IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON ADVANCES ON HIGH PERFORMANCE WIRELESS NETWORKS FOR AUTOMATION AND IIOT

Received 6 July 2023, accepted 1 August 2023, date of publication 4 August 2023, date of current version 10 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3302264

RESEARCH ARTICLE

AutomAdapt: Zero Touch Configuration of 5G QoS Flows Extended for Time-Sensitive Networking

FRANCISCO LUQUE-SCHEMPP, LAURA PANIZO, MARÍA-DEL-MAR GALLARDO, AND PEDRO MERINO

ITIS Software Institute, Universidad de Málaga, 29010 Málaga, Spain

Corresponding author: Francisco Luque-Schempp (schempp@uma.es)

This work was supported in part by the EVOLVED5G Project (European Union Horizon 2020) under Grant 101016608; in part by the 5G+TACTILE Project (NEXTGENERATION.UE, Spanish UNICO 5G I+D) under Grant TSI-063000-2021-11; and in part by the RFOG Project (Spanish Government) under Grant RTI2018-099777-B-I00.

ABSTRACT The aim of IEEE Time-Sensitive Networking (TSN) standards is to grant deterministic communication in traditional Ethernet networks for Industry 4.0. Insofar as the use cases in the Factory need some mobility, the extension of the TSN capabilities over the fifth-generation (5G) cellular network is the next step. Some challenges in TSN over 5G, such as TSN translators time synchronization functionality, are well defined in the standards, even if they have not yet been addressed in the market. However other challenges, such as the dynamic configuration of the entire network (or part of the it) based on quality requirements of the current TSN traffic pattern, are defined at a very high level and delegated to vendors for implementation. This paper addresses this challenge, using an Automata Learning approach to monitor and reconfigure the end-to-end 5G QoS flow to keep the quality of a TSN session within the required values. Additionally, algorithms are provided to build the automata from network data and predict potential deviations of the requirements to meet the expected quality. Moreover, this work presents a functional TSN over a 5G testbed where the algorithms have been tested, demonstrating that the proposed solution achieves an improvement of around 40% compared to the usual operation of the network.

INDEX TERMS Zero touch configuration, automata learning, time-sensitive networking, 5G.

I. INTRODUCTION

The competitiveness in the industry is vastly increasing due to the interest in obtaining products with a certain quality and reliability while maintaining cost-effectiveness. The *Industrial Automation* concept is progressively becoming relevant for the Industry 4.0 and numerous emerging use cases are coming across new challenges that need to be solved. Protocols and standards, such as PROFINET or EtherCAT, are implemented to complement Ethernet networks, providing reliable communications between devices. Moreover, Time-Sensitive Networking (TSN) is a series of standards developed by the IEEE 802.1 Working Group to implement deterministic connectivity (bounded latency, low jitter and low packet loss) over traditional Ethernet networks. Features such as time synchronization and scheduled traffic, among others, are two pillars to guarantee the Quality of Service (QoS) for these stringent communications. These mechanisms fulfill the communication requirements in wired environments. The combination of the above-mentioned standards with wireless communication technologies allows industrial networks to obtain the benefits associated with these technologies. A clear example of this is *mobility*, allowing the user to move from one place to another. These wireless technologies could be used as an alternative to replacing cables in the industry. However, currently it is not possible to replace the wired solutions in industrial environments altogether due to the stringent requirements of the applications. This results in a combination of both wired and wireless technologies, called hybrid networks.

The associate editor coordinating the review of this manuscript and approving it for publication was Gianluca Cena.

In the case of TSN networks, there are two wireless technologies leading progress toward hybrid TSN networks. On the one hand, the IEEE 802.11 standard provides the information related to wireless local area networks (WLANs), and has been proposed as a candidate to implement wireless TSN in [1], since it supports precise clock synchronization and moderately time-aware scheduling. Additionally, the authors have identified the challenges for supporting TSN in Wi-Fi 7 as well as some low latency use cases (e.g. process automation). Although 802.11 is a promising enabling technology, it can not meet all the requirements to guarantee deterministic communications, mainly because it does not provide protection against interference from other users or radio services operating in the same frequency band, since the spectrum is unlicensed.

On the other hand, 5G is the main candidate for wireless TSN implementation. 5G networks have emerged as an alternative to Wi-Fi in industrial environments to increase the quality with reserved spectrums, thus the definition of TSN over 5G seems to be the next natural step. Techniques and procedures related to TSN over 5G integration are detailed in [2]. Other works also tackle this integration from different perspectives, such as optimizing the end-to-end delay performance [3] or the implication on the multi-scheduling of end-to-end traffic flows [4]. In addition, industry has pushed the standardization bodies to define the whole set components to achieve TSN over 5G in specifications like [5] and [6]. These 3GPP documents introduce the concept of *translators*, the process to manage *time synchronization* as well as the general role of a new entity in the 5G network to ensure the quality in the end-to-end TSN traffic session, the *TSN Application Function (TSN AF)*. This paper focuses on the implementation of a smart TSN AF that covers the features not fully defined in the standards and is delegated to vendors for their implementation.

The problem addressed is how to *dynamically* guarantee the expected quality for the end-to-end TSN session in a fully manageable and controllable private 5G network deployed in an industrial plant. Our initial hypothesis, confirmed by previous experimental work on our 5G testbed [7], is that once the entire 5G network has been configured and the specific parameters of a given device are connected to the network, the values of Key Performance indicators (KPIs), such as latency, jitter, packet loss, etc., exhibit a high variability during a TSN session. These variations are due to internal implementation aspects, such as the size of the buffers that could provoke packet loss from time to time, changes in the location of the device resulting from its work plan, or the presence of more devices in the area. Although 5G defines some methods to adjust parameters, such as transmission power when coverage is poor, they are not always sufficient to meet TSN traffic requirements. In addition, the previous work also confirmed that the network exhibits a deterministic behavior, obtaining the same behavior under the same conditions (e.g. number of devices, network configurations, mobility, etc.). It is worth noting that in this context, a private network is considered within the limited area of a factory, where most of the devices connected to the network have well-defined work plans and the network conditions can be controlled to achieve the mentioned network behavior.

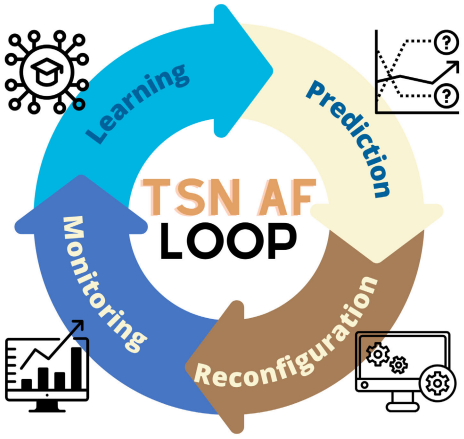

FIGURE 1. Application function loop.

Our proposal, already introduced at a high level in [7], consists of building a smart TSN AF that implements a closed loop for *Zero Touch Configuration*, which is represented in Figure 1. This loop includes the network monitoring and the reconfiguration of the network parameters that have an impact on the expected quality of the specific end-to-end session, called *5G QoS flow*, for the devices that support the TSN session.

Following the previous work, this paper shows how to increase the quality of the TSN session by dynamic reconfiguration parameters of the 5G QoS flow, like, for instance, the values of timers at the Radio Link Control (RLC) layer (more parameters are presented in Section III). The Zero Touch Configuration is implemented by using learning techniques in the area of *Automata Learning* [8], [9], [10] to be able to predict potential deviations of the KPIs and to reconfigure the end-to-end 5G QoS flow in order to avoid such deviations. More specifically, the contributions of this paper with respect to previous work presented in [7] are as follows:

1) The definition of a new kind of automata, called *Traffic Oriented input/output Automata* (TOA), to represent the evolution of the TSN session in relation to time, including information of the 5G QoS flow configuration, the device location and context, the values of the KPIs and relevant events like network configuration actions. This definition of automaton is a time-oriented extension of the previous proposal that is also richer in terms of information.
2) The improvement of the Automata Learning algorithm already defined in order to generate TOA based on real TSN network behavior over 5G from network trace monitoring.
3) The design of a first prediction and reconfiguration algorithm that exploits the TOA for a particular

scenario where it is assumed that the TSN devices follow the same work plan and use the same traffic pattern already learned by the TOA.
4) A TSN over 5G testbed that solves the main low-level aspects like time synchronization and TSN translators in charge of translation between TSN and 5G domains.
5) A full implementation of the new algorithms in the tool called *AutomAdapt* and its evaluation on top of the TSN over 5G testbed.

This paper reports a novel method and a tool for Zero Touch Configuration of 5G QoS flows based on learning methods that can be integrated in a TSN AF.

The rest of the paper is organized as follows. Section II describes related work in the areas of learning techniques for communication networks, Zero Touch Management of 5G networks and TSN implementations. Section III introduces the context of our research, describing the TSN over 5G technologies and the specific problem of (dynamically) mapping the expected quality of the TSN traffic in the 5G QoS flow configuration. Section IV presents our Automata Learning-based Zero Touch approach, including algorithms for learning and prediction. Then, Section V describes the implementation of a TSN over 5G testbed that includes all the relevant features defined in the 3GPP standards as well as our algorithms to complete a functional TSN Application Function. Finally, the conclusions and future work are presented in Section VI.

II. RELATED WORK

This section presents the three main topics related to our main contribution. First, the use of *learning techniques* to create an automaton that represents the network behavior. Second, the implementation of a *Zero-Touch* configuration mechanism to predict deviations and reconfigure the 5G network dynamically. Third, the integration of this approach into the *TSN over 5G testbed* at the University of Malaga.

A. LEARNING NETWORK TRACES

In the literature, there are two main approaches to learning and constructing models of interactions based on observations that can be used to learn on network traces. On the one hand, traditional methods of Artificial Intelligence (AI) based on machine learning such as reinforcement learning [11] and learning automata [12], and on the other hand, the automata learning approaches [8], [9] studied in the context of Formal Methods. It is worth noting the confusing terminology used to name the AI technique (learning automata) and the formal method learning technology (automata learning).

Reinforcement learning and learning automata are well-known AI techniques to learn from observed interactions. Both techniques learn how to map the environment state into actions that maximize the so-called reward. Algorithms perform an iterative process that ends when the optimal reward has been reached. This reward could be, for instance, the system's performance or the use of certain resources. These techniques have been used to optimize, for example, the configuration of wireless networks [13], [14]. It is worth mentioning that in reinforcement learning and learning automata, algorithms are aware of the environment situation (state) during their lifetime. However, the intermediate environment states are not explicitly shown in the learned model; instead they are hidden in the calculated probabilities. For this reason, it is difficult to find an explanation for a reconfiguration decision made.

In contrast, automata learning techniques try to construct a state machine model from a black-box system. There are two main approaches, namely active and passive learning. Active learning is based on queries to the underlying system that guide the construction of the automata. For instance, active automata learning has been used to construct a learned model of the Non-Access Stratum (NAS) layer in 4G networks [15] and the state machine for the IEEE 802.11 4-Way Handshake [16]. Passive learning techniques only use the observed system behavior without explicitly interacting with it. One example is the use of reverse engineering to obtain the state machines for FTP and SMTP protocols in [17] and [18]. Compared with traditional AI techniques, automata learning techniques produce a formal model of the system whose relation with the original system may be formally analyzed. In addition, any decision made on the original model by exploring the formal model can be mathematically explained. Therefore, in this paper the automata learning approach is followed to represent the evolution of the whole TSN over 5G system. In this way, the reasoning behind each decision can be explained and understood by humans.

B. ZERO-TOUCH NETWORKS APPROACH

The complex task of efficiently managing a network to meet all requirements makes approaches such as *Zero-Touch Network Management* (ZTM) and *Zero-Touch Network and Service Management* (ZSM) a necessity. In particular, for technologies such as 5G and TSN that have applications with demanding requirements that are difficult to manage. Moreover, since our work focuses on meeting the expected quality of end-to-end connections, network automation must take into account all network pieces (i.e. Radio Access Network, 5G core network, User Equipment, and TSN endpoints).

In the current literature, there are works that consider the different segments of the network separately. In the case of Radio Access Network (RAN), there exists a lot of proposals and solutions. On the one hand, diverse learning techniques are used to operate and manage the resources. The authors in [19] address the problem of optimizing resource allocation over time. For this purpose, they introduce an online learning algorithm and perform experiments via simulation to evaluate its performance. In [20], the authors identify the different factors affecting the Channel State Information (CSI) (e.g. frequency band, location, and weather), which is a essential concept to know the radio link quality. They propose a learning framework to predict CSI in 5G networks.

The aforementioned concepts and others related to the radio configuration (e.g. modulation and coding scheme - MCS) and link quality measurement (e.g. received signal strength indicator - RSSI) are important in terms of QoS assurance. However, they alone are not sufficient for demanding applications. On the other hand, the use of closed control loops (CCLs) to adequately provide services with minimum human intervention has been used for years. The authors in [21] use an automated CCL to configure and deploy the service, capturing its relevant management information and the objective. This paper follows a similar strategy, since the expected requirements of the application to configure the network are considered at the beginning of the connection. Furthermore, the CCLs are key enablers for Zero-Touch approaches. The authors of [22] address the automation aspects of multi-domain environments using several CCLs for end-to-end service management.

Nonetheless, an integrated approach of the network has to be considered to deal with end-to-end connectivity requirements. Other approaches exist to avoid manual configurations at deployment time and enable the flexibility and programmability of converged networks. Additionally, they pursue dynamic configuration and troubleshooting during service operation, which is fundamental for these kinds of applications with stringent requirements. First, Self-Organizing Networks (SON) [23] was originally developed to optimize the RAN deployment. However, new capabilities (e.g. self-learning) have been added to extend the scope of SON, including different network segments such as core and transport network. For example, the authors in [24] propose the use of big data to address the stringent 5G requirements (for example, ultra-low latency) with the aim of creating an end-to-end smart network. They also identify the useful network information at different levels that can be exploit for self-optimization. Some of the information that the authors have identified (for example, packet jitter, delay, throughput, drop rate, etc.) is used in our work to learn and subsequently make predictions in order to reconfigure the network when necessary. Second, the concept of network slicing allows to create isolated networks tailored to fulfill diverse requirements on top of a common physical infrastructure. Similarly to SON, this concept can cover one or more network segments. The paper in [25] studies network slicing as the sharing of uplink RAN resources for the main types of 5G traffic. On the contrary, the work in [26] present an end-to-end perspective of network slicing, considering the slicing in the User Equipment (UE), the transport network and the core network; in addition to the RAN. Additionally, several practical examples are presented to support this approach. The combination of these concepts and approaches can be found in the literature. For instance, the use of CCLs to support the 5G network slicing [27] and the application of learning techniques to optimize the RAN resources [28], [29].

Our work focuses on the Zero-Touch approach to implement the dynamic reconfiguration of the 5G QoS flow based on the current traffic requirements, similar to the ideas presented above. Moreover, a tool named AutomAdapt has been designed and implemented, in which a CCL and learning techniques are combined to perform this task. The work cycle of this loop is explained in detail in Section IV.

C. INTEGRATION OF TSN WITH 5G

The actual implementation of TSN over 5G is an active field which still has few testbeds or experimental implementations.

The first proposals to evaluate TSN over 5G are coming from the simulation community. In [30] the authors use the OMNeT++ simulator and the TSN model NeSTiNg to evaluate the integration of 5G into TSN as a transparent bridge. They add a new model of the end-to-end data path to start evaluating different configurations of the end-to-end path. Their model plays the role of our 5G QoS Flow extended for TSN (including the translators described in the next Section).

Regarding physical testbeds, the work in [31] reports a testbed to reproduce the idea of full synchronization using IEEE standards for TSN. The main requirement here is to allow the UE to access some timing information in the 5G radio channels in combination with timing information from the TSN domain. Unfortunately, they only use 4G networks due to instability of the 5G version's OpenAirInterface [32] software used for the wireless segment.

The work in [33] presents a testbed with 3GPP Release 15 hardware to evaluate an industrial control system. They also discuss the configuration of TSN over 5G systems and methodologies needed to characterize the Quality of Experience (QoE) for industrial use cases. Insofar as Release 15 does not support features needed for TSN, the authors extend the basic 5G network with emulation of Ethernet PDU sessions, Ethernet TSN capable devices and synchronization with explicit wired connection of both sides. This is the closest testbed to our development. This paper shares with them the use of 5G Release 15 and TSN devices. However, the wired synchronization is replaced with GPS-based clocks that allow more practical wireless deployments. Additionally, the actual commercial gNB with Stand Alone 5G is used instead of their NSA mode based on the Keysight 5G emulator. And more importantly, this work includes intelligent reconfiguration of the network during the TSN session instead of keeping a fixed configuration.

All previous works follow the same principle as our paper's, namely the extension of Ethernet-based TSN to the wireless domain using 5G. However, there is a another approach followed by the OPC foundation that combines TSN with their application layer protocol [34]. The recent survey [35] summarizes the state of the art in the integration of TSN and 5G. The integration of TSN over 5G is presented in Section V-A, which is implemented on the testbed available at the University of Malaga.

III. TSN OVER 5G

The main motivation of this work is to attain Zero Touch Configuration for TSN over 5G. Therefore, this section provides

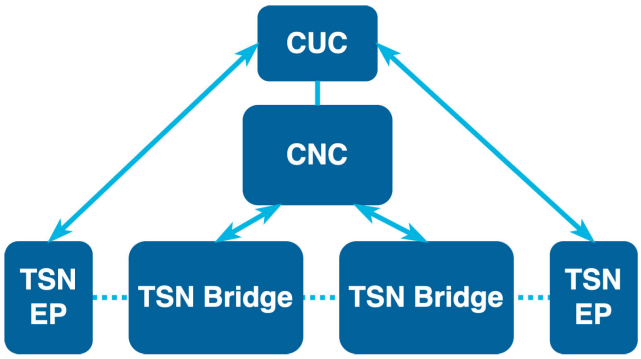

FIGURE 2. Fully centralized TSN network architecture - IEEE 802.1Qcc [37].

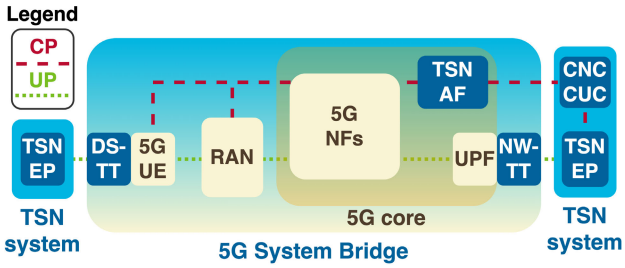

FIGURE 3. TSN over 5G network architecture.

an overview of TSN over 5G and introduces the main challenges, entities, and concepts mentioned in the rest of the paper.

The name Time-Sensitive Networking mainly refers to the standards defined by IEEE to expand Ethernet for Time-Sensitive communications [36]. In the wired scenario, the network interfaces of the final devices, called *TSN endpoints* (TSN EP), and the TSN switches deployed in the network support specific methods for sharing time information and ensuring the requirements of each *traffic pattern* sent by the endpoints. In addition, these standards state that the QoS requirements should be communicated in advance. In the case of the fully centralized TSN network architecture, depicted in Figure 2, the QoS requirements are communicated from TSN endpoints to the *Centralized User Configuration* (CUC). Then, CUC exchanges this information with the *Centralized Network Configuration* (CNC) which manages the configuration of the TSN Bridges on behalf of the TSN endpoints.

The benefits of removing cables in factories pushed 3GPP to define in Release 16 [5] an architecture where the whole 5G network, called *5G System* (5GS), can act as a transparent TSN bridge, here called *5GS Bridge*. Figure 3 shows the TSN over a 5G architecture. The 5G network is composed of the 5G core, the Radio Access Network (RAN), and the User Equipment (UE). The usual components of the 5G core have a relevant role to get the 5GS Bridge working, but they basically follow the 3GPP standards. For instance, User Plane Function (UPF) and other 5G Network Functions (NFs) are participating in many procedures to support TSN over 5G. As mentioned in the introduction, this architecture includes two new types of entities that are specialized to support TSN features over 5G, the TSN translators (DS-TT and NW-TT) and TSN AF.

Full commercial or academic TSN over 5G solutions are not yet widely available due to some open challenges. On the one hand, the TSN translators defined in recent standard releases by 3GPP (e.g. Rel. 16 or Rel. 17) are still relatively recent. In addition, they have not been tested in real scenarios by vendors. This may be due to the limited demand that currently exists, coupled with the development complexity. On the other hand, there are issues related to the complexity of implementing a general TSN AF module that, according to 3GPP, must be open to algorithms defined by network vendors. The rest of the paper is mainly focused on this topic, since our goal is to include our intelligent loop, called *AutomAdapt*, as part of the TSN AF functions, which will be thoroughly explained in Section IV.

A. STANDARDIZED FEATURES

The *TSN translators* (TTs), represented in Figure 3 at the edges of the 5G System Bridge, are responsible for ensuring interoperability of the user and control planes between the 5G and TSN networks, transforming the 5G network into a 5GS Bridge (logical TSN bridge) and achieving the aforementioned transparency for the users. In the most common scenario, the TSN endpoint connects to the 5G wireless access (5G UE) using *Device-Side TSN Translator* (DS-TT), while the wired part of the application is connected to the 5G network (UPF) using the *Network-Side TSN Translator* (NW-TT).

One of the key features of the TSN over 5G networks is *time synchronization*, which allows having a unique time reference through both TSN and 5G domains, which is indispensable for Time-Sensitive Communications (TSC). The TSN translators fulfill all functions related to time synchronization. Since 3GPP Release 16 onwards, the Precision Time Protocol (PTP) has been proposed to achieve this synchronization. However, current 5G devices (e.g. smartphones and modems) and network components like the gNBs do not yet support this feature. A transitory solution is proposed in Section V-A. Another key feature is the traffic translation between TSN and 5G network, which includes the support for TSN sessions; for instance, the QoS mapping from incoming TSN traffic, the buffering mechanism, the PDU session and TSN translator port connection (the relationship information is stored by TSN AF).

B. THE PARTIALLY-STANDARDIZED TSN APPLICATION FUNCTION

The *TSN Application Function* (TSN AF), which is located within the 5G core in Figure 3, is the entity in charge of the control and configuration of the 5GS Bridge based on the traffic pattern declarations coming from the TSN endpoints through CUC and CNC. In addition, the TSN AF stores the information of the connection related to the TSN translator

ports and Packet Data Unit (PDU) session, such as relationships, stream filters, QoS mapping, time synchronization, etc. Some aspects of the TSN AF are clearly defined in the standards, such as 5GS Bridge management, information exchange with TSN translators, and interaction with CNC for configuration. However, most of the details on how to expose and implement this function are vendor specific and not standardized. The main concepts managed by an advanced TSN AF, which will be used in the rest of the paper, are the following:

- A *TSN session* refers to the connection of a mobile TSN endpoint to interact with a fixed TSN endpoint. For example, when a human operator with a remote controller connects to drive a robot that performs a given task.
- The *TSN traffic pattern* defines the kind of traffic sent in the TSN session. The traffic pattern is communicated by the TSN endpoints to the TSN AF through the CUC and CNC. For example, the traffic pattern can define cyclic-synchronous traffic with a typical period of 1 ms and 1000 bytes of payload (more examples are presented in Table 2).
- The *expected KPIs* refers to the acceptable values for a number of metrics for a TSN traffic pattern. An acceptable delay and jitter for a TSN industrial application running over 5G should be under the typical period, for instance, [6-10] ms and [0-1] ms respectively (see Table 1 for more examples).
- The *5G QoS flow configuration* refers to the values of all the configurable parameters in the 5G network for a given TSN session (without affecting other sessions). Examples of these configurable parameters include the RLC mode (AM, TM, UM), the timers in AM mode, the buffer discard threshold in MAC layer, and the reordering timer in PDCP layer.
- The *session trace* refers to the sequence of observed states of the network and the connected devices in a specific TSN session. Each state includes the configurable parameters, the state of the devices (e.g. the location), and the actual observed values of the KPIs; for instance, a mobile device connected to the network while the location is changing. Therefore, the network configuration is being adapted and, in turn, the session trace has a number of different observed states.

The main goal of the TSN AF is to configure each **5G QoS flow** dynamically in order to offer the **expected KPIs** to support the specific **TSN traffic patterns** in a given **TSN session**. Meanwhile, the **session trace** containing all the information related to the network states (e.g. network parameter values, current KPIs values, etc.) is generated. The connection between the TSN AF and other components is crucial to attain said objective. On the one hand, the connection of the TSN AF and certain 5G core components, called Network Functions (NFs), is involved in the establishment of TSN QoS Flows. The TSN AF and other NFs are responsible for storing the information related to the mapping between TSN QoS profiles and 5GS QoS profiles. Regarding the 5G QoS profiles, it is possible to use the standardized 5QIs (QoS Identifier), define pre-configured 5QIs or dynamically assign different 5QIs. Then, the PDU Session together with the 5G QoS Flows needed can be created. On the other hand, the TSN AF is also connected with the CNC for the time synchronization service configuration. The PTP functionalities supported and configured by TSN translators are also part of the TSN AF functioning.

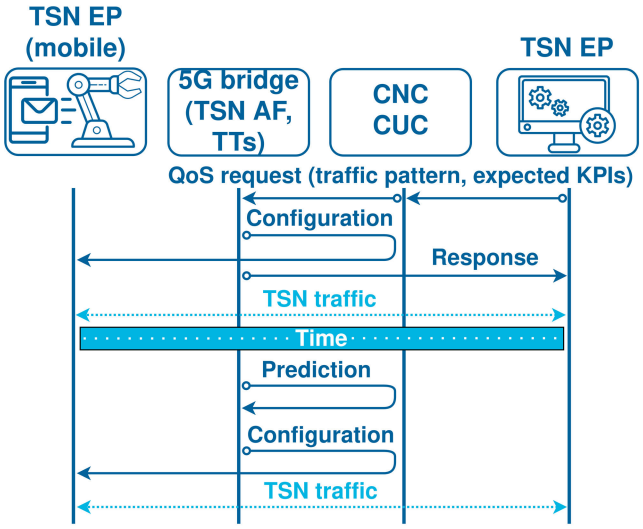

FIGURE 4. Example MSC of TSN over 5G.

Figure 4 depicts a Message Sequence Chart (MSC) of the system behavior including the most significant iterations. First, the TSN endpoint asks TSN AF (through CUC/CNC) for a certain QoS for a TSN session, indicating in advance the traffic pattern together with the expected KPIs. Then, the TSN AF and the CNC apply the necessary configuration to the 5G network and the TSN translators, respectively. Once the network configuration is adjusted to meet the traffic requirements, the TSN AF notifies the TSN endpoints that TSN traffic can be sent and received. This notification is performed through an RRC Reconfiguration message which includes the new values of the network parameters. An example of this is presented in Section V. During the TSN session, if the TSN AF detects a deviation of the observed KPIs, it can perform a reconfiguration of the network in order to fulfill the traffic requirements.

IV. AUTOMATA LEARNING APPROACH

A. OVERALL APPROACH

In order to make the TSN AF smart enough to dynamically carry out the required changes in the 5G QoS flow configuration, a closed loop with four main phases has been defined, like the one shown in Figure 5, as part of the TSN AF module. Once the TSN AF has made the first 5GS bridge configuration and the actual traffic starts from one TSN endpoint to the other, the process works as follows.

The *monitoring* module captures snapshots of the whole TSN session including the 5G QoS flow configuration and the actual values of the KPIs as *concrete network states*. These states conform the network traces that are used as input to the *learning* module to produce an automaton that represents the network behavior. In Section IV-B, the so-called *Traffic Oriented input/output Automata* (TOA) is formally defined as the underlying formalism of learned automata. Since automata are finite state machines, the TOA states are abstract and finite representations of the concrete network states. The *prediction* module uses the learned automaton to predict if the observed behavior of the real network will lead to an undesirable state (wrt the TSN and 5G QoS profiles) and proposes in advance an alternative network configuration, if possible. Finally, the *reconfiguration* module applies a new configuration to the 5G QoS flow, when required.

The whole process of monitoring, prediction and reconfiguration should be done in parallel with the actual TSN session, without negative impact on the KPIs. The learning process to build the TOA can be done in different ways. One option is just to run a number of trials to produce real session traces to build the TOA and use it later in the closed loop process. Another option is to build or to extend the TOA within the loop.

As described above, the network produces *concrete states*, composed of the current configuration and the observed KPIs, which must be mapped into simpler versions of network states, called *abstract states*, to be part of the TOA. Abstraction of concrete states is essential to produce finite and manageable TOA able to be analyzed during the prediction phase. In consequence, a major decision for the construction of the TOA is how to carry out the abstraction of concrete states. On the one hand, since the number of different 5G QoS flow configurations defined by 3GPP standards is finite, they can directly be part of the abstract states. However, actual values of the KPIs will make the TOA unfeasible unless some abstraction procedure is applied to them. The *qty* function, detailed below, is used to compact sets of values for all the observed KPIs to single values to be used for quick comparison with respect to the ideal or acceptable range of expected KPIs in the TSN session. In a practical scenario, this function will be defined by the network operator, when installing or upgrading the TSN AF module, and will provide quality ranks for the observed KPIs. For instance, Table 1 shows a realistic definition of *qty* for a specific TSN traffic pattern (cyclic-asynchronous traffic with a typical period of 10 ms and 1000 bytes of payload) and their observed KPIs. In this example, the positive values of *qty* represent valid thresholds defined by the operator for the mentioned traffic pattern. Conversely, the negative values denote at least one invalid threshold of the observed KPIs for this traffic pattern.

In the next sections, the whole process for learning TOA from network traces along with the prediction module is formally described. The prediction module makes use of the learned TOA to anticipate the non desirable quality deviation of the KPIs and suggest configuration changes when possible.

TABLE 1. Example of *qty* function values for a given traffic pattern.

qty	*delay* (ms)	*jitter* (ms)
-18	[10-12]	[8-10]
-8	[10-12]	[1-5]
-1	[0-7]	[8-10]
1	[0-7]	[0-1]
2	[0-7]	[1-5]
12	[7-10]	[0-1]
13	[7-10]	[1-5]

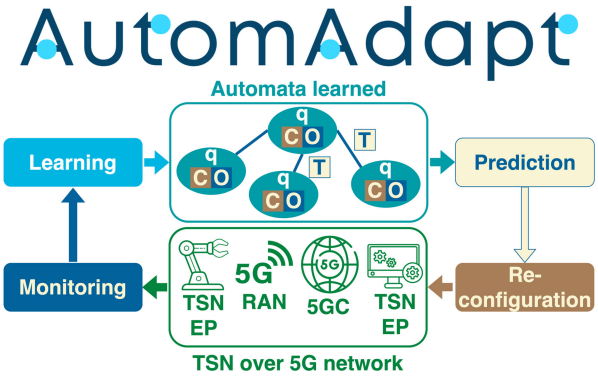

FIGURE 5. AutomAdapt tool: a Zero Touch Configuration closed loop.

However, when no reconfiguration can be suggested with the available information, the network operator will receive warnings in order to try to fix the issues manually. Such a manual configuration and its effect could be integrated into the TOA for further decisions.

B. TRAFFIC ORIENTED INPUT/OUTPUT AUTOMATA

In this section, the notion of Traffic Oriented input/output Automata (TOA) is introduced as the specific formalism that captures the behavior of the network $\mathcal{N}$ to be learned.

Let $\mathcal{C}$ be the set of all possible network configurations with $conf_d \in \mathcal{C}$ being the initial default configuration. It is assumed that $KPI = \{kpi_1, \cdots, kpi_k\}$ is the set of the k network KPIs to be monitored, and are represented as vectors $\bar{o} = \langle o_1, \cdots, o_k \rangle \in \mathbb{R}^k$ where each o_i corresponds to the value of KPI kpi_i. The symbol $\perp$ is used to represent that KPI values are unknown.

Definition 1: *Network states* ($\mathcal{N}$-states) are defined as pairs of the form $(conf, \bar{o}) \in \mathcal{C} \times (\mathbb{R}^k \cup \{\perp\})$, *conf* and $\bar{o}$ being a network configuration and a vector with the KPIs values (or $\perp$, if they are unknown), respectively. A *network trace* (or $\mathcal{N}$-trace) is a finite sequence of $\mathcal{N}$-states as $(conf_0, \bar{o}_0) \cdots (conf_n, \bar{o}_n)$.

Network states *observed* during a network session have an additional timestamp attached that records the time instant when the state has been registered.

Definition 2: *Observed network states* (observed $\mathcal{N}$-states) are defined as 3-tuples of the form $\langle ts, conf, \bar{o} \rangle$, where $ts \in \mathbb{R}^{\geq 0}$ is the timestamp component and $(conf, \bar{o}) \in \mathcal{C} \times (\mathbb{R}^k \cup \{\perp\})$ is an $\mathcal{N}$-state. An *observed network trace*

(or observed $\mathcal{N}$-trace) $\langle ts_0, conf_0, \bar{o}_0\rangle \cdots \langle ts_n, conf_n, \bar{o}_n\rangle$ is a finite sequence of observed $\mathcal{N}$-states satisfying that $\forall 0 \leq i < n.\ ts_i < ts_{i+1}$.

As mentioned above, a quality function of KPIs is used to abstract the $\mathcal{N}$-states so that they can be *finitely* stored by TOA.

Definition 3: A map $qty : \mathbb{R}^k \cup \{\perp\} \to \mathbb{Z}$ is a *valid quality function* iff the image of qty ($img(qty) \subseteq \mathbb{Z}$) is a finite set and $qty(\perp) = 0$.

Definition 4: An *abstract* $\mathcal{N}$*-state* is a pair of the form $(conf, v) \in \mathcal{C} \times \mathbb{Z}$. Given a valid quality function qty and a $\mathcal{N}$-state $(conf, \bar{o}) \in \mathcal{C} \times (\mathbb{R}^k \times \{\perp\})$, the abstraction of $(conf, \bar{o})$ is defined using qty as $qty(conf, \bar{o}) = (conf, qty(\bar{o}))$.

TOA, defined below, is built using abstract $\mathcal{N}$-states. In the sequel, qty denotes the valid quality function that relates $\mathcal{N}$-states and abstract $\mathcal{N}$-states.

Definition 5: A TOA is a tuple $\mathcal{A} = \langle Q, q_0, E, f_{\mathcal{A}}\rangle$ where Q is a finite set of automaton states, $q_0 \in Q$ being the initial state. $f_{\mathcal{A}}$ is an *injective* function that associates each automaton state $q \in Q$ with an abstract $\mathcal{N}$-state $(conf, v) \in \mathcal{C} \times \mathbb{Z}$. It is assumed that $f_{\mathcal{A}}(q_0) = (conf_d, 0)$. E is a map with the TOA edges of the form $(q, q') \mapsto label$ with $q \neq q'$. States q and q' are, respectively, the source/target states of the edge and *label* is one of the two possible actions:

- **cf**$(conf)$ with $conf \in \mathcal{C}$, which represents a transition due to a configuration change;
- **rd**(n) with $n \in \mathbb{N}^+$, which represents a transition due to a change in the quality of the observed KPIs. Argument n is a guard to register when it is possible to fire the transition.

Definition 6: Given $\mathcal{A} = \langle Q, q_0, E, f_{\mathcal{A}}\rangle$, the concretization function $\gamma : Q \to \mathcal{C} \times (\mathbb{R}^k \cup \{\perp\})$ is defined as $\gamma(q) = \{(conf, \bar{o}) | f_{\mathcal{A}}(q) = qty(conf, \bar{o})\}$.

Intuitively, each TOA state $q \in Q$ is a label of the abstract state $f_{\mathcal{A}}(q) = (conf, v)$ oriented to finitely represent the set of $\mathcal{N}$-states $\gamma(q)$ in the automaton. In addition, a TOA transition $(q, q') \mapsto label$ means a network evolution due to a configuration change (label **cf**$(conf)$) to $conf$, or to a change in the quality of the observed KPIs (label **rd**(n)), when more than n time units have passed. Each state $q \in Q$ may have several outgoing **cf**-transitions, since the network configuration can always be substituted by a different one. However, assuming that the network behaves in a deterministic manner, there may be at most one possible deviation in the quality of the observed KPIs due to the time passing, as established in the following condition:

Deviation Condition: Each TOA $\mathcal{A} = \langle Q, q_0, E, f_{\mathcal{A}}\rangle$ must satisfy that for every state $q \in Q$, there may exist *at most* one outgoing edge of the form $(q, -) \mapsto$ **rd**$(-)$ in E.

Example 1: Figure 6 shows an example of TOA that represents the behavior of a simple network with the RLC AM profile p_{AM} and the delay δ being the only configurable parameter and observed KPI, respectively. It is assumed that parameter p_{AM} can take values $p1$ (the initial default value) or $p2$. Delay δ (measured in millisecs) is abstracted using

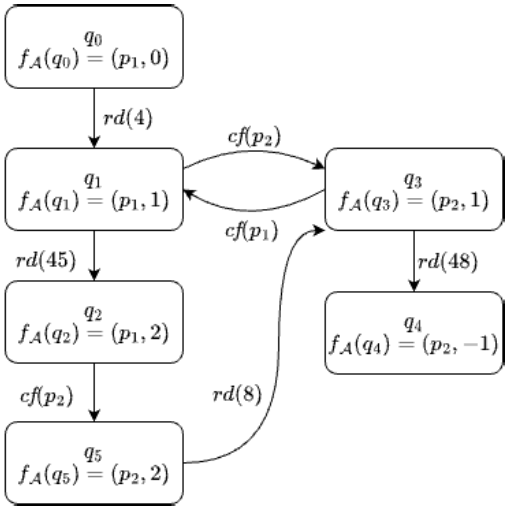

FIGURE 6. Example of TOA.

function $qty : \mathbb{R} \cup \{\perp\} \to \mathbb{Z}$ defined as follows: $qty(\perp) = 0$ and

$$qty(\delta) = \begin{cases} 1 & iff\ \delta \in (0, 5] \\ 2 & iff\ \delta \in (5, 10] \\ -1 & otherwise \end{cases}$$

In Figure 6, each automaton state q also shows the value of $f_{\mathcal{A}}(q)$ in which the first component is the RLC AM profile configured and the second one is the abstraction of the delay. Note that in the initial state q_0 the configured RLC profile is $p1$ and the delay is unknown. Transitions labeled with **rd**$(-)$ (e.g. $(q_0, q_1) \mapsto$ **rd**(4), $(q_1, q_2) \mapsto$ **rd**(45), $(q_3, q_4) \mapsto$ **rd**(48) and $(q_5, q_3) \mapsto$ **rd**(8)) represent a change in the quality of the observed delay (when 4, 45, 48 and 8 time units have passed). Finally, transitions labelled with **cf**$(-)$ (e.g. $(q_1, q_3) \mapsto$ **cf**(p_2), $(q_3, q_1) \mapsto$ **cf**(p_1) and $(q_2, q_5) \mapsto$ **cf**(p_2)) are fired by a change in the RLC AM profile.

The aim of the rest of this section is to formally relate each TOA with the subset of $\mathcal{N}$-traces it represents. To do this, in Definition 8 the so-called trace-based semantics of TOA is constructed using a transition system that produces a set of $\mathcal{N}$-traces. This set is constructed using function γ, given in Definition 6, which relates each TOA state with the set of $\mathcal{N}$-states it abstracts. Similarly, the following definition describes how TOA are also able to abstract consecutive $\mathcal{N}$-states of the real network.

Definition 7: Let $\mathcal{A} = \langle Q, q_0, E, f_{\mathcal{A}}\rangle$ be a TOA. The semantics of $\mathcal{A}$ is defined by means of the transition system $(S, s_0, \bar{\rightarrow})$ with $S \subset Q \times \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \times \mathcal{C} \times (\mathbb{R}^k \cup \{\perp\})$. Each state $s = (q, c, ts, conf, \bar{o}) \in S$ contains a TOA state $q \in Q$, a clock value $c \in \mathbb{R}^{\geq 0}$ and an observed $\mathcal{N}$-state $\langle ts, conf, \bar{o}\rangle$. The initial state is $s_0 = (q_0, 0, 0, conf_d, \perp) \in S$. Each state $(q, c, ts, conf, \bar{o}) \in S$ must satisfy that $(conf, \bar{o}) \in \gamma(q)$.

Transition relation $\bar{\rightarrow} \subset S \times \{\mathbf{cf}, \mathbf{rd}, \mathbf{dl}\} \times S$ is defined by rules (1), (2) and (3) presented below. In these rules, component ts registers the global timestamp and the value

of clock c_j of the target state $s_j = (q_j, c_j, ts_j, conf_j, \bar{o}_j)$ is calculated to register the time elapsed in q_j.

1) *Transition produced by a TOA edge labelled with a change in the quality of the observed KPI*. This transition is fired for $s_i = (q_i, c_i, ts_i, conf_i, \bar{o}_i)$ if there exists a TOA edge of the form $(q_i, -) \mapsto \mathbf{rd}(n)$ and more than n time units have passed at state q_i:
   $(q_i, c_i, ts_i, conf, \bar{o}_i) \xrightarrow{\mathbf{rd}} (q_j, 0, ts_j, conf, \bar{o}_j)$,
   if $\exists n > 0$
   a) $(q_i, q_j) \mapsto \mathbf{rd}(n) \in E$ and
   b) $c_i + (ts_j - ts_i) \geq n$
2) *Transition produced by a TOA edge labelled with a configuration change*. This transition is fired for $s_i = (q_i, c_i, ts_i, conf_i, \bar{o}_i)$ if there exists a TOA edge of the form $(q_i, -) \mapsto \mathbf{cf}()$ and rule (1) cannot be applied:
   $(q_i, c_i, ts_i, conf_i, \bar{o}_i) \xrightarrow{\mathbf{cf}} (q_j, 0, ts_j, conf_j, \bar{o}_j)$, if
   a) $(q_i, q_j) \mapsto \mathbf{cf}(conf_j) \in E$ and
   b) if there exist $q_k \in Q$ and $n \in \mathbb{N}$ such that $(q_i, q_k) \mapsto \mathbf{rd}(n) \in E$, then $c_i + (ts_j - ts_i) < n$
3) *Transition produced only by the time passing*. It can be fired when rule (1) cannot be applied:
   $(q, c_i, ts_i, conf, \bar{o}_i) \xrightarrow{\mathbf{dl}} (q, c_j, ts_j, conf, \bar{o}_j)$ if
   a) $c_j = c_i + (ts_j - ts_i)$ and
   b) if there exist $q_k \in Q$ and $n \in \mathbb{N}$ such that $(q, q_k) \mapsto \mathbf{rd}(n) \in E$, then $c_j < n$

The **deviation condition** given above assures that there is no ambiguity when applying rule (1), since at most one outgoing **rd**-transition exists for each state q_i. In addition, observe that the **rd**-transition defined in rule (1) is mandatory if conditions (1.a) and (1.b) hold. Otherwise, rules (2) and (3) can be applied in a non-deterministic manner, if possible.

Example 2: The transition system determined by a TOA is clarified by demonstrating two sequences of states produced by the automaton in Figure 6:

$$seq_1 = (q_0, 0, 0, p_1, \perp) \xrightarrow{\mathbf{rd}} (q_1, 0, 4, p_1, 2ms) \xrightarrow{\mathbf{dl}} (q_1, 4, 8, p_1, 1ms) \xrightarrow{\mathbf{dl}} (q_1, 8, 12, p_1, 4ms) \xrightarrow{\mathbf{rd}} (q_2, 0, 45, p_1, 7ms) \cdots$$

Transition from $(q_0, 0, 0, p_1, \perp)$ to $(q_1, 0, 4, p_1, 2ms)$ is produced by rule (1) when 4 time units have passed. The two consecutive **dl**-transitions in seq_1 are generated by rule (3) since the elapsed time is less than 45. The last transition is produced by rule (1) and involves a change in the quality of the observed KPI. Note that the sequence could continue by applying **dl**-transitions indefinitely.

$$seq_2 = (q_0, 0, 0, p_1, \perp) \xrightarrow{\mathbf{rd}} (q_1, 0, 4, p_1, 2ms) \xrightarrow{\mathbf{cf}} (q_3, 0, 8, p_2, 2ms) \xrightarrow{\mathbf{dl}} (q_3, 4, 12, p_2, 4ms) \xrightarrow{\mathbf{cf}} (q_1, 0, 16, p_1, 3ms) \cdots$$

Sequence seq_2 starts as seq_1 and then carries out two **cf**-transitions. The sequence could continue by alternatively changing the network configuration as long as the time elapsed at the corresponding states is less than 16 or 48.

Given a state $s = (q, c, ts, conf, \bar{o})$ of the transition system $(S, s_0, \bar{\rightarrow})$, the $\mathcal{N}$-state $s_\downarrow$ is constructed by projecting s over its two last components as $s_\downarrow = (conf, \bar{o})$.

Definition 8: The trace-based semantics of the transition system $(S, s_0, \bar{\rightarrow})$ is given by $\mathcal{O}(S, s_0, \bar{\rightarrow})$ defined as $\{s_{0\downarrow} \cdot s_{1\downarrow} \cdots s_{n\downarrow} \mid \forall\, 0 \leq i < n.\ s_i \bar{\rightarrow} s_{i+1}\}$.

That is, $\mathcal{O}(S, s_0, \bar{\rightarrow})$ contains the set of $\mathcal{N}$-traces (according to Definition 1) that can be constructed applying transition relation $\bar{\rightarrow}$ from the initial state s_0 and projecting the resulting states over their two last components.

Definition 9: A TOA $\mathcal{A}$ accepts a set of $\mathcal{N}$-traces $\mathbf{T}$ iff $\mathbf{T} \subseteq \mathcal{O}(S, s_0, \bar{\rightarrow})$, $(S, s_0, \bar{\rightarrow})$ being the transition relation determined by $\mathcal{A}$

Example 3: The sequence of projected states determined by seq_1 and seq_2 of Example 2 is shown to clarify how the elements of $\mathcal{O}(S, s_0, \bar{\rightarrow})$ are constructed:
$seq_{1\downarrow} = (p1, \perp) \cdot (p1, 2ms) \cdot (p1, 1ms) \cdot (p1, 4ms) \cdot (p1, 7ms)$
$seq_{2\downarrow} = (p1, \perp) \cdot (p1, 2ms) \cdot (p2, 2ms) \cdot (p2, 4ms) \cdot (p1, 3ms)$

C. CONSTRUCTION OF THE LEARNED AUTOMATON

The algorithm *Learn* is now described as a method for constructing a TOA from a finite set of *observed network traces* $\mathcal{T} = \{\pi^1, \cdots, \pi^p\}$ produced during p different TSN sessions in a given 5G network. Each trace $\pi^i = z^i_0 \cdot z^i_1 \cdot \cdots \cdot z^i_{n-1} \in \mathcal{T}$ is a finite sequence of *observed* $\mathcal{N}$*-states* z^i_j (Definition 2) produced when a configuration change event takes place or when a fixed time threshold has elapsed. Algorithm *Learn* returns a TOA $\mathcal{A}_p$ which is able to accept all the behaviors in $\mathcal{T}$ according to Definition 9. To construct this automaton, the algorithm makes use of a valid quality function qty as defined by Definition 3.

Algorithm *Learn* proceeds as follows. Firstly, it creates an initial TOA $\mathcal{A}_0$, with only an initial state q_0 and no transitions. It also creates the dictionary lv to associate each automaton state with the last time instant that was visited. Initially, the dictionary only includes the pair $q_0 \mapsto 0$ to indicate that state q_0 is reached at time 0. Finally, it calls function *Compose* to successively expand the TOA, incorporating the behavior displayed by each trace of $\mathcal{T}$.

The algorithm works by traversing the input automaton and adding new states to it when required. Thus, in the function, variable q_c always points to the current automaton state and $conf_c$ is the current configuration at q_c. The following auxiliary functions are used in the algorithm to consult/update the automaton under construction $\mathcal{A}$ or the map lv. Function $qty(\bar{o})$ (e.g. line 13) calculates the abstract representation of the observed (see Definition 3). Function $get(conf, qty(\bar{o}))$ (e.g. line 13) returns (if it exists) a state $q \in Q$ such that $f_{\mathcal{A}}(q) = (conf, qty(\bar{o}))$. Otherwise, it returns *None*. Function $newState(conf, qty(\bar{o}))$ (e.g. line 15) creates a new state q, adds it to Q, and extends $f_{\mathcal{A}}$ to associate q to the pair $(conf, qty(\bar{o}))$. Function $addTran(q, lb, q')$ (e.g. line 17)

creates a new transition from state q to q' labeled by lb. If key (q, q') is not in E, *addTrans* simply adds pair $(q, q') \mapsto lb$ to E. Otherwise, if the label is of the form **cf**(*conf*), E does not change. When the label is of the form **rd**(t), if $(q, q') \mapsto$ **rd**(t') is the current pair in E, it is substituted by $(q, q') \mapsto$ **rd**($min(t, t')$), $min(t, t')$ being the minimum value between t and t'. Function *update*(q, ts) (e.g. line 18) adds or updates lv so that pair $q \mapsto ts$ is in the dictionary. Finally, function *getValue*(q) (e.g. line 21) returns the time stamp registered in lv for state q, if q is in the dictionary, or *None* otherwise.

The function *Compose*, shown in Algorithm *Learn*, proceeds as follows. First, it initiates the current state q_c to q_0. Then, it iterates through all observed $\mathcal{N}$-states $z_i = \langle ts, conf, \bar{o} \rangle$ and modifies the current automaton to incorporate the observed behavior when required. It uses the function *get* to check whether the automaton contains a state q that matches the configuration and the observed quality of z_i. If not, a new state (also called q) is created to force the automaton to accept z_i.

Compose distinguishes two cases. On the one hand, lines 16-19 manage the case in which z_i has a configuration *conf* that differs from that of the current state $conf_c$. On the other hand, lines 20-24 focus on the case in which z_i has the same configuration *conf* as the current state, but the quality of the observed values $qty(\bar{o})$ is different, which leads to a different automaton state q. In both cases, the algorithm adds a new transition from the current state q_c to q. Observe that when the transition is fired by a change in the quality of the observed values (lines 20- 24), it is labelled with **rd**(t'), where t' is the time elapsed in q_c. In any case, when a new transition is added to the automaton, the dictionary of lv must also be updated with the map $q \mapsto ts$ and q becomes the new current state. When all trace states have been traversed, the new automaton is returned in line 27.

The following results establish the correctness of the algorithm in the sense that all the $\mathcal{N}$-traces used to construct the TOA are accepted by the automaton.

Definition 10: Given an observed $\mathcal{N}$-trace $\pi = \langle ts_0, conf_0, \bar{o}_0 \rangle \cdots \langle ts_{n-1}, conf_{n-1}, \bar{o}_{n-1} \rangle$, $\pi_{i\downarrow}$ $(0 \leq i < n)$ denotes the $\mathcal{N}$-trace $(conf_0, \bar{o}_0) \cdots (conf_i, \bar{o}_i)$ obtained discarding the last $n - i - 1$ states and removing the timestamp. When no states are discarded, $\pi_{\downarrow}$ is written instead of $\pi_{n-1\downarrow}$.

Lemma 1: Let $\mathcal{B}_0 = \mathcal{A}$ be the input TOA of *Compose* and $\pi = \langle ts_0, conf_0, \bar{o}_0 \rangle \cdots \langle ts_{n-1}, conf_{n-1}, \bar{o}_{n-1} \rangle$ the observed $\mathcal{N}$-trace to be learned. For all $0 < i \leq n$, if $\mathcal{B}_i$ is the automaton built after the i-th iteration of loop **for** (line 10), then $\mathcal{B}_i$ accepts the $\mathcal{N}$-trace $\pi_{i\downarrow}$.

Lemma 2: For all $1 \leq i \leq p$, the automaton $\mathcal{A}_i$ returned by function *Compose* (line 5) accepts the set of $\mathcal{N}$-traces $\{\pi^1_{\downarrow}, \cdots, \pi^i_{\downarrow}\}$.

Theorem 1: Let $\mathcal{A}$ be the TOA returned by Algorithm Learn from the set of observed $\mathcal{N}$-traces $\mathcal{T} = \{\pi^1, \cdots, \pi^p\}$. Then the set of $\mathcal{N}$-traces $\mathbf{T} = \{\pi^1_{\downarrow}, \cdots, \pi^p_{\downarrow}\}$ is accepted by $\mathcal{A}$.

The convergence of the Algorithm *Learn* depends on the number of states of the automaton built from the set $\mathcal{T}$ which constitutes its input. If it is assumed that $\mathcal{T}$ is finite and that the number of states in each trace of $\mathcal{T}$ is also finite, then termination of the Algorithm *Learn* holds trivially. However, assuming that the automaton is continuously being updated with new observed traces to record new behaviors, convergence depends on the number of different possible configurations (the size of set $\mathcal{C}$) and on the number of possible abstract KPI values (the size of $img(qty)$). However, on the one hand, it is assumed that $img(qty)$ is finite and on the other, following [5], it can also be assumed that there exists a finite number of realistic configuration values. In consequence, the automaton built is finite.

Finally, it is worth noting that *Learn* returns an automaton that represents the behavior of the network for sessions with a specific traffic pattern. Since TSN specifies a finite set of possible traffic patterns [2], and the traffic pattern does not change over a session, our approach is to generate a set of independent automata, each one representing the network when using a different traffic pattern. Thus, in prediction and reconfiguration phases, the automaton corresponding to the announced traffic pattern will be used.

Algorithm Learn Construction of the Automaton $\mathcal{A}_k$ Accepting a Set of Traces $\mathcal{T}$

```
1  Algorithm Learn (↓ T = {π^1, ··· , π^p}, ↑ A_p) :
2      A_0 := ⟨{q_0}, q_0, ∅, {q_0 → (conf_d, 0)}⟩;
3      lv = {(q_0, 0)};
4      for i := 1 ... p do
5          A_i := Compose(A_{i−1}, π^i, lv) ;
6      return A_p;
7  Function Compose (↕ A, ↓ π = z_0 ... z_{n−1}, ↕ lv) :
8      q_c := q_0;
9      B_0 := A;
10     for i := 0 ... n − 1 do
11         (conf_c, _) := f_A(q_c);
12         ⟨ts, conf, o̅⟩ := z_i;
13         q := B_i.get(conf, qty(o̅));
14         if q = None then
15             q := B_i.newState(conf, qty(o̅));
16         if conf ≠ conf_c then
17             B_i.addTran(q_c, cf(conf), q);
18             lv.update(q, ts);
19             q_c := q;
20         else if q ≠ q_c then
21             t' := ts − lv.getValue(q_c);
22             B_i.addTran(q_c, rd(t'), q);
23             lv.update(q', ts);
24             q_c := q;
25         B_{i+1} := B_i;
26     A := B_n;
27     return A;
```

D. AUTOMATA FOR PREDICTION AND RECONFIGURATION

This section describes Algorithm *Predict*, which can predict the deviation of the observed KPIs to undesirable quality values and propose some alternative network configurations in advance based on a learned TOA $\mathcal{A}$. The strategy is based on analyzing the transitions of the form $\mathbf{rd}(n)$ of $\mathcal{A}$ (i.e, those that reflect a change in the quality of the KPIs). Thus, when the network stays at an automaton state in which a KPI degradation is observed after n time units, the algorithm informs that a reconfiguration action should be carried out, if possible.

The algorithm executes in parallel with the network. Both the algorithm and the network communicate via channels *cin* and *cout*. The network sends periodically observed $\mathcal{N}$-states of the form $\langle ts, conf, \bar{o}\rangle$ through *cin*, and the algorithm uses *cout* to send the result of the prediction. The CSP notation is used to denote the emission (!) and reception (?) of messages.

When *Predict* receives a new observed $\mathcal{N}$-state *nextState*, it responds through channel *cout* with one of the following messages:

- *Continue*: *nextState* has the expected quality wrt its configuration, and no action is needed.
- *Reconf*(*List*(*conf*)): The network must be reconfigured with any of the configurations in *List*(*conf*) to avoid having the observed KPIs reach an undesired observed quality. Deciding which of the proposed configurations is the best is beyond the scope of the algorithm.
- *Warning*: *nextState* has an undesired quality level, but the algorithm cannot suggest any reconfiguration action.
- *Unknown*: *nextState* is not recognized by the automaton. In this case, the algorithm ends.

Algorithm *Predict* makes use of function $desirable : \mathbb{Z} \rightarrow \mathbb{B}$, which given an integer number that represents a quality value (of $img(qty)$), returns whether this quality level is desirable for the observed KPIs. In addition, the algorithm uses the auxiliary functions *getConfigs* and *getRd*. Function *getConfigs* takes the TOA $\mathcal{A}$, a state q_c of $\mathcal{A}$ and it returns a list with the configurations, which label the outgoing transitions from q_c, leading to automaton states with desirable quality values. Function *getRd* has the same input as *getConfigs*. It returns the natural number (greater than 0) that goes with the outgoing transition from q_c labelled with $\mathbf{rd}()$, if it exists, or 0, otherwise. Finally, it is assumed that the network reconfiguration takes at most T_r time units. This constant is used in the algorithm to control when a reconfiguration action must be carried out.

As commented above, algorithm *Predict* proceeds by reading network states (line 4) from channel *cin* and responding via *cout*. To calculate the proper response, it traverses the TOA $\mathcal{A}$, using variable q_c (initialized to the automaton initial state) and variable t used to register the timestamp when the current state q_c was last reached.

The algorithm first checks whether the sequence of network states being observed is accepted by the TOA (lines 10-19). When the current observed state is not recognized by $\mathcal{A}$, the algorithm returns *Unknown* (line 33). In particular, this occurs in the following cases:

- no automaton state matches *nextState* (line 10),
- no transition from q_c to q exists, q being the automaton state representing *nextState* (line 14),
- $\mathcal{A}$ contains a transition of form $\mathbf{rd}(n)$ from q_c to q, but the network has not spent at least n time units at the current state to enable the transition (line 16).

Otherwise, *nextState* is accepted by $\mathcal{A}$ and variables q_c and t are properly updated. Lines 20-31 decide the actions to be taken considering the current observed $\mathcal{N}$-state:

- if the observed values do not have the desired quality, a *Warning* message is sent via *cout* (line 21),
- if the observed KPIs have the desired quality and there is no forthcoming **rd**-transition in less than T_r time units, value *Continue* is sent through *cout* (line 25),
- if the observed values still have the desired quality but there is an imminent **rd**-transition towards a state with a non-desired quality, the algorithm consults $\mathcal{A}$ to check if this can be avoided by means of a network reconfiguration (line 27). If there are no alternative configurations, the algorithm sends a *Warning* (line 29) and otherwise, it sends a *Reconf* message with the list of possible new configurations (line 30).

Example 4: Following with the previous examples, let us suppose that the TOA $\mathcal{A}$ in Figure 6 is the input to the *Predict* algorithm and that the value of T_r is 4 time units. 0 and 1 are considered to be the *desirable* quality values for KPI δ, that is, *Predict* will help to avoid reaching states q_2, q_4 and q_5. It is assumed the algorithm receives the following sequence of observed $\mathcal{N}$-states through *cin*:
$\langle 0, p_1, \perp\rangle \cdot \langle 4, p_1, 2ms\rangle \cdot \langle 8, p_1, 3ms\rangle \cdot \langle 12, p_1, 2ms\rangle \cdot \langle 16, p_1, 3ms\rangle \cdots$

The first observed $\mathcal{N}$-state corresponds to q_0 in $\mathcal{A}$, the second one represents a valid transition $\mathbf{rd}(4)$ from q_0 to q_1. The third and fourth states correspond to stay in q_1. Until now, the algorithm has responded to each observed $\mathcal{N}$-state with *Continue* since all states are recognized by $\mathcal{A}$ and the network has elapsed just 8 time units in q_1, which is less than $16 - T_r$ (the time instant at which q_5 could be reached). The fifth state also corresponds to stay in q_1 but, at this point, the network has spent 12 time units at this state, and in less than $T_r = 4$ time units the system may reach q_5. This is why the algorithm calls function *getConfigs* and obtains a list with just one alternative configuration, which means that it is possible to change the RLC AM mode from p_1 to p_2 to avoid the undesirable quality. In this case, *Predict* sends the message *Reconf*($\{p_2\}$) suggesting that the network should be reconfigured to p_2. Depending on the action performed by network, the next observed $\mathcal{N}$-state could probably show the change in the configuration (e.g. transition $\mathbf{cf}(p_2)$ from q_1 to q_3) or the evolution of observed KPIs to an undesirable quality (e.g. transition $\mathbf{rd}(16)$ from q_1 to q_2). In this last case, the next response of *Predict* will be *Warning*.

Algorithm Predict Reconfiguration

```
1  Algorithm Predict(↓ A, ↓ cin, ↓ cout):
2      q_c := A.q_0;
3      t := 0;
4      cin?nextState;
5      if (nextState = None) then
6          goto line 34 ;
7      (ts, conf, ō) := nextState ;
8      qty := qty(ō);
9      q := A.get(conf, qty);
10     if (q = None) then
11         goto line 33;
12     else if (q ≠ q_c) then
13         label = A.getTrans(q_c, q);
14         if (label = None) then
15             goto line 33;
16         if (label = rd(n) && ts − t < n) then
17             goto line 33;
18         q_c := q;
19         t := ts;
20     if (!desirable(qty)) then
21         cout!Warning
22     else
23         m := getRd(A, q_c);
24         if (m = 0||m − (ts − t) > T_r) then
25             cout!Continue;
26         else
27             lconf := getConfigs(A, q_c);
28             if (lconf.isEmpty()) then
29                 cout!Warning;
30             else
31                 cout!Reconf(lconf);
32     goto line 4 ;
33     cout!Unknown;
34     return;
```

Function getConfigs

```
1  Function getConfigs(↓ A, ↓ q_c, ↑ lConf):
2      succ_d := A.getCfSucc(q_c).filter(desirable);
3      for q in succ_d do
4          lConf.add(A.getTrans(q_c, q));
5      return lConf;
```

Function getRd

```
1  Function getRd(↓ A, ↓ q_c, ↑ m):
2      trd := A.getRdTrans(q_c);
3      m := 0;
4      if trd ≠ None then
5          (q_c, q, m) := trd;
6          if desirable(q) then
7              m := 0;
8      return m;
```

V. TESTBED AND EVALUATION

The objective of this section is to present some experiments carried out to evaluate the *AutomAdapt* tool for Zero Touch Configuration. For this purpose, the following main elements are introduced. First, the TSN over 5G testbed available at the University of Malaga,[1] which includes prototype implementations of the standardized features, namely the time synchronization and the development of TSN translators to support TSN sessions ; second the key implementation details of the partially standardized TSN AF presented in Section III-B, the *AutomAdapt* tool, which follows the architecture shown in Figure 5. Finally, the results of the different experiments that evaluate *AutomAdapt* implementation.

A. TSN OVER 5G TESTBED

Currently there are no commercial 5G testbeds that fully integrate TSN technology, mainly because of the open issues described in Section III. A solution for these open challenges is proposed as follows.

First, due to the fact that the network does not yet have the necessary capabilities to transport the synchronization packets as introduced by recent 3GPP standards, this paper proposes the use of two TSN Grand Master (GM) clocks (instead a single GM clock) to achieve the time synchronization at the edges of the 5G network. A single GPS (Global Positioning System) signal is split to provide the same time reference to these GM clocks. The difference between the synchronization packets distributed by both GM clocks has been verified with an oscilloscope as negligible.

Secondly, since there are no TSN translators on the market either, the TSN translators in our testbed have been developed using P4 domain-specific language [38], which is an ideal candidate for data plane programming of network devices. The P4 functionality is based on match-action tables, which contain different table entries. In addition, the P4Runtime interface allows programming the control plane in real time. Therefore, the table entries can be managed (added, removed, or modified) dynamically and the packet headers can be adapted, for instance, by inserting or eliminating the IEEE 802.1Q protocol (also called dot1Q).

Eventually, the use of an Application Programming Interface (API) on the user application side allows declaring in advance the traffic to be sent. This way, the TSN AF can initiate all necessary procedures and communications with the other NFs previously. The API has been developed within the

[1] ITIS/Victoria Network http://www.victoria-network.eu

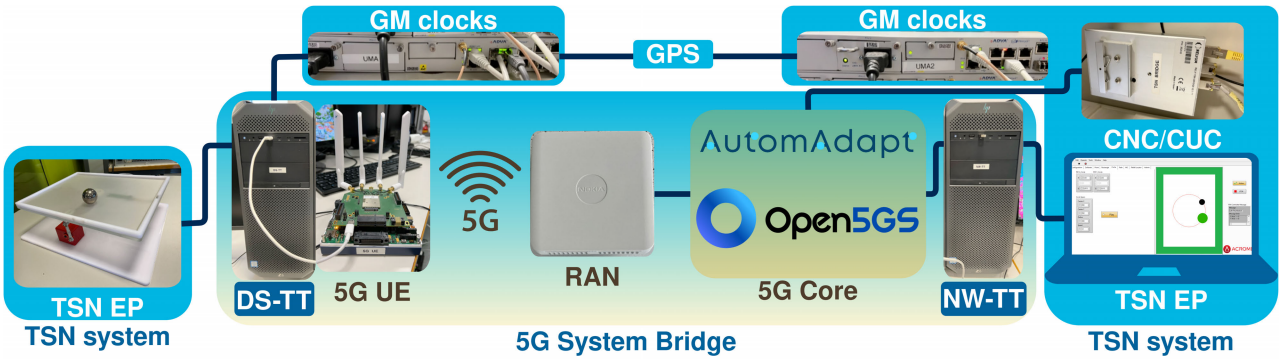

FIGURE 7. TSN over 5G testbed at the university of malaga.

scope of the EVOLVED-5G project and details are available at TSN AF API repository.[2]

Figure 7 shows the main elements of the TSN over 5G testbed:

- Two TSN endpoints (Relyum), in this case a ball balancing table (ACROME), is used as a visual demonstrator.
- A 5G UE (Telit fn980m modem).
- TSN translators, DS-TT and NW-TT, running on two workstations.
- A fixed TSN bridge (Relyum) acting as CNC and CUC.
- The Open5GS,[3] which is an Open Source implementation for 5G Core.
- The RAN part is composed of a Nokia indoor deployment (n78 band) managed by the Nokia AirScale system.
- Time synchronization devices (ADVA) used as TSN GM clocks.

B. AutomAdapt IMPLEMENTATION

This section describes some implementation issues of *AutomAdapt*. In particular, it presents the design of the zero touch configuration loop shown in Figure 5. The *AutomAdapt* functionality is divided into four modules: monitoring, learning, prediction and reconfiguration. In addition, the figure shows other components that are part of the testbed, such as the network components, and the learned automata. The functionality of each module is explained below, together with the connections between them, the inputs-outputs, and the interaction with the learned automata and the real network components. The AutomAdapt code is available at AutomAdapt repository .[4]

The **monitoring** module is in charge of collecting information from the different points of the network. This information includes all relevant header information of the packets sent and received to/from TSN endpoints, through RAN, 5G core network and TSN translators. Currently, the monitoring module has access to all these network components. However, the most relevant task for the monitoring implementation and, in turn, for the final users, is to obtain this information at the TSN endpoints, thus making it possible to have end-to-end KPIs.

The monitoring module is composed of two Java programs (talker and listener) that capture all packets at the TSN endpoints, process them to extract all the relevant information (e.g. timestamp, packet ID, headers, etc.) and insert this information into Kafka topics. Then, the KPI generator, which is another Java program, uses Kafka streams to process the data, generate the KPIs (delay, jitter, throughput, and packet loss), and inject the resulting KPIs into specific Kafka topics. Eventually, these KPIs are inserted in an Influx Data Base (InfluxDB), where they can be processed with different functions, such as sum or average. Note that fine-grained information can be obtained, even differentiating between each packet sent through the network.

The data can be visualized through a dashboard that allows plotting several types of graphs and selecting the time frame to be displayed. Note that the dashboard provides an extra function for users, allowing them to see at a glance the status of the network (KPIs and values of relevant parameters) as an extra function for the user, although it is not required for AutomAdapt operation.

The **learning** module is responsible for constructing a set of *Automata*, where each automaton represents the behavior of the network during TSN sessions with a given traffic pattern and the defined *qty* function. The module, written in Python, implements the *Learn* algorithm explained in depth in Section IV-C, and also the functionality necessary to make queries to the InfluxDB in order to obtain the traces to be learned.

The **prediction and reconfiguration** module is written in Python and implements the `Predict` algorithm explained in detail in Section IV-C. It is in charge of detecting in advance a plausible deviation of the observed KPIs. If a deviation is detected, the module tries to fetch a list of valid network configurations. Then, it decides on the best configuration to apply. Since the selection of the best proposed

[2]TSN AF API: https://github.com/EVOLVED-5G/TSN_FrontEnd
[3]Open5GS: Open Source 5G Core https://open5gs.org
[4]AutomAdapt repository: https://github.com/FLSchempp/AutomAdapt

configurations is beyond the scope of the *Predict* algorithm, it has been decided to sort the configurations proposed for reconfiguration based on the elapsed time in those states, i.e., the configuration learned with the longest elapsed time is selected. Finally, the TSN 5G Bridge is reconfigured.

Some parameters defined by the network operator are specific to the implementation, and are explained below:

- Monitoring cycle time. Period used to calculate the average results of the KPIs (e.g. 1s, 10s, 60s). It is worth noting that a lower value allows for finer-grained results. Conversely, this requires further data processing.
- Configuration goal time. Minimum time needed to consider a proposed configuration suitable to reconfigure the network (e.g. 5min, 1h). It is important to note that a higher value will possibly result in a better performing network configuration. However, in this case fewer configurations are likely to be proposed.
- Stability time. Time required after a network reconfiguration for the changes to be propagated and reflected in the monitoring module (e.g. 1min, 3min). Note that no other network configurations are applied during this period.
- *qty* function error value. An additional value of the *qty* function is defined (in this case, *qty* = -30) to take into account, in the learning process, any range of values different from those defined in Table 1.

C. EVALUATION

Now, the details of the AutomAdapt tool evaluation are presented, as follows: first, the identification of the specific aspects such as the scenario, traffic pattern, and network configuration; then, the explanation of the evaluation process of the AutomAdapt operation cycle, which includes monitoring, learning, prediction, and reconfiguration. Moreover, the results of the experiments performed using the TSN over 5G testbed at the University of Malaga are analyzed. Note that the setup used for evaluation can be replicated using the hardware components presented in Section V-A. In addition, the software components and the instructions to reproduce the process are available at the AutomAdapt repository[4].

- The *evaluation scenario* is made up of the hardware and software included in the TSN over 5G testbed shown in Figure 7.
- TSN endpoints can inject different traffic patterns. Table 2 shows some examples of *traffic patterns* based on the TSN traffic classes defined in [39]. Different experiments have been carried out with several traffic patterns, which have different typical periods and payloads.
- The TSN over 5G testbed supports the configuration of multiple parameters from the core network to the RAN. This leads to millions of different *5G network configurations*. To simplify the presentation, in the following experiments AutomAdapt will learn a set of network traces with different configurations of the RLC AM parameters. Currently, the Nokia RAN supports the configuration of 12 parameters (6 parameters for both downlink and uplink) in the RLC layer (AM mode), which are the following:
  - $MaxRetx_{Threshold}$: the maximum number of ARQ retransmissions allowed.
  - $Poll_{Byte}$: the interval between polls relative to the number of bytes transmitted.
  - $Poll_{PDU}$: the interval between polls in PDU.
  - $t_{PollRetr}$: the value of the timer to retransmit a poll
  - $t_{StatusProhibit}$: timer to prohibit the transmission of acknowledgment status reports.
  - $t_{Reassembly}$: the value of the timer for reassembly

  These parameters can accept between 8 and 62 possible different input values (for more information, see [40]). Table 3 shows 2 possible sets of configurations for these parameters.

TABLE 2. Traffic pattern examples.

Traffic pattern	*Periodicity*	*Typical period* (ms)	*Payload* (bytes)
1	cyclic-async	2	100
2	cyclic-async	10	500
3	cyclic-async	10	1000
4	cyclic-async	20	1000

TABLE 3. 5G network configuration (RLC AM) examples.

Configuration	*ID 1*	*ID 2*
$MaxRetx_{Threshold}$	t3	t3
$Poll_{Byte}$	kB75	kB10
$Poll_{PDU}$	p8	p8192
$t_{PollRetr}$	ms50	ms220
$t_{StatusProhibit}$	ms140	ms10
$t_{Reassembly}$	ms5	ms120

Once the specific aspects of the evaluation have been identified, in order to use AutomAdapt, the *qty* function has to be defined by specifying the correspondences of the observed KPIs with the intervals. The experiments use a *qty* based on the examples presented in Table 1, which defines different intervals for the observed delay and jitter. Furthermore, the *desirable* function has to be established, that is, the desirable observed KPIs during the TSN session. The evaluation process of the AutomAdapt operation cycle consists mainly of two phases, explained below.

1) INITIAL LEARNING PHASE

In this phase, different network configurations with a given traffic pattern have been randomly applied. AutomAdapt has been fed with the observed network traces produced by each network configuration in order to produce an automaton for each traffic pattern. In order to perform an evaluation of AutomAdapt, different automata have been generated that will be used in the next phase of prediction and reconfiguration. To generate these automata, a starting point and

TABLE 4. Automata learned during initial learning phase.

Automaton ID	*Automaton build time*	*Number of states*	*Number of configurations*
1	24h	234	91
2	48h	473	182
3	72h	724	274
4	96h	986	408

TABLE 5. 5G network configurations (RLC AM) used to obtain baseline results for comparison.

Baseline configuration	*ID 1*	*ID 2*	*ID 3*	*ID 4*
$MaxRetx_{Threshold}$	t6	t3	t32	t32
$Poll_{Byte}$	kB8	kB6500	kB8	mB40
$Poll_{PDU}$	p24576	p20480	p1024	p40960
$t_{PollRetr}$	ms500	ms140	ms70	ms40
$t_{StatusProhibit}$	ms800	ms90	ms100	ms40
$t_{Reassembly}$	ms170	ms45	ms10	ms180
Desire network state time (%)	41,99	40,81	45,02	47,22

several time intervals (e.g. 24h or 48h) are chosen to generate automata with a number of different states. Table 4 represents the details of these automata, including the time it took to build each automaton, the number of states created and the number of different network configurations tested. Predictably, the number of network states and configurations progressively increases over time.

2) PREDICTION AND RECONFIGURATION PHASE

Once the aforementioned automata have been created, AutomAdapt is ready for prediction and reconfiguration. Specifically, AutomAdapt continuously monitors the current status of the network. In case AutomAdapt has learned from the current network state that it is possible to reach another state with deviations of the expected quality or the current state does not meet the requirements, it explores network configurations that can be applied in order to fulfill the expected requirements using the automaton learned during the initial learning phase. In order to evaluate AutomAdapt, several tests were performed using traffic pattern 3 (see Table 2). In addition, the desired network state is to maintain the delay in the range of [0, 7]ms and the jitter in the range of [0, 1]ms, corresponding to $qty = 1$ (see Table 1). First, the network's observed KPIs are monitored for at least 12 hours using the same network configuration, i.e., without running AutomAdapt. This information will be used as a baseline for comparing the improvement in network performance, specifically KPIs, when AutomAdapt is not in operation versus when it is in operation. Table 5 shows the network configurations together with the baseline observed KPIs in terms of elapsed time in the desired network state.

Secondly, AutomAdapt is executed multiple times using the automata presented in Table 4 to obtain the observed KPI results while making predictions and applying reconfigurations to the network.

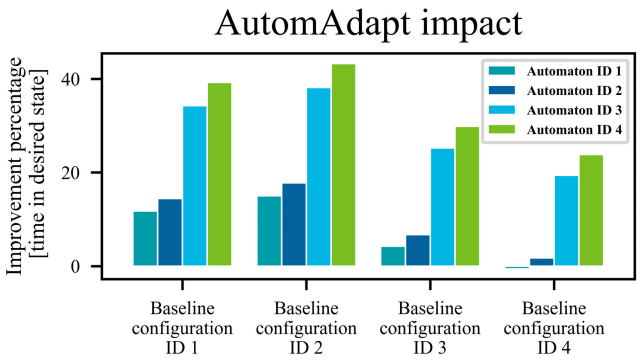

FIGURE 8. AutomAdapt impact: elapsed time in desired network state.

Figure 8 presents a comparison of elapsed time within the desired network state, where the observed KPIs meet the traffic requirements. The figure specifically illustrates the percentage change, ideally improvement, in elapsed time within the desired network state when AutomAdapt is running and its reconfigurations are applied, as compared to the elapsed time in the same desired network state when AutomAdapt is not running (i.e., with a fixed network configuration). It can be observed that the improvement increases progressively if an automaton with a larger number of states is used (see Table 4), which is logical given the extended duration of the initial learning phase and a greater number of network configurations. Consequently, AutomAdapt possesses more information, enabling it to predict and reconfigure the network more effectively. Table 6 presents the information related to the automata resulting from the prediction and reconfiguration phase, called Automaton ID'.

After the evaluation and comparison process, the following conclusions are drawn:

- A considerable improvement (in terms of time elapsed with desirable network KPIs) is obtained from a certain number of states learned by the automaton. For example, in the case of Automaton ID 1 and 2 (<500 states) the percentage of improvement is low, even becoming negative in the case of baseline configuration 4 and using the Automaton ID 1. On the contrary, Automaton ID 3 and 4 can reach an improvement of about 35-45%, which is a remarkable impact on the expected traffic quality.
- Since AutomAdapt continuously learns, even during the prediction and reconfiguration phases, the resulting automata could contain new states that differ from the previously learned ones. This indicates that the automaton did not learn these states during the initial phase, rendering them unusable for prediction and reconfiguration. Nonetheless, once these new states are added to the automaton, they are considered for AutomAdapt. During the different executions, approximately the same number of new states were added (see Table 6).
- The number of reconfigurations applied by AutomAdapt (see Table 6) to maintain the network in the desired state

TABLE 6. Resulting automata during prediction and reconfiguration phase.

Automaton ID'	*Number of states*	*Number of new states*	*Number of* reconfigurations
1	252	18	28
2	494	21	15
3	747	23	17
4	1004	18	11

during a given period of time decreases as the number of states learned becomes larger, reaffirming that the amount of information learned by AutomAdapt is linked to effectiveness.

- It is important to have a set of baseline observed KPIs for comparison, since the results obtained may differ by about 15-20%.

The complete AutomAdapt data set (learned automata, network configurations and reconfigurations applied over time) is accessible in the AutomAdapt repository[4], including all details and files of the evaluation process.

An example trace of the operating cycle is presented below. Figure 9 depicts the evolution of the observed KPIs obtained from an experiment during the initial learning phase, in which the left y-axis represents the delay and the right y-axis the jitter. During this experiment, AutomAdapt monitors and learns the behavior of the network during a TSN session; by using the same conditions as in the evaluation process, i.e. $qty = 1$ (see Table 1) and traffic pattern 3 (see Table 2). In addition, the network configuration ID 1 (see Table 3) is used. In this case, AutomAdapt creates the network state $q1$, which is shown in Figure 6, and learns that using this network configuration the requirements for the given traffic pattern are met for about 230s. This is followed by several deviations, which are also learned by the automaton and are essential for prediction and reconfiguration.

This way, next time AutomAdapt is monitoring and detects that the network has returned to an already learned network state, AutomAdapt is able to predict the deviations of the expected KPIs and apply the necessary reconfiguration to meet the requirements. An example of this process during the prediction and reconfiguration phase is represented in Figure 10. The left part shows the evolution of the observed KPIs, with which AutomAdapt identifies that the network is in the automaton state $q1$, which was previously learned. Since a deviation of the observed KPIs is expected after 230s in this automaton state, a reconfiguration of the network has to be carried out in advance. AutomAdapt will apply a known configuration that continues to meet the traffic requirements during the TSN session. On the right side of Figure 10, the evolution of the observed KPIs after the network reconfiguration is represented. In this case, AutomAdapt creates the automaton state $q2$, which is also shown in Figure 6. Note that $qty = 1$ (see Table 1) and traffic pattern 3 (see Table 2) are still being used. However, network configuration ID 2 (see Table 3) is now in use,which has greater values of $Poll_{PDU}$ and $t_{PollRetr}$. On the one hand, a higher value for

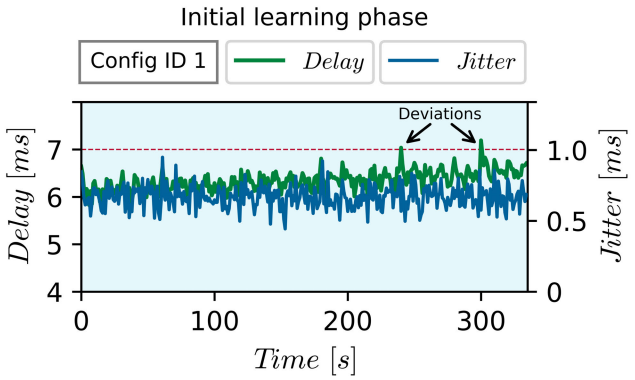

FIGURE 9. Observed KPIs during the initial learning phase.

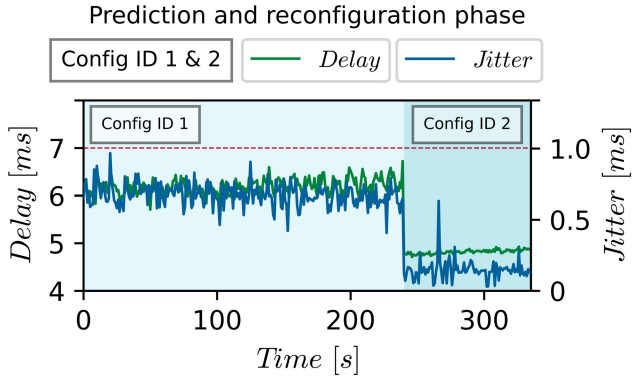

FIGURE 10. Observed KPIs during the prediction and reconfiguration phase.

$Poll_{PDU}$ can reduce control signaling and decrease delay by allowing the transmission of a larger number of PDUs before requesting status information to the UE. On the other hand, a longer $t_{PollRetr}$ value gives the UE more processing and response time to send the status information, which could reduce unnecessary retransmissions and decrease the delay. Additionally, by increasing $t_{Reassembly}$ allows waiting for all RLC data PDUs from higher layers to arrive and be reassembled and can prevent these unnecessary retransmissions. The decrement of the $t_{StatusProhibit}$ value also reduces the delay between PDUs. However, it can lead to higher control signaling. As a result of the reconfiguration, KPIs are improved, effectively meeting the requirements for the given traffic. Note that, although the explanation is valid for all types of traffic, AutomAdapt does not apply this reconfiguration to another type of traffic if it is not a state previously learned in the automaton corresponding to that type of traffic.

VI. CONCLUSION

This paper presents *AutomAdapt*, which is a novel implementation of TSN over 5G with self-reconfiguration based on learning techniques. Automata learning is used to build the Traffic Oriented input/output Automata (TOA) which abstracts the behavior of the network based on real traces of TSN sessions. Then the TOA is employed for prediction and reconfiguration with actual TSN applications. In addition, the

paper shows the evaluation of AutomAdapt and an example of its operation cycle, including the analysis of the results obtained when running the tool in a real scenario. These results show the percentage of improvement during the time in which the network is in the desired state when AutomAdapt is used versus baseline values of the network obtained with static network configurations, that is, without any prediction or reconfigurations applied.

There are three main open lines for future work; first, regarding prediction and reconfiguration, adding Machine Learning techniques based on the actual automaton (TOA) to enhance the current prediction algorithm when reaching *unknown* or *warning* states; and second, improving the underlying testbed infrastructure by moving to 5G Release 16 and using Ethernet PDU sessions. In addition, time synchronization will be improved in the Device-Side Translator (DS-TT) by sending the TSN synchronization information over the 5G network.

REFERENCES

[1] T. Adame, M. Carrascosa-Zamacois, and B. Bellalta, "Time-sensitive networking in IEEE 802.11be: On the way to low-latency WiFi 7," *Sensors*, vol. 21, no. 15, p. 4954, Jul. 2021.

[2] 5G-ACIA. (Feb. 2021). *White Paper: Integration of 5G with Time-Sensitive Networking for Industrial Communications*. [Online]. Available: https://5g-acia.org/whitepapers/integration-of-5g-with-time-sensitive-networking-for-industrial-communications/

[3] P. M. Rost and T. Kolding, "Performance of integrated 3GPP 5G and IEEE TSN networks," *IEEE Commun. Standards Mag.*, vol. 6, no. 2, pp. 51–56, Jun. 2022.

[4] D. Ginthör, J. von Hoyningen-Huene, R. Guillaume, and H. Schotten, "Analysis of multi-user scheduling in a TSN-enabled 5G system for industrial applications," in *Proc. IEEE Int. Conf. Ind. Internet (ICII)*, Nov. 2019, pp. 190–199.

[5] *3GPP System Architecture for the 5G System (5GS)*, document 3GPP TS 23.501, version 16.15.0, 3rd Generation Partnership Project (3GPP), Technical Specification (TS), Dec. 2022. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/23_series/23.501/23501-gf0.zip

[6] *3GPP System Architecture for the 5G System (5GS)*, document 3GPP TS 23.501, version 17.7.0, 3rd Generation Partnership Project (3GPP), Technical Specification (TS), Dec. 2022. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/23_series/23.501/23501-h70.zip

[7] F. Luque-Schempp, L. Panizo, M.-d.-M. Gallardo, P. Merino, and J. Rivas, "Toward zero touch configuration of 5G non-public networks for time sensitive networking," *IEEE Netw.*, vol. 36, no. 2, pp. 50–56, Mar./Apr. 2022.

[8] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, Nov. 1987.

[9] B. Steffen, F. Howar, and M. Merten, "Introduction to active automata learning from a practical perspective," in *Formal methods for Eternal Networked software Systems* (Lecture Notes in Computer Science), vol. 6659, M. Bernardo and V. Issarny, Eds. Berlin, Germany: Springer, 2011, pp. 256–296.

[10] A. Stevenson and J. R. Cordy, "A survey of grammatical inference in software engineering," *Sci. Comput. Program.*, vol. 96, pp. 444–459, Dec. 2014.

[11] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, p. 1054, Sep. 1998.

[12] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Upper Saddle River, NJ, USA: Prentice-Hall, 1989.

[13] M. N. Qureshi, M. I. Tiwana, and M. Haddad, "Distributed self optimization techniques for heterogeneous network environments using active antenna tilt systems," *Telecommun. Syst.*, vol. 70, no. 3, pp. 379–389, Mar. 2019.

[14] Y. Guo, S. Li, W. Jiang, B. Zhang, and Y. Ma, "Learning automata-based algorithms for solving the stochastic shortest path routing problems in 5G wireless communication," *Phys. Commun.*, vol. 25, pp. 376–385, Dec. 2017.

[15] M. Chlosta, D. Rupprecht, and T. Holz, "On the challenges of automata reconstruction in LTE networks," in *Proc. 14th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, New York, NY, USA, Jun. 2021, pp. 164–174.

[16] C. M. Stone, T. Chothia, and J. de Ruiter, "Extending automated protocol state learning for the 802.11 4-Way Handshake," in *Computer Security*, J. Lopez, J. Zhou, and M. Soriano, Eds. Cham, Switzerland: Springer, 2018, pp. 325–345.

[17] J. Antunes, N. Neves, and P. Verissimo, "Reverse engineering of Protocols from network traces," in *Proc. 18th Work. Conf. Reverse Eng.*, Oct. 2011, pp. 169–178.

[18] C. Lee, J. Bae, and H. Lee, "PRETT: Protocol reverse engineering using binary tokens and network traces," in *ICT Systems Security and Privacy Protection*, L. J. Janczewski and M. Kutyłowski, Eds. Cham, Switzerland: Springer, 2018, pp. 141–155.

[19] M. A. Qureshi and C. Tekin, "Fast learning for dynamic resource allocation in AI-Enabled radio networks," *IEEE Trans. Cognit. Commun. Netw.*, vol. 6, no. 1, pp. 95–110, Mar. 2020.

[20] C. Luo, J. Ji, Q. Wang, X. Chen, and P. Li, "Channel state information prediction for 5G wireless communications: A deep learning approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 1, pp. 227–236, Jan./Mar. 2020.

[21] F. Cuadrado, J. C. Duenas, and R. Garcia-Carmona, "An autonomous engine for services configuration and deployment," *IEEE Trans. Softw. Eng.*, vol. 38, no. 3, pp. 520–536, May 2012.

[22] N. F. S. de Sousa, M. T. Islam, R. U. Mustafa, D. A. L. Perez, C. E. Rothenberg, and P. H. Gomes, "Machine learning-assisted closed-control loops for beyond 5G multi-domain zero-touch networks," *J. Netw. Syst. Manage.*, vol. 30, no. 3, p. 46, Jul. 2022.

[23] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, "A survey of machine learning techniques applied to self-organizing cellular networks," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2392–2431, 4th Quart., 2017.

[24] A. Imran, A. Zoha, and A. Abu-Dayya, "Challenges in 5G: How to empower SON with big data for enabling 5G," *IEEE Netw.*, vol. 28, no. 6, pp. 27–33, Nov. 2014.

[25] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi, "5G wireless network slicing for eMBB, URLLC, and mMTC: A communication-theoretic view," *IEEE Access*, vol. 6, pp. 55765–55779, 2018.

[26] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2429–2453, 3rd Quart., 2018.

[27] K. Govindarajan, S. Goel, P. Jayachandran, S. Glover, J. P. Villaverde, J. Cresp, J. Viale, S. Martin, and F. Livigni, "Closed loop optimization of 5G network slices," in *Proc. 15th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2023, pp. 186–188.

[28] Y. Abiko, T. Saito, D. Ikeda, K. Ohta, T. Mizuno, and H. Mineno, "Flexible resource block allocation to multiple slices for radio access network slicing using deep reinforcement learning," *IEEE Access*, vol. 8, pp. 68183–68198, 2020.

[29] G. Sun, Z. T. Gebrekidan, G. O. Boateng, D. Ayepah-Mensah, and W. Jiang, "Dynamic reservation and deep reinforcement learning based autonomous resource slicing for virtualized radio access networks," *IEEE Access*, vol. 7, pp. 45758–45772, 2019.

[30] L. Martenvormfelde, A. Neumann, L. Wisniewski, and J. Jasperneite, "A simulation model for integrating 5G into time sensitive networking as a transparent bridge," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, vol. 1, Sep. 2020, pp. 1103–1106.

[31] M. Gundall, C. Huber, P. Rost, R. Halfmann, and H. D. Schotten, "Integration of 5G with TSN as prerequisite for a highly flexible future industrial automation: Time synchronization based on IEEE 802.1AS," in *Proc. 46th Annu. Conf. IEEE Ind. Electron. Soc. (IECON)*, Oct. 2020, pp. 3823–3830.

[32] F. Kaltenberger, A. P. Silva, A. Gosain, L. Wang, and T.-T. Nguyen, "OpenAirInterface: Democratizing innovation in the 5G era," *Comput. Netw.*, vol. 176, Jul. 2020, Art. no. 107284.

[33] K. Nikhileswar, K. Prabhu, D. Cavalcanti, and A. Regev, "Time-sensitive networking over 5G for industrial control systems," in *Proc. IEEE 27th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2022, pp. 1–8.

[34] F. Prinz, M. Schoeffler, A. Lechler, and A. Verl, "Dynamic real-time orchestration of I4.0 components based on time-sensitive networking," *Proc. CIRP*, vol. 72, pp. 910–915, Jan. 2018.

[35] Z. Satka, M. Ashjaei, H. Fotouhi, M. Daneshtalab, M. Sjödin, and S. Mubeen, "A comprehensive systematic review of integration of time sensitive networking and 5G communication," *J. Syst. Archit.*, vol. 138, May 2023, Art. no. 102852.

[36] L. Lo Bello and W. Steiner, "A perspective on IEEE time-sensitive networking for industrial communication and automation systems," *Proc. IEEE*, vol. 107, no. 6, pp. 1094–1120, Jun. 2019.

[37] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements*, Standard 802.1 WG, Higher Layer LAN Protocols Working Group, Oct. 2018.

[38] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[39] *IEC/IEEE 60802 TSN Profile for Industrial Automation*, document Draft V1.4, Jun. 2022. [Online]. Available: http://www.ieee802.org/1/files/private/60802-drafts/d1/60802-d1-4.pdf

[40] *3GPP Radio Link Control (RLC) Protocol Specification*, document 3GPP TS 38.322, version 15.5.0, 3rd Generation Partnership Project (3GPP), Technical Specification (TS), Mar. 2019. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/23_series/23.501/23501-gf0.zip

FRANCISCO LUQUE-SCHEMPP received the B.Sc. degree in telematics engineering and the M.Sc. degree in telematics engineering and telecommunication networks from the University of Malaga, where he is currently pursuing the Ph.D. degree in computer science. His current research interests include time-sensitive networking, deterministic communications, automata learning, and mobile networks.

LAURA PANIZO received the B.S. degree in telecommunications engineering and the Ph.D. degree in computer science from the University of Malaga. She is currently an Assistant Professor in computer languages with the University of Malaga. She has participated in different European projects, such as EuWireless and, more recently, EVOLVED-5G. Her research interests include formal methods for embedded and mobile communication systems.

MARÍA-DEL-MAR GALLARDO is currently a Professor in computer languages with the University of Malaga. She has a large number of publications in high-impact journals and conferences and has participated in 24 regional and national and European projects. Her research interest includes application of formal methods for building reliable software.

PEDRO MERINO is currently a Professor of computer communications and the Director of the ITIS Software Institute, University of Malaga (UMA). He is involved in software defined 5G/6G networks and software reliability. He has participated in more than 40 projects, including 12 European FP7 and H2020 projects. He represents UMA in NetworldEurope and 6G IA.