QoS-Enabled Semantic Routing for Industry 4.0 based on SDN and MOM Integration

P. Bellavista, M. Fogli, L. Foschini, C. Giannelli, L. Patera and C. Stefanelli

2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR), 2021

Cite this as

P. Bellavista, M. Fogli, L. Foschini, C. Giannelli, L. Patera and C. Stefanelli, "QoS-Enabled Semantic Routing for Industry 4.0 based on SDN and MOM Integration," *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, Paris, France, 2021, pp. 1-6, doi: 10.1109/HPSR52026.2021.9481869.

QoS-Enabled Semantic Routing for Industry 4.0 based on SDN and MOM Integration

Paolo Bellavista*, Mattia Fogli†, Luca Foschini*,
Carlo Giannelli†, Lorenzo Patera*, Cesare Stefanelli†
*University of Bologna, Bologna, Italy, {paolo.bellavista, luca.foschini, lorenzo.patera}@unibo.it
†University of Ferrara, Ferrara, Italy, {mattia.fogli, carlo.giannelli, cesare.stefanelli}@unife.it

***Abstract*—Industry 4.0 environments pose unique challenges for the realization of the communication substrate at the shop floor, due to the strict Quality of Service (QoS) requirements, the high heterogeneity of the employed data exchange protocols, and the different network technologies and addressing schema toward the machines. To address those issues, the paper proposes a distributed support based on a Message Oriented Middleware (MOM) and a Software Defined Network (SDN) control plane that coordinate to enable semantic routing by also allowing traffic differentiation as well as in-network processing at intermediate network nodes. Seminal results, collected in realistic industrial settings, confirm the feasibility of our proposal.**

I. Introduction

The forthcoming Industry 4.0 revolution is pushing for the dynamic interaction of industrial processes and data flows produced by machines on the shop floor. To ensure the security and safety of production processes the network serving the shop floor is often segmented in different IP address spaces and configured through complex traffic flow management rules. In addition, there are a plethora of different industrial protocols that have to coexist. As a result, effective, efficient, and Quality of Service (QoS) sensitive packet transmission in these environments is rather challenging.

Indeed, one of the main open issues is enabling communication among machines in a straightforward manner notwithstanding their actual network location and different data exchange protocols/formats, while also ensuring the proper management of per-traffic flow QoS requirements. Although some seminal solutions aim to tackle this problem, they typically focus mostly on the QoS management by largely neglecting dynamic (re)routing issues [1]–[4].

To fill that gap, this paper proposes to adopt an overlay networking solution that acts as a semantic routing substrate that works both at application and network layers to enable remote nodes identification (in particular for remote industrial equipment) based on network trunks and elements that packets should traverse, rather than the private and time-variable IP address of the destination. Such overlay network is in charge of handling traffic flows by also considering their QoS service requirements and granting integrated management of such flow at both application (i.e., via Message Oriented Middleware – MOM – message brokering) and networking (i.e., via Software Defined Network – SDN – network path control) layers. Moreover, our solution originally enhances packet headers by adding different types of tags to drive the data dispatch and to specify the required QoS and the processing actions that might be executed at traversed network nodes according to the specific semantics of the data payload. For instance, in case of traffic congestion, data aggregators at the edge could manage packets containing a specific semantic (temperature values provided by a given sensor) by sending only periodically an average value.

978-1-6654-4005-9/21/$31.00 ©2021 European Union

In particular, the proposed architecture consists of a local MOM broker, in our solution implemented as an Advanced Message Queuing Protocol (AMQP) broker, and an SDN controller that is aware of the actual and current addressing of services and deployed equipment. Our solution eases the MOM-based brokerage of data depending on (abstract) machine identification/location, specific data models, and data flow semantics, including also QoS requirements. At the same time, the SDN controller interacts with the MOM broker and, in its turn, deploys mapping rules on network elements, by considering QoS requirements in a per traffic flow and fully dynamic manner. We implemented the proposed architecture and we run some seminal tests that demonstrate its feasibility in a realistic deployment scenario.

The remainder of the paper consists of five sections. Section II presents the main design principles of our solution, while Section III introduces the proposed distributed architecture. Section IV reports collected experimental results. Section V presents a selection of related efforts in the area, and Section VI concludes the paper.

II. Principles and design guidelines for QoS-enabled semantic routing

Future industrial networks will be characterized by an unprecedented degree of heterogeneity and complexity. Traditional solutions, mainly based on the direct interconnection of machines one to each other and machines towards the control room, cannot provide the required degree of flexibility. Therefore, there is the need to adopt novel solutions able not only to manage the deluge of data generated by the underlying industrial machinery layer, but also to allow the dynamic and flexible reconfiguration of the topology driven by QoS requirements. By considering the momentum of MOM as an enabler of the Industry 4.0 vision, we believe it will become a pillar of future industrial ecosystems. However, let us note

that even if the adoption of MOM enables critical features to facilitate message dispatching by making it independent on actual machine location, it does not provide features allowing to control how packets flow through middle network devices along the data routing path. In fact, once a message is sent from a broker to a receiver (or vice versa), the path the message traverses and the QoS network elements provide cannot be managed, since out of the visibility and control scope of the MOM. While this approach allows to abstract from low-level networking details, the ability to dictate the behavior of underlying network devices is essential in industrial networks that typically require satisfy of stringent QoS requirements.

To address the above open issues, we claim the importance of adding an SDN control plane so to ease the management and dynamic reconfiguration of network elements that act as the (distributed) communication substrate between the machines and the MOM. In addition, the SDN controller could provide network-wide abstractions to define and enforce fine-grained network policies. Accordingly, we propose to integrate MOM and SDN paradigms in a joint solution that allows the QoS-enabled dispatching of MOM-based messages by also limiting the management burden on network operators and machine technicians. At the top level, the MOM infrastructure is in charge of identifying the set and (abstract) location of destination nodes a message should be dispatched to. At the bottom level, application gateways, deployed close to machines, act as intermediaries among machines (and their proprietary industrial protocols) and the MOM (based on widely adopted standard protocols) by allowing to specify the required QoS in a semantically enriched manner. In the middle level, the SDN controller exploits its centralized point of view to (re)configure the communication substrate, and the network elements therein, in relation to the current state of the network as well as QoS requirements identified by application gateways.

Based on the collaboration among the MOM infrastructure and the SDN controller, network elements can be properly configured to i) select the best route towards the destination and forward messages accordingly, ii) manage competing traffic flows in a coordinated manner, e.g., to ensure prompt dispatching of mission-critical messages even if at the expense of less critical messages, and iii) enforce in-network processing to reduce the network utilization, e.g., by merging consecutive packets in only one.

To this purpose, we enhance packets by adding custom tags to specify how network devices should manage them in terms of QoS and processing. Based on such tags, network devices exploit dynamically deployed traffic rules and ad-hoc software modules to enable proper rerouting, per-traffic flow prioritization, and in-network processing with the goal of significantly improving networking performance and reducing the overall network overhead. For example, by considering two traffic flows between the MOM broker and a machine, proper routing table management allows to forward traffic flows tagged as "mission-critical" via a large-bandwidth low-latency path (if available). In addition, a QoS software module can selectively delay packets of traffic flows tagged as "not-urgent", where the magnitude of the imposed delay can also depend on the current level of network saturation. Finally, an in-network processing module can exploit the knowledge about the carried data model to manage packet content. In particular, let us note that the knowledge of packet semantic allows the adoption of a wide range of highly expressive traffic flow management mechanisms. For instance, it is possible to forward packets only if they satisfy a given rule, e.g., if they carry temperature values greater than a threshold, or to apply functions to send pre-processed values, e.g., sending only one packet with the average temperature resulting from a series of received temperature values.

From a functional point of view, the in-network processing layer sits atop the data forwarding layer. As in the case of SDN deployment, we do not argue that all the network devices should provide the in-network processing capability. Instead, we promote a pragmatic approach where legacy and novel solutions cooperate effectively. Since the SDN controller holds a network-wide view, it knows which network devices offer in-network processing functions and which not. Therefore, traffic can be optimally handled by maximizing the in-network processing (e.g., routing of packets carrying values that can be averaged towards network devices providing that aggregation function) while ensuring QoS requirements.

To support the dynamic and semantic-driven management of routing, QoS, and in-network processing, we propose to enrich packet semantic with the following five tags: source ID, destination ID, QoS, in-network processing, and data model:

- source ID and destination ID are logical names that decouple the "who" and "where." In fact, the overload of semantic affecting traditional IP addresses intrinsically prevents seamless mobility. Moreover, since messages are actually sent to the broker by application gateways, it is not possible to exploit the sender IP address to discriminate among different machines;
- the QoS tag allows to specify if a traffic flow should be considered either as mission-critical or non-critical, possibly defining multiple levels. Of course, this tag can be enriched with other priority levels in case it is required to support finer-grained management of traffic flows, such as gold/silver/bronze traffic types [5];
- the in-network processing tag provides information about the possibility to process the packet directly on network devices. When in-network processing is allowed, such a tag also specifies the function to apply and the target field(s);
- the data model tag identifies the syntax of the payload. Both the SDN controller and the in-network processing modules deployed on network devices take advantage of that tag. The former exploits the information provided by the data model tag to decide where to route the packets (perhaps not all the network devices can process all the data models), whereas the latter takes advantage of it to understand how to process the packet.

III. SDN-MOM Distributed Architecture

The proposed architecture, mostly working at the application layer, adopts the typical SDN approach by identifying two main areas: *Control Plane* and *Data Plane*. In the *Control Plane* area we deploy: the MOM controller, interacting with the MOM broker; the SDN controller, controlling network elements; and the Gateway controller, managing the many application gateways deployed in the environment. In the *Data Plane* area takes place the implementation of: the MOM, the Gateway components, and, in the middle, network elements equipped with in-network processing modules running atop them and managed by the SDN controller.

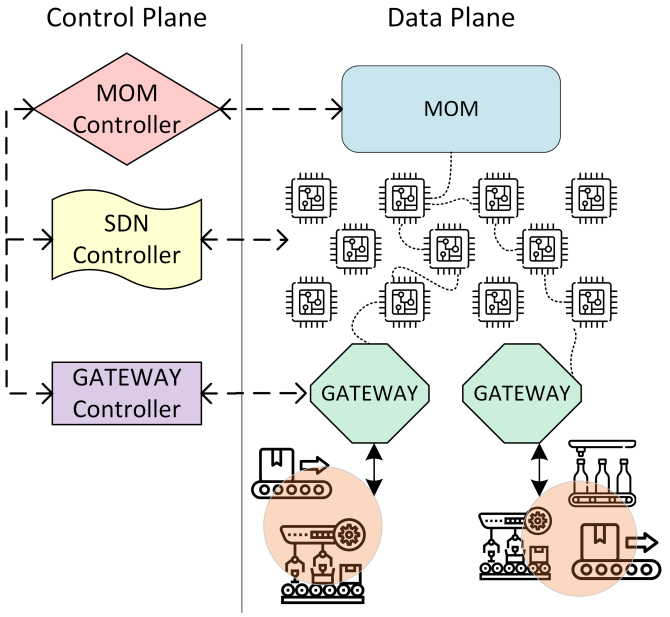

Fig. 1. Functional/layered view of the SDN-MOM distributed architecture.

Each component has different duties and responsibilities:

- The **MOM Controller** is demanded to sniffing and re-routing the traffic flowing into the MOM topics. It subscribes to several MOM topics to receive messages and to analyze the traffic. It consumes part of the header and can return back onto the MOM the message, performing decisions based on the header and on the information received from the SDN Controller and the Gateway Controller. The messages can be forwarded to a specific topic, duplicated among different topics, or consumed and pulled out from the flow. At the same time, the MOM Controller issues information that will be used from the SDN controller to correctly configure the SDN devices for achieving the desired level of QoS on the specific flow.
- The **Message-Oriented Middleware**-MOM is one of the core pieces of our infrastructure. It is the logical single point of communication between several firm sectors. It contains topics written by the Gateways and can be read by multiple other Gateways, based on the plant communication requirements. The MOM is responsible for guaranteeing differentiated QoS policies with different semantics. Typically, the at-most-once semantic can be used for best-effort machinery traffic. Otherwise, at-least-once semantic can be used for monitoring mission-critical assets and for control traffic. Moreover, some messages can be sent with high priority, guarantying differentiated traffic management and avoiding congestion.
- The **SDN Controller** centralizes network intelligence in a separate component, disassociating the packet forwarding process from the routing processes. The SDN Control Plane consists of one or more controllers that are considered the brain of the network, where all intelligence is embedded. The SDN Controller configures the network resources. In our infrastructure, the SDN Controller has full knowledge of the network and the paths, guarantying a fine-grained ruling of the traffic coming from the Gateways. Differentiated policies can be applied based on the content of the messages, following the MOM Controller rules. The traffic can be duplicated, aggregated, blocked, and re-routed on different data paths. The SDN Controller role is dual: on the one hand, it has to gather and make accessible to the MOM Controller information about machinery position, router congestion, and gateway policies. On the other hand, it has to translate the MOM Controller policies to actual configurations on network entities.
- The **Gateway Controller** emits control messages directed to the Gateway of the infrastructure. It works in strict coordination with the MOM Controller and SDN Controller, to avoid congestions and to maintain topic abstractions coherent with the real machine distribution. Its management duties comprehend: checking of the state of all the gateways, which must be configured coherently with the machine on which are acting; synchronization with SDN and MOM controllers, that can send re-configuration messages to avoid congestion. Practically, it can manage the header that is applied from the Gateways to each packet, modify the priority of the messages, and define levels of QoS applied directly to the data-extraction phase.
- The **Gateway** duties comprehend the data gathering, the data transformation to an internal MOM-specific representation, the header addition, and the interconnection between the industrial machinery world and the MOM topic-centric world. In industrial scenarios, it is common to have machines that use different languages and protocols for data exporting and representation (e.g. Modbus, Profibus, OPC UA, OMG DDS, EtherCAT [6]). For this purpose, the Gateways can be specialized with ad-hoc libraries and push or pull strategies based on the specific machinery from which to gather information. Moreover, the QoS can be managed directly at this level, avoiding high useless throughput when the plant is working in a normal condition.

Fig. 1 depicts a schematic of the entire infrastructure. Dashed paths between controller entities in the control plane, and between control and data planes represent the management/configuration data exchanges that are logically separate from data flows. Continuous line paths represent data flows between machines and our architecture. Finally, dotted paths represent network data paths dynamically (re)configured according to the SDN Controller policies.

IV. Performance analysis

To demonstrate the feasibility and efficiency of the proposed solution, the section outlines primary performance results achieved based on our proof-of-concept prototype. On the one hand, Section IV-A shows how the adoption of our gateways allows to enrich packets with meaningful tags, sent by a plethora of gateways to a (logically) centralised MOM broker. On the other hand, Section IV-B focuses on the computational overhead imposed on intermediate nodes performing in-network processing, with the primary goal of demonstrating its scalability in terms of packet rate on nodes with relatively limited hardware capabilities.

The testbed comprises 4 Amazon EC2 VMs based on Ubuntu Server 20.04 LTS and equipped with 1 vCPU and 1 GB of RAM. The first VM acts as industrial machinery simulator, simulating an arbitrary number of machines transmitting messages based on pre-defined templates. The second VM acts as Gateway, collecting packets from the machines, enriching packets with proposed tags, and routing them to the MOM broker. The third VM hosted between the Gateway and the MOM broker runs the in-network processing module. Lastly, the fourth VM acts as a MOM broker.

A. Integrated SDN-MOM Semantic Routing

The SDN Controller and in-network processing modules act as core functional services in our architecture. As the reader may know, the main SDN implementations (e.g., Openflow [7]) work by forwarding to the SDN controller the packets of the flow not having a configured compatible rule for the delivery. The SDN Controller, thanks to its complete knowledge of the monitored locality, can make different routing decisions based on information about packet source or destination addresses, current network topology, and traffic, and also taking into account information arriving from external sources, in our case the MOM and Gateway Controllers.

Our implementation configures the networking rules based on application-specific header information to perform semantic routing on the network devices and on top of the MOM. In order to have in each moment the correct header associated with the payload, we decided to enrich the messages directly at the Gateway level of the infrastructure, so that only complete messages traverse the platform. The gateways are configured to set for each machine the specified header, containing information about QoS, but also additional information about machine `geo_position`, and `innet_processing`. In the following Listing 1, there is an example of the header added to each message ingested in the data platform.

```
"machine_id":"m00000001",
"machine_serial":"123456789",
"source_id":"m00000001",
"destination_id":"rabbitmq",
"geo_position":{
        "continent_name":"EU",
        "city_name":"Bologna",
        "country_iso_code":"IT",
        "region_name":"Emilia-Romagna",
        "location":{
                "lat":"44.493690",
                "lon":"11.343080"
        }
},
"innet_processing":{
        "func":"sum",
        "field":"TEMP_1.value"
},
"payload_description":"sensor_temp",
"QoS":5,
"message_rate":100
```

Listing 1. JSON header example

Going further from pure application-level additional information, `message_rate` and the `QoS` tags need in-depth analysis. The `QoS` tag contains an integer from 1 to 10 that commands the desired quality in the entire platform. The SDN Controller exploits that tag to guarantee a correct path for reaching the quality expected and, where appropriate, for reducing the data quantity using the `innet_processing` additional details. The `message_rate` tag is expressed in milliseconds and represents the time interval between a data gathering and another.

Since the data volume traversing the platform can be managed by both Gateways and in-networking modules, we have conducted some tests to demonstrate how those techniques can be used in a conjunct profitable manner. In our first test, we exploit the Gateway capabilities of modulating the traffic entering the platform based on the `message_rate` tag. For this experiment, we defined three `message_rate` levels: bronze (110 ms), silver (60 ms), and gold (30 ms). During the execution, we change the levels by communicating the `machine_id` and the new `message_rate` values to the Gateway Controller that will re-configure the correct Gateway. On the MOM VM, we run the AMQP broker implementation (the open-source RabbitMQ [8]).

Fig. 2 shows the three levels of data gathering modified dynamically during one execution of about 20 minutes. At time 6:56 the `message_rate` changes from bronze to gold to supply the higher detail of the processes running on machinery. At time 12:16 the `message_rate` is set to silver quality.

B. In-network Processing

This section presents the experiments conducted to investigate the overhead introduced while performing in-network

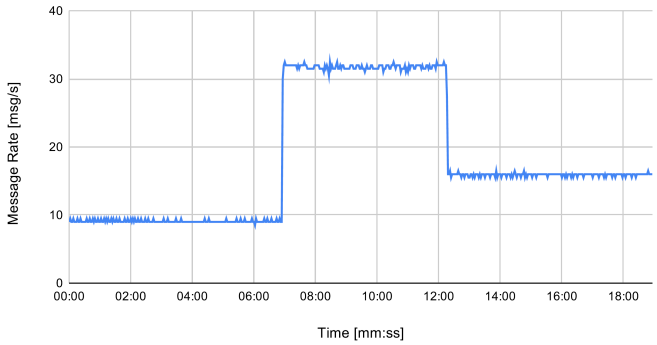

Fig. 2. Bronze, gold and silver levels managed on Gateway.

packet processing tasks to aggregate packets being part of the same flow. For the sake of simplicity, the Gateway directly sends all generated traffic to the VM running the in-network processing module. We implemented an in-network processing module that sorts the incoming traffic in queues and performs an average function on fields representing temperature values when a queue grows to 10 elements. Such a module takes advantage of the tags we introduced to enrich the packet semantic and identifying packets of different flows. In particular, a flow is identified by the logical names of sender and receiver (source ID and destination ID, respectively) and the aggregation function. Using that information, the in-network processing module inspects packet headers and sorts packets in different queues according to their tags.

As soon as a queue reaches a certain threshold, i.e., 10 packets, the in-network processing module applies the aggregation function on the target payload fields and sends out a single packet containing the computed result, i.e., the average value. The primary outcome is a sharp decrease of packets sent to the following node, since the in-network processing module lowers the overall network usage by in-situ processing of packets in relation to the in-network processing configuration, 90% in the case above.

However, such a network traffic reduction entails the following (at least) two important consequences. First, since packets are inspected, queued, and processed, they reach their destinations with an additional delay, due not only to packet queuing, but also to packet processing. Therefore, packets with strict QoS requirements may not be eligible to flow through in-network processing modules (the SDN controller must dictate alternative paths for them accordingly). Second, the in-network processing task itself may require more computational resources than those available when overwhelmed by high network traffic volumes. Through the experiments described in the following, we seek to evaluate how the incoming packet rate, namely, number of packets per (s)econd, affects the performance of in-network processing. In other words, we aim to outline the relationship between the incoming packet rate, the CPU load, and the outgoing packet rate.

The experiments vary along two dimensions. The first dimension is the percentage of CPU exploitable by the in-network processing module. Since the in-network processing module can be deployed on nodes already performing other activities, we assume that it cannot take advantage of all the available processing capabilities, since reserved for other functions. To this end, we limit the CPU usage of the in-network processing module to 25% and 50% of the overall processing availability using `cpulimit` [9]. The second dimension is the packet rate sent by the application gateway to the in-network processing module. We generate increasing packet rates ranging from 2K to 16K in step of 2K packets/s.

To make the experiments reproducible in an automated manner, we developed a test framework based on Ansible [10], a configuration management tool, and several shell scripts. The choice of Ansible over other configuration management tools available on the market is mainly due to its intrinsic simplicity. In fact, Ansible does not require anything else except an SSH connection between the control node (where Ansible is installed) and managed nodes. We use `tcpdump` to capture the network traffic and a Python package called `psrecord` [11] to monitor the CPU activity of the in-network processing module.

As Fig. 3 shows, the in-network processing module can successfully handle the totality of the incoming packets with 50% of the CPU and incoming packet rates up to 12K packets/s. The CPU usage grows up to nearly 100% of the total when the incoming packet rate is 16K packets/s. Interestingly, in that circumstance, the in-network processing module can handle roughly 80% percent of the incoming packets (by losing the remaining 20%). In the light of those results, the SDN controller may decide to redirect up to 12K packets/s towards the in-network processing module to make possible the 100% of processed packets while not overloading the CPU. As the CPU maximum availability decreases to 25% of the total, the in-network processing module cannot handle more than around 6K packets/s. However, it is interesting to note how the performance dramatically decreases as the incoming packet rate grows. As Fig. 4 shows, with an incoming packet rate equals to 16K packets/s, the in-network processing module cannot even handle 50% of packets. On the one hand, in-network processing can lower the number of packets traversing the network and reduce network utilization consequently. On the other hand, without detailed knowledge on the limits of the in-network processing modules deployed on the field, they can quickly become bottlenecks for the whole network.

V. Related Work

Traditionally adopted in data centers and campus networks, the SDN paradigm has also recently emerged as a novel solution to address the increasing network complexity characterizing forthcoming industrial networks. [12] proposes an SDN IIoT architecture that aims to cope with the wide and heterogeneous range of resources being part of intelligent manufacturing environments while promoting a flexible network resource management. Integrating SDN and Edge Computing into IIoT, [1] proposes an adaptive routing mechanism to

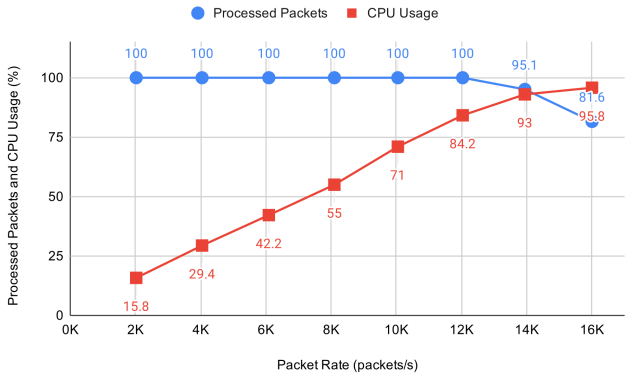

Fig. 3. CPU usage limited to 50% of the total.

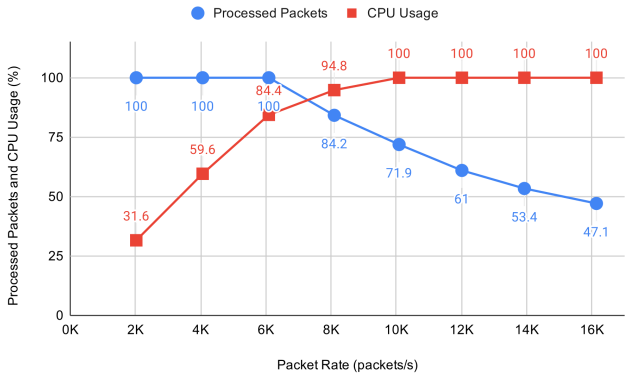

Fig. 4. CPU usage limited to 25% of the total.

dispatch packets with different delay constraints in a per-flow tailored manner while taking into account time deadlines, traffic load balances, and energy consumption. Similarly, [2] adopts SDN in IIoT environments to efficiently handle the edge-cloud interplay by considering the energy efficiency, latency, and bandwidth. In conclusion, some efforts regarding edge IoT scenarios combined with industrial wireless sensor networks. [3] proposes an SDN-based solution to cope with the pressing real-time constraints that typically affect industrial processing. Such a solution aims to handle both transmission scheduling and node mobility by ensuring bounded end-to-end delays. Using a fog-based architecture for Industry 4.0, [4] aims to improve industrial performance by adopting end-to-end QoS control.

Compared with this previous work, we propose a MOM-integrated SDN-based solution to enable QoS semantic routing for Industry 4.0. Furthermore, we promote in-network processing as a fundamental player in lowering the deluge of data that will affect future industrial networks.

VI. Conclusions and Future Work

The paper presents an innovative solution exploiting the SDN approach to support the semantic routing of traffic flows within a MOM-based Industry 4.0 environment. In particular, leveraging gateways deployed close to industrial equipment we enrich with semantic information packets sent to the MOM broker adding tags that specify abstract equipment location information, desired QoS, and carried content. Then, SDN-based in-network processing modules deployed on network elements efficiently perform fine-grained per traffic flow management actions by exploiting tags injected in packets.

Boosted by obtained performance results, we are now working on further research activity directions. In particular, we started investigating the adoption of a federated approach, allowing the dynamic and efficient dispatching of MOM-based messages between different Industry 4.0 domains. In addition, we plan to extend our experimental testbed by exploiting wider and more complex topologies.

References

[1] X. Li, D. Li, J. Wan, C. Liu, and M. Imran, "Adaptive transmission optimization in sdn-based industrial internet of things with edge computing," *IEEE Internet of Things J.*, vol. 5, no. 3, pp. 1351–1360, 2018.

[2] K. Kaur, S. Garg, G. S. Aujla, N. Kumar, J. J. P. C. Rodrigues, and M. Guizani, "Edge computing in the industrial internet of things environment: Software-defined-networks-based edge-cloud interplay," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 44–51, 2018.

[3] L. L. Bello, A. Lombardo, S. Milardo, G. Patti, and M. Reno, "Experimental assessments and analysis of an sdn framework to integrate mobility management in industrial wireless sensor networks," *IEEE Trans. on Industrial Informatics*, vol. 16, no. 8, pp. 5586–5595, 2020.

[4] N. B V and R. M. R. Guddeti, "Fog-based intelligent machine malfunction monitoring system for industry 4.0," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2021.

[5] P. Bellavista, A. Dolci, and C. Giannelli, "Manet-oriented sdn: Motivations, challenges, and a solution prototype," in *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, 2018, pp. 14–22.

[6] P. Drahoš, E. Kučera, O. Haffner, and I. Klimo, "Trends in industrial communication and opc ua," in *2018 Cybernetics & Informatics (K&I)*. IEEE, 2018, pp. 1–5.

[7] *ONF SDN Evolution*, Open Networking Foundation, 9 2016, version 1.0. [Online]. Available: https://opennetworking.org/wp-content/uploads/2013/05/TR-535_ONF_SDN_Evolution.pdf

[8] "Rabbitmq," https://www.rabbitmq.com/, [Online; accessed 4-May-2021].

[9] "Cpu usage limiter for linux," https://github.com/opsengine/cpulimit, [Online; accessed 5-May-2021].

[10] "Ansible," https://www.ansible.com/, [Online; accessed 2-May-2021].

[11] "Record the cpu and memory activity of a process," https://github.com/astrofrog/psrecord, [Online; accessed 5-May-2021].

[12] J. Wan, S. Tang, Z. Shu, D. Li, S. Wang, M. Imran, and A. V. Vasilakos, "Software-defined industrial internet of things in the context of industry 4.0," *IEEE Sensors Journal*, vol. 16, no. 20, pp. 7373–7380, 2016.