constantly relocated. A plethora of network-oriented functionalities are provided to support uninterrupted connectivity with the devices, handle dynamic routing protocols, implement TSN techniques, and also address mobility considerations.

The internal structure of the networking functionalities of the VO is depicted in Fig. 2. These functionalities are organized as two distinct elements, specifically the TSN Control plane and Reactive Routing. Fig. 2 illustrates that the internal networking components in the (c)VO may consist of either components exclusive to the cVO (*e.g.,* Clustering and Schedule Engine) or components that are deployed either to VO and cVO (*e.g.,* Network Model and Flow & Path Model).

By utilizing TSN, we will ensure low latency and jitter in the communication between IoT devices and their associated VOs. The deployment of these VOs can be carried out as containers on servers that are positioned at the network's edge, or on network devices (*i.e.,* IoT Gateways) at the far-edge of the compute continuum. TSN will assign greater precedence to the transmission of data pertaining to high-priority traffic communication. This communication may be related to tasks, such as the streaming of videos captured by cameras, the transfer of data collected by sensors, and the dissemination of instructions to control robotic arms. In contrast, other types of traffic, such as best-effort, will be assigned with a lower level of priority. The Qbv-based TSN scheduler [3], specifically the TAPRIO

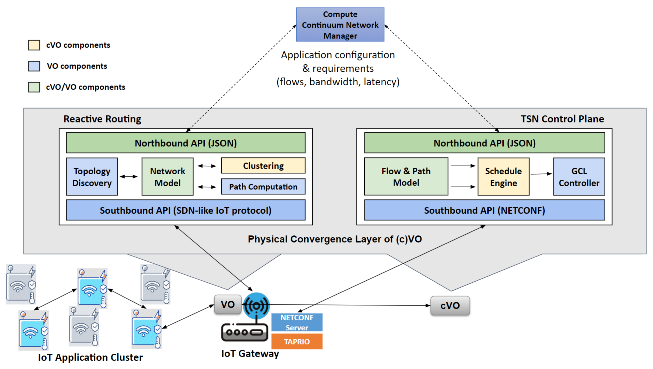

Fig. 2. Physical convergence layer of the VOStack.

qdisc, is employed on IoT Gateways that are placed between IoT devices and (c)VOs.

On the other hand, Reactive Routing maintains network connectivity between wireless IoT nodes and the IoT gateway. The former typically collect measurements from their environment and communicate them periodically to a VO through the IoT gateway. There are also cases of deployments with both sensors and actuators, where the central system does not have the role of measurement collection, but is instead notified of specific events or has a coordinating role with the nodes. The network paths between the wireless sensor nodes and the IoT gateway (or the wireless sensor nodes) are occasionally multi-hop, as their distance may extend beyond their individual network coverage. While traditional IoT network protocols can be employed in such scenarios (*e.g.,* RPL), our primary focus lies on Software-Defined Wireless Sensor Networking (SD-WSN) approaches, which offer several distinct advantages. SD-WSN enables dynamic routing adjustments based on a global view of the network, *e.g.,* handling routing changes due to mobility or signal interference. Furthermore, they provide logically-centralized and programmable routing control, allowing easy integration with the distributed control across the compute continuum, fostering enhanced network intelligence and monitoring capabilities.

4 SOFTWARE-DEFINED WIRELESS SENSOR NETWORKING (SD-WSN)

SD-WSN can support intelligent, programmable, and logically-centralized control mechanisms that dynamically adjust protocol functionalities to achieve improved performance and resource utilization, addressing specific performance demands from IoT applications. However, wireless IoT deployments are often affected by radio signal issues, *e.g.,* due to mobility or interference, which can impair control communication. Furthermore, the latter may also be characterized by increased overhead. To tackle the challenges of intermittent connectivity with the controller, control message scalability, and mobility, several SD-WSN solutions have been proposed in the literature.

For example, they target control channel issues, such as (i) SDN-WISE [2] introducing stateful routing tables and proactive routing decisions to reduce the number of interactions with the controller; (ii) TinySDN [9] implementing a distributed control plane architecture based on multiple controllers; and (iii) Atomic-SDN [4] proposing a time-sliced mechanism that separates the SDN control from the WSN data plane messages using designated flooding periods for the control messages. Application-aware service provisioning is investigated in Soft-WSN [5], implementing topology control, device and network management to meet run-time and application-specific requirements.

CORAL-SDN [25] and its evolution VERO-SDN [27] solutions adopt a modular controller architecture, multiple configurable topology discovery and flow establishment mechanisms [26] and an out-of-band approach to handle control overhead, *i.e.,* utilize a long-range low-rate control interface and a short-range high-rate data interface. Furthermore, SD-MIoT [28] augments VERO-SDN with a better support of mobile IoT nodes, through: (i) mobility-aware topology control, utilizing a hybrid of global and local topology discovery algorithms; (ii) routing mechanisms adapted to mobility employing a hybrid combination of reactive and proactive flow establishment methods; and (iii) an intelligent algorithm that detects passively in real-time the mobile network nodes.

We argue that the solutions above could benefit from their seamless operation over the Cloud-to-Edge continuum, in terms of improved scalability, intelligent adaptability and performance. Our approach builds upon the results from SD-MIoT [28].

4.1 SDN Control Plane

Hereby, we investigate a new SDN controller architecture where the Compute Continuum Network manager implements high-level network management functionalities (*e.g.,* communicates application or flow requirements), the cVO translates such requirements into a customized protocol configuration for a number of VOs and the latter implement autonomous network control features, *e.g.,* topology discovery or flow establishment. As shown in Fig. 2, the (c)VOs implement Reactive Routing through the following SDN control facilities: (i) Topology Discovery, (ii) Network Model, (iii) Clustering, and (iv) Path Computation, which we detail below.

Topology Discovery (TD). We currently support three topology discovery mechanisms, as introduced, extended and utilized in [26], [27], [28], *i.e.,* the Node's Advertisement Flooding (TC-NA), the Node's Neighbors Requests from the Controller (TC-NR), as well as their hybrid combination. TC-NA is an epidemic algorithm inspired by the topology discovery mechanism detailed in [29], employed by the state-of-the-art non-SDN IoT routing protocol IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL). In TC-NA, TD periodically communicates topology discovery control packets to its assigned border router, triggering the broadcasting of "Neighbors' Discovery" short-range beacon message from the latter. Each receiving neighbor creates a response message that informs TD for any link existence between the beacon node and itself. TC-NR is a centralized topology discovery algorithm better aligned to SDN paradigm and dynamic topologies, *i.e.,* collects topology information through individual requests to the nodes from the TD. It is flexible enough to send targeted topology requests on specific nodes or parts of the network without overloading the rest of the topology. Lastly, we also utilize a hybrid combination of TC-NA/TC-NR [28] for heterogeneous topologies that consist of both mobile and fixed nodes, for maximum adaptation to the dynamic characteristics of the network. TD is implemented in each VO and can have bespoke configuration, depending on the deployed IoT application and the network characteristics (*e.g.,* level of dynamicity). The results of TD are also being forwarded to the cVO, *i.e.,* building up the global picture of all deployments.

Network Model (NM). NM is responsible for topology maintenance and representation, *i.e.,* the former ensures that the latter is up-to-date. Although IoT protocols such as RPL use distributed mechanisms residing at the nodes (*e.g.,* trickle timer [29]), our topology maintenance is fully coordinated from the VO, allowing an easy integration of context-sensitive solutions. For example, different static topology refresh periods can be assigned to fixed (*i.e.,* higher values) compared to mobile nodes (*i.e.,* lower values), regulating control overhead and achieving efficient topology maintenance. The topology representation assigns frequently updated link quality values to each edge of the graph, which can be selected from the administrator (*e.g.,* Link Quality Indicator (LQI) or Received Signal Strength Indicator (RSSI)). NM resides both at VOs, *i.e.,* maintaining the topology graph of IoT nodes assigned to it, and the cVO (*i.e.,* keeping track of all IoT nodes in the area).

Path Computation (PC). PC specifies the end-to-end paths from source to destination nodes (*i.e.,* the latter could be the IoT gateway passing the data to the VO). These paths should be aligned to the requirements of the IoT application, *e.g.,* reduce delay, achieve reliable communication, avoid loops or deadlocks, as well as constructing alternative paths. PC determines these paths that are being translated to flow rules in tuples of Destination and Next Hop node addresses, being stored to the individual flow table of each node. In constrained and dynamic IoT environments the flow rule expiration mechanisms are also important. In our case, the flow rule expiration is being handled entirely by the PD, so it can be aligned to other important network control decisions (*e.g.,* topology discovery or maintenance parameters). We currently support four types of flow establishment processes (*i.e.,* being responsible to construct and maintain the forwarding tables of the nodes), the Next-Hop Only (NHO), the Complete Path (CP), the NHO-CP combination as well as the Proactive Flow Establishment (PFE) mechanisms. NHO communicates to the nodes their own forwarding rule only, *i.e.,* to reach the next node, where the CP informs all the intermediate nodes participating in the routing path. An NHO-CP combination is being employed in the case of topologies consisting of both mobile and fixed nodes, where the fixed part employs CP and the mobile the NHO, allowing the maintenance of the dynamic parts of the paths, only. Lastly, PFE employs clustering to classify links based on their connectivity quality history, which guides proactive flow establishment rules.

Clustering. Clustering capabilities are being supported at the level of cVO or Compute Continuum Network Manager for various reasons, including: (i) the appropriate association of IoT nodes with particular VOs, which may also involve a network slicing activity at the IoT network level; (ii) the association of bespoke protocol configuration per node type or characteristic, *e.g.,* mobile nodes may be configured by TC-NR topology discovery and refresh their connectivity status more frequently [28]; and (iii) implement proactive routing by classifying links based on their connectivity quality history [12]. At this point of investigation, we experimented with the K-means algorithm for IoT mobility and a combination of partitional clustering with similarity based measures (*i.e.,* based on dynamic time warping and k-medoids algorithm) to implement proactive SDN-based flow establishment.

In the current work, we employ the infrastructure implemented in the context of [27], *i.e.,* we assume IoT nodes, that support two radio interfaces: a long range interface for the SDN control channel and a short range for data communication, thus implementing out-of-band SDN control. Although we assume one border router (BR) is located at the IoT Gateway, the approach allows for multiple BRs, thus supporting elaborate partition of the IoT infrastructure.

The protocol assumes a control and a data network stacks installed in each IoT node, with the former catering for the long range communication and SDN control messages and processes, while the latter handling low power short range wireless communication and the forwarding layer of the SDN protocol.

4.2 Interactions between SDN Control and Data Plane

The southbound API manages control messages exchanged between the SDN Controller and the IoT nodes in order to support the functionality described in the previous section. Currently, we are investigating the extension of the API as reported in [27], in the lines of which API messages fall under the following categories: topology control, routing control and device control.

The *topology control* messages concern the basic functionalities of topology discovery and maintenance. Thus, this message class includes Border Router (BR) related messages (registration, solicitation, new BR), and messages related to node discovery; for instance, new node response messages, or messages related to the initiation of neighbourhood discovery. The *routing control* message group contains messages related to the establishment of paths among node devices and include all the forwarding rule management actions; for instance, adding forwarding rules to nodes. Finally, the *data delivery control* message group includes messages related to IoT data delivery, *i.e.,* add/remove subscription messages of nodes to data generated by the node, since the VO will be in charge of controlling IoT data delivery, as well.

5 TIME-SENSITIVE NETWORKING

Low-latency communication between IoT devices and their respective (c)VOs comprises a crucial requirement for meeting the various strict service related KPIs (*e.g.,* in terms of response time). TSN comprises an enabler for low latency, as it can provide bounded latency and jitter [13, 15, 16], as well as cater for the protection of high-priority traffic, such that data can be delivered within certain deadlines.

TSN, in particular, comprises a set of standards developed by the Time-Sensitive Networking task group within the IEEE 802.1 working group. These standards define methods for transmitting data over Ethernet networks with high time-sensitivity and determinism. A relevant standard for the purpose of low-latency communication between IoT-(c)VO is IEEE 802.1Qbv [11], associated with the so-called Time Aware Shaper (TAS). More specifically, IEEE

802.1Qbv introduces a transmission gate operation concept for each traffic class queue. At the egress port of a TSN bridge, outgoing packets go through a Traffic Classification block that categorizes different streams to their respective traffic classes. The packets are then enqueued into various traffic classes based on the state of the transmission gates. These gates are either open or closed, and their status is controlled by a Gate Control List (GCL). GCL are commonly configured by the TSN control plane. A widely used instantiation of a TSN control plane is the Centralized Network Control (CNC), which has full knowledge of the network topology and flow requirements. In the fully centralized TSN control plane model, the flow requirements are conveyed to the CNC via a logical entity, namely Centralized User Configurator (CUC).

5.1 TSN Control Plane

As far as the TSN Control plane is concerned, we utilize a prototype implementation of a Central Network Controller (CNC) that consists of three internal modules: (i) Flow & Path Model, (ii) Schedule Engine, and (iii) GCL Controller.

Flow & Path Model. Path and Flow Model module has two main functionalities. The first one is the categorization of incoming flows into distinct traffic classes, such as high-priority and best-effort. This is achieved based on some predefined rules that can match applications' network requirements (*e.g.,* latency less than 1 ms) to traffic classes and determine whether the request should be categorized as critical or non-critical. The second functionality of this module is the path configuration and by path, we define the fixed network between the IoT Gateway and the VO. This path will be used as an input to the Schedule Engine module.

Schedule Engine. The Schedule Engine module is implemented as a cVO component and is responsible for implementing a scheduling model to determine a scheduling pattern for the incoming flows in order to satisfy their latency requirements. In this implementation, the scheduling model is based on constraint programming, since the latter offers versatility in problem modelling, and efficient heuristic search in combination with powerful constraint propagation techniques. It should be mentioned that the TSN scheduling problem is classified as an NP-complete problem and various strategies have been proposed to address this problem in the literature, including Satisfiability Modulo Theories (SMT) [7, 24], Constraint Programming [8], Heuristics [3, 18], and Genetic Algorithms [17].

GCL Controller. The GCL Controller module receives the output of the Schedule Engine as its input and is responsible for configuring time intervals on the Gate Control List, and determining the duration over which each queue is open for transmission. The GCL Controller utilizes YANG [6] in order to send the GCL configuration to the IoT Gateway.

Furthermore, the TSN Control Plane incorporates two application programming interfaces (APIs). The proposed system includes a Northbound API, implemented using a well-defined JSON schema, which is capable of processing requests related to application configuration and requirements from the Compute Continuum Network

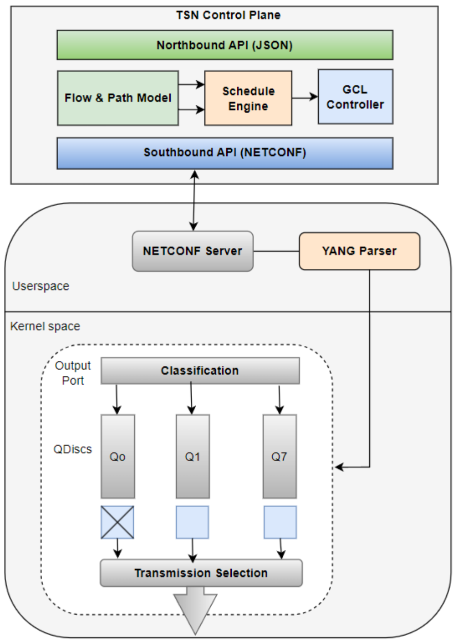

Fig. 3. Interaction between the TSN data and control plane.

Manager. Additionally, a technology-specific Southbound API, utilizing NETCONF [10], is responsible for transmitting the GCL configuration to the IoT Gateway through the use of Remote Procedure Calls (RPC).

5.2 TSN Data Plane

For TSN data plane activation, we utilize TAPRIO (Time-Aware Priority Packet Scheduler) - a powerful queuing discipline available in the Linux kernel's traffic control (tc) tool. To ensure proper packet classification into the appropriate traffic class, TAPRIO utilizes the priority field of the socket buffer (skb) employed by the network stack of the Linux Kernel. This enables TAPRIO to effectively assign time-sensitive flows to their respective priority queues.

In our implementation, we map traffic classes to queues by modifying the DSCP (Differentiated Services Code Point) field of the packet header. As such, we prioritize traffic based on specific service requirements and deliver the desired quality of service (QoS) to different types of data streams. To achieve the modification of the skb priority field before packets are directed to the queuing discipline, we employ the use of iptables, a versatile packet filter tool operating at the IP layer. By incorporating the relevant classifier rules into iptables, we effectively manipulate the skb priority field with precision. As such, we establish the appropriate priority for the skb (socket buffer) as packets traverse the network. Through this comprehensive setup, we effectively integrate TAPRIO into the data plane of our IoT Gateway, enabling the timely delivery of data between IoT devices and (c)VOs.

5.3 Interactions between TSN Control and Data Plane

Leveraging on IEEE 802.1Qcc [1], we utilize a hybrid TSN implementation in order to automate the process of the TAPRIO configuration on the egress interface of the IoT gateway [19]. In principle, CNC communicates with the TSN bridges via remote network management protocols such as NETCONF, RESTCONF and IETF YANG data models. In the case of a client/server-based network management protocol architecture, the TSN bridge acts as a management server, whereas CNC acts as a management client.

In the TSN architectural framework illustrated in Fig. 3, a YANG Parser, deployed at the userspace of the TSN bridge, parses the YANG-TSN model to a set of actions that can be applied directly to the queuing disc layer of the Linux kernel (Fig. 3). The CNC establishes communication through the NETCONF [10] plugin by utilizing the YANG data model. The NETCONF plugin functions as a management client and establishes communication with the NETCONF server that is operational on each TSN bridge, such as an IoT Gateway. Following the completion of their computational process, the TSN schedules are transmitted to the IoT gateway.

6 CONCLUSIONS

In this paper, we presented a software stack for IoT-oriented virtual objects in order to address various convergence and interoperability aspects, mainly in terms of semantics and communication protocols. Such a VOStack can further enhance the functionality of IoT devices by offering the ability to plug into VOs generic and IoT-specific functions that can lead to both performance and energy efficiency gains. We mainly focused on the functionality of the *Physical Convergence* layer of the VOstack, and more specifically on control plane functionalities for TSN and reactive routing in order to address aspects in the communication between IoT devices and their associated (c)VOs at the (far-)edge of the network, such as intermittent connectivity, mobility, as well as (ultra-)low latency requirements.

ACKNOWLEDGMENTS

This work was funded by the European Union's Horizon Europe research and innovation program under grant agreement No. 101070487 (NEPHELE).

REFERENCES

[1] 2019. *IEEE, Std 802.1Qcc-2019: Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks–Amendment 9: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements.*

[2] A. G. Anadiotis, L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo. 2015. Towards a software-defined Network Operating System for the IoT. In *2nd IEEE World Forum on Internet of Things.* 579–584. https://doi.org/10.1109/WF-IoT.2015.7389118

[3] Frimpong Ansah, Mohamed Amine Abid, and Hermann de Meer. 2019. Schedulability analysis and GCL computation for time-sensitive networks. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, Vol. 1. IEEE, 926–932.

[4] Michael Baddeley, Usman Raza, Aleksandar Stanoev, George Oikonomou, Reza Nejabati, Mahesh Sooriyabandara, and Dimitra Simeonidou. 2019. Atomic-SDN: Is Synchronous Flooding the Solution to Software-Defined Networking in IoT? *IEEE Access* 7 (2019), 96019–96034.

[5] Samaresh Bera, Sudip Misra, Sanku Kumar Roy, and Mohammad S Obaidat. 2016. Soft-WSN: Software-defined WSN management system for IoT applications. *IEEE Syst. J.* 12, 3 (2016), 2074–2081.

[6] Martin Bjorklund. 2010. *YANG-a data modeling language for the network configuration protocol (NETCONF).* Technical Report.

[7] Silviu S Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. 2016. Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems.* 183–192.

[8] Jialin Dai, Zhongcheng Wang, and Long Zhong. 2021. Research on Gating Scheduling of Time Sensitive Network Based on Constraint Strategy. In *Journal of Physics: Conference Series*, Vol. 1920. IOP Publishing, 012089.

[9] Bruno Trevizan De Oliveira, Lucas Batista Gabriel, and Cintia Borges Margi. 2015. TinySDN: Enabling multiple controllers for software-defined wireless sensor networks. *IEEE Latin America Trans.* 13, 11 (2015), 3690–3696.

[10] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. 2011. Network Configuration Protocol (NETCONF). In *RFC 6241.*

[11] IEEE. 2016. *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic.* IEEE. 1–57 pages.

[12] Sarantis Kalafatidis, Sotiris Skaperas, Vassilis Demiroglou, Lefteris Mamatas, and Vassilis Tsaoussidis. 2023. Logically-Centralized SDN-Based NDN Strategies for Wireless Mesh Smart-City Networks. *Future Internet* 15, 1 (2023). https://doi.org/10.3390/fi15010019

[13] Gagan Nandha Kumar, Kostas Katsalis, Panagiotis Papadimitriou, Paul Pop, and Georg Carle. 2021. Failure Handling for Time-Sensitive Networks using SDN and Source Routing. In *2021 IEEE 7th International Conference on Network Softwarization (NetSoft).* 226–234.

[14] Bertha Mazon-Olivo and Alberto Pan. 2022. Internet of Things: State-of-the-art, Computing Paradigms and Reference Architectures. *IEEE Latin America Transactions* 20, 1 (2022), 49–63. https://doi.org/10.1109/TLA.2022.9662173

[15] Gagan Nandha Kumar, Kostas Katsalis, Panagiotis Papadimitriou, Paul Pop, and Georg Carle. 2023. SRv6-based Time-Sensitive Networks (TSN) with low-overhead rerouting. *International Journal of Network Management* 33, 4 (2023).

[16] Ahmed Nasrallah, Akhilesh S Thyagaturu, Ziyad Alharbi, Cuixiang Wang, Xing Shao, Martin Reisslein, and Hesham ElBakoury. 2018. Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 88–145.

[17] Maryam Pahlevan and Roman Obermaisser. 2018. Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks. In *2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA)*, Vol. 1. IEEE, 337–344.

[18] Maryam Pahlevan, Nadra Tabassam, and Roman Obermaisser. 2019. Heuristic list scheduler for time triggered traffic in time sensitive networks. *ACM Sigbed Review* 16, 1 (2019), 15–20.

[19] George Papathanail, Lefteris Mamatas, and Panagiotis Papadimitriou. 2023. Towards the Integration of TAPRIO-based Scheduling with Centralized TSN Control. In *2023 IFIP Networking Conference (IFIP Networking).* IEEE, 1–6.

[20] Marco Picone, Marco Mamei, and Franco Zambonelli. 2023. A Flexible and Modular Architecture for Edge Digital Twin: Implementation and Evaluation. *ACM Trans. Internet Things* 4, 1, Article 8 (feb 2023), 32 pages. https://doi.org/10.1145/3573206

[21] Hafizur Rahman and Md. Iftekhar Hussain. 2020. A comprehensive survey on semantic interoperability for Internet of Things: State-of-the-art and research challenges. *Transactions on Emerging Telecommunications Technologies* 31, 12 (2020). https://doi.org/10.1002/ett.3902

[22] C. C. Sobin. 2020. A Survey on Architecture, Protocols and Challenges in IoT. *Wirel. Pers. Commun.* 112, 3 (jun 2020), 1383–1429. https://doi.org/10.1007/s11277-020-07108-5

[23] Dimitrios Spatharakis et al. 2023. A Lightweight Software Stack for IoT Interoperability within the Computing Continuum. In *5th International Workshop on Intelligent Systems for the Internet of Things.*

[24] Wilfried Steiner. 2010. An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks. In *2010 31st IEEE Real-Time Systems Symposium.* IEEE, 375–384.

[25] Tryfon Theodorou and Lefteris Mamatas. 2017. CORAL-SDN: A Software-Defined Networking solution for the Internet of Things. In *IEEE Conf. on Netw. Function Virtualization and Softw. Defined Netw.* 1–2.

[26] Tryfon Theodorou and Lefteris Mamatas. 2017. Software defined Topology Control Strategies for the Internet of Things. In *IEEE Conf. on Netw. Function Virtualization and Softw. Defined Netw.* 236–241.

[27] T. Theodorou and L. Mamatas. 2020. A Versatile Out-of-Band Software-Defined networking solution for the Internet of Things. *IEEE Access* 8 (June 2020), 103710–103733.

[28] Tryfon Theodorou and Lefteris Mamatas. 2021. SD-MIoT: A Software-Defined Networking Solution for Mobile Internet of Things. *IEEE Internet of Things J.* 8, 6 (2021), 4604–4617. https://doi.org/10.1109/JIOT.2020.3027427

[29] Tim Winter et al. 2012. *RPL: IPv6 routing protocol for low-power and lossy networks.* Technical Report. IETF. 641–648 pages.