SDN-based configuration solution for IEEE 802.1 Time Sensitive Networking (TSN)

Siwar BEN HADJ SAID, Quang Huy TRUONG, and Michael BOC
Institut LIST, CEA
Université Paris-Saclay
F-91120, Palaiseau, France
{siwar.benhadjsaid, quanghuy.truong, michael.boc}@cea.fr

ABSTRACT

Maintenance and update of Time-Sensitive Network (TSN) configurations pose an immediate challenge on the "Time-to-Integrate" aspect of new devices and traffics: adding new communicating sensors on production lines, adding new Engine Control Units (ECUs) or new software applications in a car. Meanwhile, the Software-Defined Networking (SDN) approach has proven its effectiveness to ensure proper quality of service for ongoing traffics even in evolving topologies and its ability to integrate a multitude of features (configuration parameters, specific metric computation, etc.). In this regard, this paper investigates a first step in the instantiation of the fully centralized IEEE 802.1Qcc model using the SDN approach targeting industrial and automotive contexts. We discuss how SDN can speed-up the Time-to-Integrate process by analyzing how one of the most important TSN standards, i.e., IEEE 802.1AS can be configured in such a framework.

CCS Concepts

•**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

Keywords

TSN, SDN, time synchronization

1. INTRODUCTION

Factory digitalization or zero defect manufacturing are ongoing initiatives that put stress on existing communication network infrastructures designed to ensure forwarding of critical traffics between machines. In the automotive sector as well, new hardware and software are proposed to ensure better comfort and new features for autonomous driving sharing the same network infrastructure designed for legacy critical control/command traffics. In those two sectors, there is a need to facilitate the coexistence of critical, time sensitive traffics with less critical or best effort traffics. In this sense, the IEEE Time Sensitive Network (TSN) working group proposes a set of amendments to IEEE standards that features time synchronization, traffic scheduling, gates control, and frame preemption to name a few in order to achieve such seamless coexistence.

Precise scheduling of traffics using TSN standards requires, however, a complete knowledge of the network infrastructure and of traffics. Indeed, although it is possible to reserve resources for best effort traffics, there are different levels of requirements even in non-critical traffic that may cause intricate configurations, e.g., setting critical traffic as best effort to ensure a bounded latency of non-critical traffics while ensuring that the final forwarding will ensure the maximum latency of critical traffics. Such configurations are time-consuming (each standard specifies a certain number of parameters and constraints that are inter-dependent) and require offline configuration/simulation tools to ensure the respect of traffics constraints.

In non-deterministic Ethernet networks, Software-Defined Networking (SDN) is a paradigm that provides a great range of freedom to flexibly and centrally reconfigure networks. The basic idea of SDN is to control the behavior of network devices by a logically centralized controller. Centralized configuration approaches adapted from the concept of SDN can help to overcome the issue of re-scheduling traffic and deploying the TSN configuration on the network.

This paper explores an SDN-based instantiation of the IEEE 802.1Qcc fully centralized architecture around the configuration of the time synchronization standard. This standard is central to the operation of gates (IEEE 802.1Qbv) and to the reconstruction of clocks for multimedia traffics (IEEE 1722). We list the constraints and the requirements of the time synchronization standard in the automotive and industrial contexts, and propose an approach to ensure their fulfillment in a dynamic environment using an SDN framework.

The rest of the paper is structured as follows: Section 2 explores the ongoing efforts around the centralization of deterministic network management. Then, Section 3 discusses through concrete use cases, why SDN could be beneficial for TSN network management. Section 4 gives an overview of the SDN solution used for this IEEE Qcc instantiation. Section 5 describes the mechanisms deployed to configure time synchronization in a testbed (IEEE 802.1AS). Finally, Section 6 concludes this paper.

RTN18, July 2018, Barcelona, Spain

2. RELATED WORK

The configuration of real-time networks is a complex problem that has been highlighted in the past [15]. [9, 16, 7] were among the first studies to discuss, via use cases, the benefits that SDN could provide to the management systems of industrial networks. [11] was the first work to propose a proof-of-concept where the SDN concept is applied to real-time Ethernet. However, this proof-of-concept was more about the integration between OpenFlow protocol and Powerlink real-time Ethernet to show how the SDN manage the datapath in special use cases such as link failure. In our work, we want to propose a complete SDN solution able to configure all TSN features even the ones that are not related to datapath management such as the time synchronization configuration.

The relevance of our work is highlighted by the fact that two major standardization bodies, IEEE and IETF, are working on standardizing an automatic and flexible configuration framework for deterministic networks. The IEEE 802.1 TSN Working Group (WG) proposed the amendment IEEE 802.1Qcc [13] where three network configuration models are specified: fully distributed, centralized network/distributed user, and fully centralized. In term of managing traffic schedules, paths for data, redundant paths and time synchronization, a centralized configuration approach seems to be more relevant for TSN configuration. Also, the IETF DetNet WG prefers having a centralized control and management plane that will ensure deterministic data paths that operate over Layer 2 bridged (i.e. through the collaboration with the centralized network configuration model of TSN WG) and Layer 3 routed networks [8]. The fully centralized configuration model is depicted in Figure 1. It is composed of Centralized User Configuration (CUC) entity and a Centralized Network Configuration (CNC). While the CUC is responsible for building up the applications' requirements, computing the configuration setting and enforcing it (e.g. setting up gates schedules, reserving resources, etc.) in Bridges are done consistently by CNC. Therefore, CNC will be in charge of configuring TSN features namely credit-based shaper, frame preemption, scheduled traffic, per-stream filtering and policing, and frame replication and elimination for reliability. The IEEE 802.1Qcc configuration model still lacks many details about the functionality to be included in the CNC and how it will ensure the management of Bridges. Several existing management protocols can fill the gap such as NETCONF, RESTCONF, and SNMP.

[4, 10] proposed to combine between machine to machine communication protocol (OPC UA) and TSN networks in order to respect the constraints imposed by industrial automation. Both of these studies rely on the fully centralized model of IEEE 802.1Qcc and use NETCONF protocol for the communication between CNC and Bridges in the network. [10] presents, in details, the different components as well as the workflow of their configuration solution. However, the proposed solution was particularly focusing on the self-configuration of traffic scheduling feature. In addition, the feasibility of the solution has not been proven.

Unlike the above studies, we are proposing an SDN-based instantiation of IEEE 802.1Qcc fully configuration centralized model that is able to configure any TSN feature in the network. Moreover, we are showing, via a testbed, how our solution allows to configure the time synchronization feature.

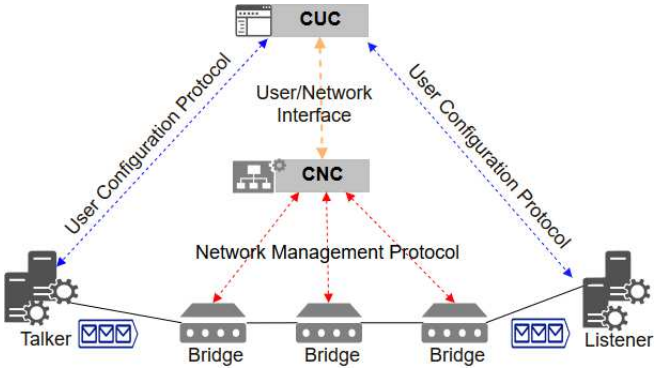

Figure 1: Fully centralized network model of IEEE 802.1Qcc

3. WHY DO WE NEED SDN FOR TSN NETWORKS CONFIGURATION?

The anticipated configuration complexity of TSN networks is a very hot topic in automotive and industrial contexts. In fact, adding new TSN mechanism leads to an increased number of parameters to be configured in End stations and Bridges. Moreover, choosing the adequate values for these parameters depends on different information such as traffic pattern, network topology, etc. Therefore, during engineering and validation tests, the effort of configuring the networking devices becomes high and severely influences the competitiveness.

Usually, the configuration/reconfiguration process of an automotive or industrial network requires the use of offline configuration tools such as simulation or network modeling tools. For example, because of the changing of network requirement, firstly, network engineers need to re-examine the network characteristics e.g. all traffic pattern (e.g. message size and periodicity, source and destination, traffic constraints), characteristics of each device (e.g. processing capabilities, the mean residence time of messages inside the device, queue sizes, etc.) and each link (e.g. bitrate, propagation delay, etc.). Based on such characteristics, an appropriate configuration for the network is generated using an offline tool. After that, each device will be configured manually according to the new configuration. The increasing number of parameters, that needs to be configured, causes extra costs during engineering, testing, and validation stage as configuration errors may happen. In addition, the Time-to-Integrate of the new configuration is also one of the major concerns while this whole manual process must take time.

When the change in the network topology or traffic pattern happens more frequently, the static and manual configuration manner is no more adapted. Therefore, a dynamic and centralized configuration solution such as SDN is required to manage such situations. In the following, we provide some examples where we can see that the manual configuration is no more suitable.

3.1 Change in network topology

Adding a new device or a new feature to the network will lead to generating a new configuration setting using offline tools, shutting down all networking devices and reconfiguring them. For instance, adding new ECU to automotive network leads to adding new traffic. Consequently,

after determining the pattern of new traffic, new gates schedules (IEEE 802.1Qbv) should be calculated using the offline tool with respect to the QoS of the whole existing traffic. After shutting down the network, the new schedules should be configured in the networking devices.

With the SDN solution, the network reconfiguration can be done on-the-fly without interrupting the network operation. The SDN controller may detect the added device and configure it. It can also detect the new traffic pattern by monitoring the message frequency, size, and destination. Based on that information, the controller determines to which VLAN the data flow can be assigned, configures this VLAN in the device, computes new gate schedules with respect to this new traffic, and then updates the traffic scheduling accordingly.

3.2 Time synchronization configuration

The time synchronization for TSN networks has been specified in the standard IEEE 802.1AS [12]. In order to ensure an accurate time synchronization, IEEE 802.1AS imposes several constraints such as:

- There is only one time domain and one GrandMaster in the network.
- All networking devices (End stations and Bridges) shall be IEEE 802.1AS capable. As a consequence, all networking devices shall implement the Best Master Clock Algorithm (BMCA). This algorithm will ensure that there is only one GrandMaster to which the other networking devices are slaves.
- All networking devices shall allow to configure 15 parameters [3]: parameters related to GrandMaster selection (priority1 and priority2), parameters related to propagation delay calculation (e.g. delayAsymmetry, initialLogPdelayReqInterval, etc.), parameters related to time synchronization (e.g. initialLogAnnounceInterval, initialLogSyncInterval, etc.)
- The frequency with which IEEE 802.1AS messages are sent shall be the same in the network (e.g. all networking devices shall send the Sync message each 125 μs).
- The network shall not include more than 7 hops (clocks are synchronized up to 1 μs over 7 hops).

In networks that exhibit a strong requirement for low start-up time, it would be preferable not to support dynamic algorithms such as BMCA and clock spanning tree during network operation in order to reduce the startup timing system [2]. This leads to having more parameters to be configured in each device, as shown in Table 1 with respect to the AVnu specification for automotive TSN, thereby spending more effort on configuration.

Table 1: parameters to be configured for setting up the GrandMaster and clock spanning tree

IEEE 802.1AS	AVnu specification
priority1	portRole
priority2	isGM
	grandmasterIdentity
	asCapable

In case the network uses BMCA to select the GrandMaster, only priority1 and priority2 parameters are needed to be configured in devices. However, in the case of not using BMCA and the clock spanning tree algorithms, network engineer should configure in each device the following parameters: portRole, isGM, grandmasterIdentity and asCapable. More specifically, the boolean isGM is used to indicate whether the device is the GrandMaster. The grandmasterIdentity parameter indicates the GrandMaster identity when the isGM is set to false. As there is no clock spanning tree, network engineers should configure for each Ethernet port the parameters portRole and asCapable. In the portRole, they should indicate the role of the Ethernet port (i.e. Master/Slave). In the boolean asCapable, they should indicate whether the current device and the device at the other end of the link attached to this port can interoperate with each other via the IEEE 802.1AS protocol.

Consequently, according to AVnu specification, network engineers should decide which device will be the GrandMaster and the related clock spanning tree. Then, besides the other IEEE 802.1AS configurable parameters (i.e. parameters related to the frequency with which messages are sent), they should manually configure other devices with the four parameters that we cited beforehand. However, when the Grandmaster fails, network engineers should replace it with a new Grandmaster and, consequently, reconfigure all devices to consider the new GrandMaster.

With SDN solutions as we will show in the Section 5, the network reconfiguration will be an easy task. We need only to specify in the SDN controller the Grandmaster identity. Knowing the network topology, it computes the clock spanning tree to determine the portRole of each Ethernet port in each device. Then, it configures devices accordingly. For instance, in each device non-GrandMaster, it sets the isGM to false, configures grandmasterIdentity parameter, sets the role of each Ethernet port, and configures the frequency with which IEEE 802.1AS messages are sent.

3.3 Metrics monitoring

Because the offline tools are not able to provide thoroughly the entire run-time attributes. Metrics such as link propagation delay or packet residence time in the Bridge should be correctly estimated to avoid errors in the calculation of configuration settings. For that network, engineer should setup testbeds beforehand in order to correctly measure these metrics. Such task is time consuming and any error in the testbed setting impacts the final outputs of an offline configuration generator solution. Instead, in the SDN solution, the SDN controller can push a default configuration in the network (i.e. it could be the output of an offline tool). Then, it can monitor and measure, during network operation, the propagation delay of each link by setting the appropriate probes. Then, if the measured delay is different from what was considered in the default configuration, it will compute new configuration based on the changed values and reconfigure the network accordingly.

4. SDN ARCHITECTURE OVERVIEW

In this section, we provide an overview of SDN components and particularly we are focusing on NEON software, an SDN solution developed by CEA LIST that supports fast devices configuration and services deployment - within seconds - in dynamic and unconfigured infrastructures contexts

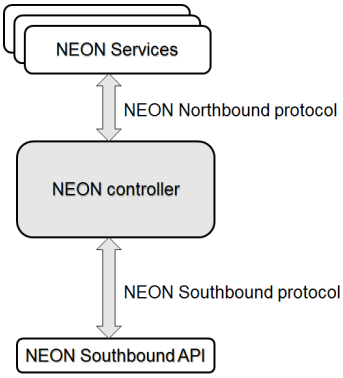

Figure 2: NEON software.

[6] [17]. Figure 2 shows NEON software decomposition. The communication between the different component uses commands in JSON format.

4.1 NEON southbound API

This module is responsible for adapting and configuring managed networking devices such as access points, routers, or gateways. It is an active module listening for commands sent by NEON controller or services. In addition, it is able to communicate with software and hardware resources in the managed network devices in order to configure them according to commands sent by the controller.

Moreover, this module detects any change in the networking device such as the activation of a new network interface and notifies NEON controller about it. In addition, NEON southbound API can provide NEON controller with statistics on several metrics such as the network bandwidth usage.

4.2 NEON controller

It is a logically centralized entity and represents the intelligence of the whole network. Beyond merely ensuring the routing related tasks, it must ensure all aspects of management and reliability of the network. The controller manages all managed networking devices via their NEON southbound protocol APIs.

On the other hand, the controller communicates with its different services via NEON northbound protocol. It can relay requests and replies coming from the services to the managed networking devices. In addition, the controller provides NEON services with an abstract view of the network topology (which may include statistics and events).

4.3 NEON services

They are presented as software running on top of the controller. Several network functions (e.g. security, routing, monitoring, ...) could run as NEON services; connected to the controller through a TCP/IP connection, they receive events such as a new device is added to the network and react accordingly. For example, they use some predefined NEON protocol action messages to configure interfaces in the data plane or to get specific statistics.

5. SDN SOLUTION FOR TSN NETWORK CONFIGURATION

In this section, we present the SDN solution that we de-

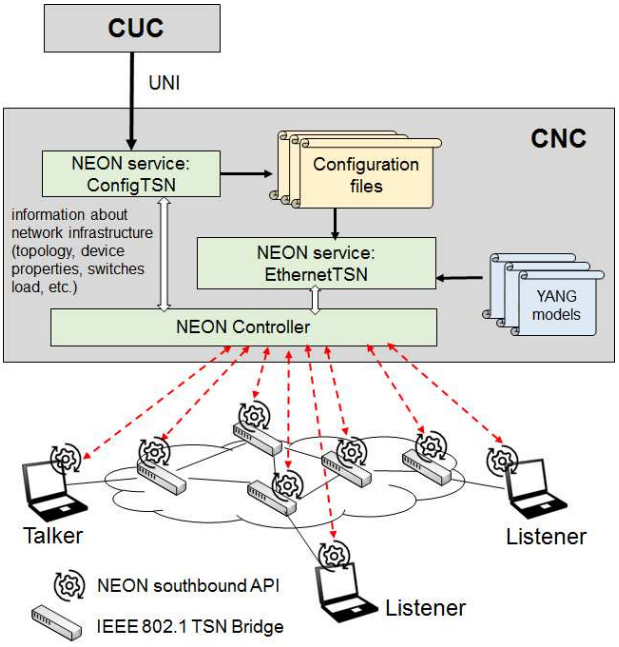

Figure 3: NEON integration in IEEE 802.1Qcc fully centralized model.

veloped for IEEE 802.1 TSN network configuration. First, we give an overview of the architecture. As SDN solution, we decided to use NEON software that we described in Section 4. Particularly, we focus on NEON services namely configTSN and EthernetTSN that we developed to ensure the adequate TSN configuration. Each time, we show the mapping between our solution and the IEEE 802.1Qcc fully centralized model. After that, we show how our solution is able to configure the time synchronization feature on each device in the network with respect to IEEE 802.1AS requirements.

5.1 Architecture Overview

The architecture of our SDN-based configuration solution is depicted in Figure 3. NEON controller with NEON services, particularly configTSN and EthernetTSN, can be mapped to the CNC entity in the IEEE 802.1Qcc fully centralized model.

- **NEON southbound API:** Devices in the network (Bridges and End stations) shall support NEON southbound API that will be responsible for receiving configuration information from NEON controller and applying it to themselves. We extended NEON southbound API with plugins that are responsible for generating the adequate commands to configure the device. First, at startup, the NEON southbound API connects to NEON controller. Then, it receives, from NEON controller, a configuration request that contains parameters with their configuration values. Depending on the vendor, the NEON southbound API forwards the configuration request to the adequate plugin that will be in charge of configuring the device with the received configuration values.
- **NEON controller:** It participates in TSN configu-

ration via two tasks. First, it is responsible for notifying EthernetTSN service whenever a device is up in the network and, then, relaying JSON request/reply between the service and the device. The second task of NEON controller is to notify ConfigTSN service with any change in the network (e.g. new device in the network, link/device failure, faulty device, overload situation, link delay, etc.). Following this notification, ConfigTSN service can determine a new configuration and update configuration files with new values. For example, NEON controller can command NEON southbound API to monitor the propagation delay of a specific link. In case the measured propagation delay is different from the default value that has been configured at the network startup, NEON southbound API notifies NEON controller. The latter notifies ConfigTSN service with the measured value.

- **ConfigTSN:** The role of this service is to produce the adequate configuration files for each device in the network. This service may start with a default configuration that was generated by an offline tool to prepare for each device its proper default configuration file. As in NEON solution, NEON southbound API generates and associates each device with a unique identifier (UUID), ConfigTSN associates the name of each configuration file with the UUID of the targeted device. Along with network operation, NEON controller relays any event (e.g. new device, new flow, link failure, the measured propagation delay, etc.) to the ConfigTSN service that will compute and generate new configuration files.

- **EthernetTSN:** The role of this service is to prepare the configuration requests to be sent to each device in the network. As the parameters to be configured in each device are vendor-specific, this service will be in charge of matching between the parameters that are presented in the configuration file generated by the ConfigTSN service and the YANG data model related to that device. When NEON controller detects the presence of a new device equipped with NEON southbound API, the controller informs EthernetTSN service and provides it with the UUID of this device. The service will look for the configuration file associated with this UUID. Then, it matches between the configuration file and the device YANG data model, prepares and sends to the device the appropriate configuration command.

5.2 Time synchronization deployment

The time synchronization is a key feature in TSN networks [12]. We developed a proof-of-concept where we used our solution to configure and deploy the time synchronization in a small network. The testbed is shown in Figure 4. It is composed of 2 End stations (Talker and Listener) connected via 2 TSN switches.

- **End station:** It is an embedded platform equipped with a Linux and an HP Intel I210-T1 Ethernet card [14]. The I210 chipset supports the TSN standards IEEE 802.1AS and IEEE 802.1Qav. We installed the software linuxptp [5] that allows managing IEEE 802.1AS.

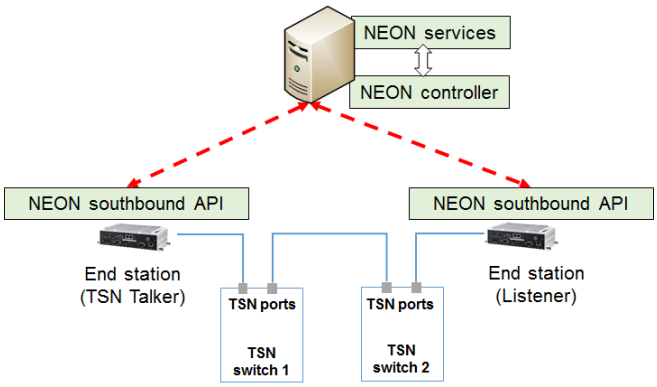

Figure 4: Time synchronization deployment proof-of-concept.

- **TSN switch:** It is produced by ANALOG DEVICES [1]. It is equipped with 2-ports (100 Base-Tx) TSN Ethernet module. The switch supports IEEE 802.1AS and IEEE 802.1Qbv standards. The configuration of the switch is done via a web interface.

We install NEON southbound API in each End station. NEON controller and the NEON services described above are installed in a standard Linux-based server that is connected to both end stations.

When the TSN switches are turned on, the BMCA is programmed to start automatically. The GrandMaster selection consists in comparing, one by one, between 5 attributes namely priority1, clockClass, clockAccuracy, offsetScaledLogVariance, priority2 and clockIdentity. Today, among these 5 attributes, the board offers the possibility to configure only priority1 and priority2. We configured in both switches to have the same values for priority 1 (246) and priority2 (248). The BMCA operation has led to selecting switch 1 to be the GrandMaster.

The ConfigTSN prepared the default configuration files for each end station. In the configuration file associated to the end station (talker), the priority1 is set to 246. However, in the configuration file of End station (listener), the priority1 is set to 133. According to the standard IEEE 802.1AS, priorities with lower values take precedence on priorities with higher values. Therefore, after running the configuration process, the end station (listener) should be chosen as the GrandMaster, as it has the lowest value of priority1.

In our configuration solution, we require the IEEE 802.1AS YANG data model. For that, we get inspired by the IEEE 802.1AS Management Information Base (MIB) to specify the appropriate YANG model. We published this model in the Internet draft [3].

In the End station (Talker) and End station (Listener), we start NEON southbound API programs with a UUID-1 and UUID-2, respectively. When NEON controller receives connection requests from these End stations, it notifies the EthernetTSN service. This latter will look for the configuration file related to each device and do the matching with the IEEE 802.1AS YANG data model [3]. Then, it prepares the adequate configuration command to be sent for each device. Upon receiving the configuration commands, NEON southbound API starts linuxptp software with the received configuration values. By having a look at the lin-

uxptp software output in the End station (talker), we can see that it indicates the End station (listener) identity as the Grandmaster identity. Moreover, we checked the time synchronization on the web interface of each TSN switch. Both of these web interfaces indicate that the end station (listener) has been selected to be GrandMaster.

5.3 Discussion and next steps

Through the example of time synchronization deployment, we showed that our solution is able to manage (i.e. start/stop, configure/reconfigure) on-the-fly the time synchronization module in End stations. As End stations are Linux-based platforms and linuxptp is an open software, it was possible to develop a plugin in NEON southbound API that can easily manage linuxptp. Conversely, the web interfaces were the only way to configure the TSN switches that we are using. Although we are able to create a plugin in NEON southbound API to configure these switches via HTTP requests, this is still not the appropriate manner to ensure the automatic configuration that we are targeting. Ideally, the TSN switch vendors shall expose monitoring and configuration APIs of their devices and provide their data model (YANG, MIB or JSON) as well. Nowadays, TSN switch vendors are aware of this critical issue and some of them are working towards exposing APIs of their devices [18].

For the future work, there are many directions. From implementation aspects, the current version of ConfigTSN service needs to be enhanced. Today's, it does not react to the notifications received from NEON controller about changes in the network topology. The evolution of this service will consist in implementing the required mechanisms to consider these notifications and react accordingly. For example, when the GrandMaster disappears from the network (i.e. device failure), ConfigTSN should designate a device from the network to be the GrandMaster, generate new configuration files and trigger the EthernetTSN service to push this configuration in the network. Moreover, from theoretical aspects, integrating the artificial intelligence in our solution, in order to compute the adequate traffic scheduling for the network, will be considered.

6. CONCLUSION

In this paper, we highlighted the need for SDN solutions to ensure an easy and automatic (re-)configuration of industrial and automotive networks. Particularly, we proposed an SDN-based network configuration solution based on the use of NEON software, an SDN solution developed by CEA-LIST that ensure the dynamic configuration of networks. A first proof-of-concept has been developed where we showed how our solution enables an automatic configuration of the time synchronization feature in the network.

7. REFERENCES

[1] ANALOG DEVICES. TSN evaluation kit. http://www.innovasic.com/products/tsn-kit. [Online; accessed 16-April-2018].

[2] AVnu Alliance. Automotive Ethernet AVB Functional and Interoperability Specification, Revision 1.5. specification, 2016.

[3] S. Ben Hadj Said and M. Boc. YANG Model of IEEE 802.1AS. Technical Report draft-benhadjsaid-detnet-gptp-yang-00, 2018.

[4] D. Bruckne, R. Blair, M.-P. Stanica, A. Ademaj, W. Skeffington, D. Kutscher, S. Schriege, R. Wilmes, K. Wachswender, L. Leurs, M. Seewald, R. Hummen, E.-C. Liu, and S. Ravikumar. OPC UA TSN A new Solution for Industrial Communication.

[5] R. Cochran. The Linux PTP project. http://linuxptp.sourceforge.net/. [Online; accessed 13-April-2018].

[6] S. Decremps, S. Imadali, and M. Boc. Fast Deployment of Services in SDN-based Networks: The Case of Proxy Mobile IPv6. *Procedia Computer Science*, 40:100 – 107, 2014. MoWNet'2014.

[7] J. L. Du and M. Herlich. Software-defined networking for Real-time Ethernet. In *ICINCO (2)*, pages 584–589, 2016.

[8] J. Farkas, B. Varga, R. Cummings, Y. Jiang, and Y. Zha. DetNet Flow Information Model. Technical Report draft-ietf-detnet-flow-information-model-01, 2018.

[9] A. Gopalakrishnan. Applications of software defined networks in industrial automation, 2014.

[10] M. Gutiérrez, A. Ademaj, W. Steiner, R. Dobrin, and S. Punnekkat. Self-configuration of IEEE 802.1 TSN networks. In *Emerging Technologies and Factory Automation (ETFA), 2017 22nd IEEE International Conference on*, pages 1–8. IEEE, 2017.

[11] M. Herlich, J. L. Du, F. Schörghofer, and P. Dorfinger. Proof-of-concept for a software-defined real-time ethernet. In *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*, pages 1–4. IEEE, 2016.

[12] IEEE. Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks. IEEE P802.1AS, 2011.

[13] IEEE. Bridges and bridged networks amendment: Stream reservation protocol (SRP) enhancements and performance improvements. IEEE P802.1Qcc/D2.1, 2018.

[14] Intel. Intel Ethernet Controller I210 Datasheet. https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/i210-ethernet-controller-datasheet.pdf. [Online; accessed 16-April-2018].

[15] G. Kálmán. Mass configuration of network devices in industrial environments. In *The Twelfth International Conference on Networks (ICN 2013)*, pages 107–111. IEEE, 2013.

[16] G. Kálmán. Applicability of software defined networking in industrial ethernet. In *Telecommunications Forum Telfor (TELFOR), 2014 22nd*, pages 340–343. IEEE, 2014.

[17] M. Labraoui, M. Boc, and A. Fladenmuller. Self-configuration mechanisms for SDN deployment in Wireless Mesh Networks. In *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–4, June 2017.

[18] TTTech. DE IP Solution Edge. https://www.tttech.com/products/industrial/deterministic-networking/fpga-asic/de-ip-solution-edge/. [Online; accessed 16-April-2018].