Instituto de Tecnología e Ingeniería del Software

INFORME TÉCNICO

ID: ITIS-2022-001

SOFTWARE

UNIVERSIDAD DE MÁLAGA | uma.es

itis.uma.es @itis_uma

Title: Towards Zero Touch Configuration of 5G Non-Public Networks for Time Sensitive Networking

Authors: Francisco Luque-Schempp, Laura Panizo, María-del-Mar Gallardo, Pedro Merino and Javier Rivas

How Published: IEEE Network, vol. 36, no. 2, pp. 50-56, March/April 2022,

DOI: 10.1109/MNET.006.2100442

RIUMA handle: https://hdl.handle.net/10630/24749

ITIS id: ITIS-2022-001

Zero touch configuration of 5G Non-Public Networks to support Time Sensitive Networking

Francisco Luque-Schempp*, Laura Panizo†, María-del-Mar Gallardo‡, Pedro Merino§ and Javier Rivas¶
University of Malaga, Malaga, Spain
Email: {schempp*, laurapanizo†, mdgallardo‡, pmerino§, javier_rivas¶}@uma.es

***Abstract*—The need to increase mobility and to remove cables in industrial environments is pushing 5G as a valuable communication system to connect traditional deterministic Ethernet-based devices. One alternative is the adoption of Time Sensitive Networking (TSN) standards over 5G Non-Public Networks (5G NPN) deployed in the company premises. This scenario presents several challenges, being the configuration of the 5G part the most relevant to provide latency, reliability and throughput balance suitable to ensure that all the TSN traffic can be delivered in time. Our research work addresses this problem from the perspective of the learning automata. Our aim is to learn from the live network to build a smart controller that can dynamically predict and apply a suitable configuration of the 5G NPN that can satisfy the requirements of the current TSN traffic. The article presents the main ideas of this novel approach.**

I. Introduction

Time Sensitive Networking (TSN) is defined as an evolution of Ethernet standards to guarantee deterministic traffic in critical systems like industry environments. Such deterministic traffic requires specific values for Key Performance Indicators (KPIs) such as latency and jitter to match the strict control cycles and traffic windows for industry devices. Nowadays, TSN is a technology mainly used for wired environments in factories. As far as devices in such factories require mobility, shifting TSN to the wireless world is needed. TSN over 5G is a proposal contained in recent 3GPP standards not already implemented [1]. These standards define different procedures and entities, for instance, TSN translators that allow the interaction between the TSN and 5G domains. In addition, the whole 5G network should be configured to get the balance among the Radio Access Network (RAN) parameters, the transport network and the core network to achieve the expected KPIs that guarantee the correct delivery of the TSN traffic. We can assume that most of the 5G networks in industries will be private networks, so called 5G Non-Public Networks (5G NPNs), which involve specific use cases and constraints.

A relevant feature of TSN that eases the configuration of the 5G NPN is the declaration of the intended traffic pattern in advance. Figure 1 represents this process and the role of the 5G network to make the configuration (note that the 5G network in this context is called 5G Bridge). The two TSN end points, talker and listener, communicate to CUC (Centralized User Configuration) their intention to send a traffic with a specific pattern and requirement in terms of key Performance indicatos (KPIs). CUC forwards such intent to CNC (Centralized Network Configuration). It is expected that in the TSN over 5G scenario, these components communicate with the 5G network in order to agree on the configuration that satisfies the requirement of all the traffic patterns.

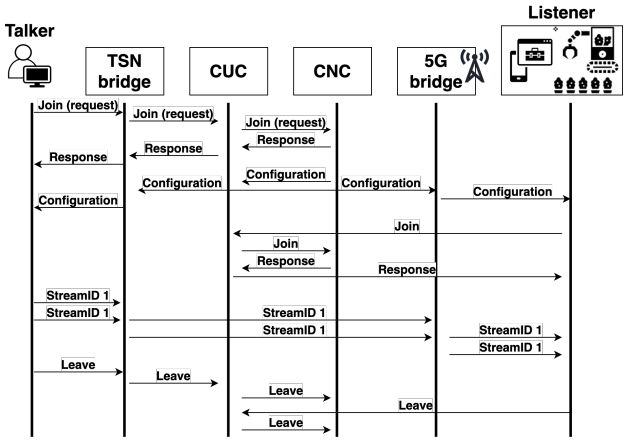

Fig. 1: TSN workflow

This article presents a novel approach to automate the configuration of the 5G NPN to support TSN traffic. We propose to use a centralized *smart controller* to dynamically monitor, learn, predict and reconfigure the 5G NPN in order to satisfy the KPIs requested by the TSN talkers. The article presents the main ideas of this novel approach with a relevant example over a realistic tested.

The rest of the article is structured as follows. Next section presents the challenge of configuring the 5G-TSN bridge, including a number of example parameters that can be customized. Then, we discuss the learning automata approach and algorithm to build a kind of digital twin of the 5G NPN with the TSN traffic and current KPIs. Then, we illustrate the use of the learning algorithm with an example with realistic patterns of TSN traffic over 5G with different network configurations, and we present the testbed supporting this experimental work. In the final section, we present the conclusions.

II. The challenge: zero touch configuration of the 5G NPN

Figure 2 presents the architecture of TSN over 5G with the components introduced in the previous Message Sequence Diagram. The block including the whole 5G NPN and TSN

Radio parameter	impact on traffic
Modulation and Coding Scheme (MCS)	Higher values increase throughput and better use resources if radio propagation are favorable, otherwise causes errors , delay and losses
Mode of the Radio Link Control (RLC)	Can enforce re transmission in case of losses but produce higher latency,
Timers to control re transmission with Acknowledge RLC mode (AM)	Limit the number of re-transmissions to balance reliability-latency
Proactive scheduling in the MAC layer	Reduce latency by allocating resources in advance
Explicit assignment of resources to one UE in the MAC / PHY layer	Can increase throughput and reliability for one UE but excess could be unfair for other users and reduce system capacity

TABLE I: Configurable parameters in 5G NPN radio

translators is called the 5G Bridge in the 3GPP standards defining TSN integration in 5G networks. Our challenge is to develop a smart component to keep the configuration of the 5G Bridge (the enhanced 5G Network) and the TSN bridges and endpoints with coherent configurations that ensure the requirements of the deterministic communication at any time. Such *smart controller* will get in advance the information on the expected KPIs for the traffic to be injected in the network and will interact with the RAN and the 5G Core (as a TSN Application Function) to customize the 5G NPN in order to support such KPIs.

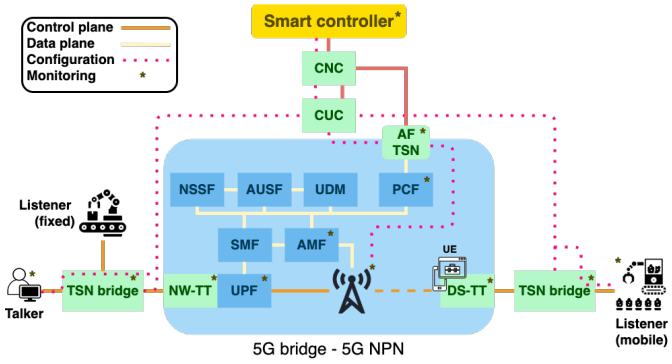

Fig. 2: Smart configuration of TSN over 5G NPN

The dynamic configuration of the 5G NPN in a factory mainly concerns to radio parameters. However, a few options are also available in the core network through the 5G PCF (Policy Control Function). Tables I and II list some configurable radio and 5G core parameters, as well as their potential impact on the quality perceived by the users. The values of all these parameters have a joint impact to achieve low latency or low jitter for TSN-like traffic, and thus, the right combination is crucial for success.

Apart from the common 5G NPN configuration, each TSN translator can request specific QoS per port in order to accommodate the specific scheduling rules of such port. Such QoS is obtained with specif QoS in the 5G flows requested to the 5G network. The composition of both configurations, the 5G NPN and TSN translators, produces the configuration of the 5G-Bridge.

Core parameter	impact on traffic
Priority, preemption vulnerability	Indicates if a service data flow may lose the resources assigned to it in order to admit a service data flow with a higher priority level
Preemption capability	A service data flow may get resources that were already assigned to another service data flow with a lower priority level
Guaranteed bitrate for uplink or downlink	Guarantee for some flows will impact others

TABLE II: Configurable parameters in the 5G NPN core

The ambition behind our research work is to build the smart controller, represented in Figure 2, covering both domains, TSN and 5G, in a coherent way that ensures the configuration of the whole 5G Bridge on time to address the new traffic patterns advertised by the end points plus the existing traffic in the network. The objective is to keep the values of the global KPIs of the TSN traffic as close as possible to the expectations by the endpoints. To do so, the smart controller performs continuous monitoring, prediction of deviation of KPIs and re-configurations of the 5G Bridge.

In general, the ability to control the network without human interaction is called *zero touch* management in the literature. The use of intelligent techniques towards zero touch management of 5G networks has been already proposed [2], [3], [4]. In this paper we propose a novel approach based on learning automata to address the challenge.

III. Learning Automata Approach

Learning have been applied to wireless and mobile networks to solve different problems, which ranges from adaptive rate controllers that adapt to the radio link conditions [5], [6], deep learning for IoT into the edge [7], routing problems [8] to classification problems in [9]. Among the most used learning approaches are machine learning (e.g. in [9]) or automata learning (e.g. in [5], [10], [8]).

Automata learning, also known as model learning, refers to techniques that produce a behavioral model of the system under learning (SUL) represented by some kind of automaton. We can distinguish two main approaches, namely passive and active automata learning. In the former, the automaton is generated from a finite set of system traces. In the active approach, the automaton is incrementally constructed based on queries to oracles that determine whether the SUL and the learned model (called hypothesis) are equivalent. The difficulty to obtain oracles make the active approach unfeasible in some cases. Independently of the algorithm or the underlying formalism, we can affirm that the automata learned is, in the context of industry, a *digital twin* [11] of the network that can be used to simulate and test new configurations.

Compared with other machine learning methods, automata learning has several advantages. Firstly, the learned model can interact with a partially unknown environment. In addition, we can use model-based analysis, such as model-based testing (MBT) [12] or model checking [13], to check properties on the real system. Finally, as we have previously commented, it is possible to generate different kinds of automata that

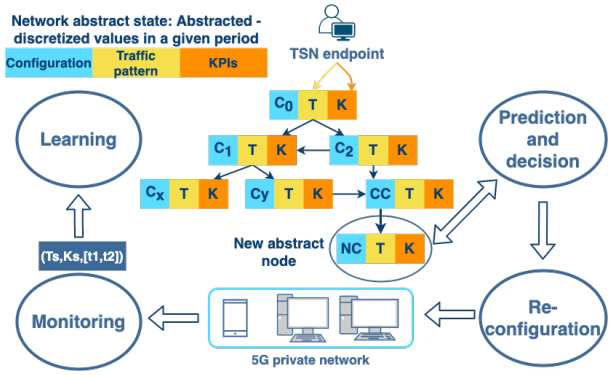

Fig. 3: Overall approach

can capture discrete input/output relations or event timing or stochastic relations.

Figure 3 shows how our framework works. It consists of four different phases that make use of automata learning and model-based testing approaches. The combination of both techniques is not new. On the one hand, automata learning may use MBT to implement equivalence checking between the SUL and the learned model and, on the other, learning may support the generation and refinement of models for MBT [14].

The objective of our approach is to find configurations of a 5G private network that can support the KPIs required by different TSN traffic flows. Initially, we construct (or learn) an automaton that describes how the network KPIs are affected by different network configurations and TSN traffic patterns. This automaton is learned from a set of *traces*, each one containing a sequence of 5G network states observed at certain time instants. An automaton state, called network abstract state in Figure 3, consists of 3 main components: a network configuration (e.g. parameters in Tables I and II, like MCS), a traffic pattern (e.g. VLAN priority, scheduling window period and phase, traffic shaping credit), and the KPIs values for this configuration and traffic pattern (e.g. throughput, delay). Observe that the two first components of the abstract state constitute the automaton inputs to be learned, and the third one is the expected automaton output. In the prediction and decision phase, we use the learned automaton to predict the network KPIs. A MBT algorithm explores the automaton states to find a state that matches the TSN flows and the required KPIs. This state, if it exists, determines a suitable network configuration. We followed a similar approach in [15] to synthesize the maneuvers of a dam to avoid flooding. Next, in the re-configuration phase, we apply the configuration obtained in the previous step to the real 5G private network in which the real TSN flows are running. Then, in the monitoring phase, we observe the behavior of the real network to determine whether this behavior is the predicted. We can use the monitored information to carry out a new learning phase. In this way, if the network does not behave as predicted, we can refine the automaton by learning new traces. In contrast, if the network behaves as expected, we reinforce that the model is accurate.

To simplify the presentation of the learning algorithm and the example, we simplify the problem to learn traces produced by a two TSN end points. However, a real 5G NPN deployment can include dozens of base stations (gNB in 5G terminology) and hundred of TSN end points.

A. Construction of the learned automaton

In this section, we describe the algorithm to construct an automaton from a finite set of traces $\mathcal{T} = \{\pi^1, \cdots, \pi^m\}$ observed during m executions of a given 5G private network which, in the following, we denote with $\mathcal{S}$.

Each trace $\pi = \pi_0 \cdot \pi_1 \cdot \ldots \cdot \pi_{n-1} \in \mathcal{T}$ is a finite sequence of *observed states* π_i during a network execution or an event fired by the environment towards the network. Each observed $\mathcal{S}$-state is a tuple $\langle ts, conf, ((it, ot), (ik, ok)) \rangle$ where the first component $ts \in \mathbb{R}$ is the timestamp where the state has been observed; the second component $conf$ is the configuration and the third component is a pair of the form $((it, ot), (ik, ok))$ corresponding to a stream between two end points using the network. Thus, each (it, ot) contains the intent and observed traffics of the corresponding stream. In particular, $it \in \mathbb{R}^3$ contains the minimum and maximum expected traffic values and, in case of error, the interval size to group the incorrect traffic values, and $ot \in \mathbb{R}$ is the real traffic measured at time ts. Similarly, (ik, ok) corresponds to the intent and observed KPIs for the stream. Thus, if $\mathcal{M}$ is the set of all network KPIs to be monitored, for all $m \in \mathcal{M}, ik(m) \in \mathbb{R}^3$ is a 3-tuple containing the minimum and maximum expected values for the magnitude m, and the size of intervals to group the incorrect values measured. Finally, $ok \in \mathcal{M} \to \mathbb{R}$ is a function that associates each KPI in $\mathcal{M}$ with the real value read at time ts.

In this paper, we assume that the possible events are $config(c)?$, $join(it, ik)?$ and $leave?$, where $config(c)?$ is a message for $\mathcal{S}$ to change its configuration to $c \in \mathcal{C}$. Similarly, $join(it, ik)?$ is a message for $\mathcal{S}$ to initiate a new session with the intent traffic and KPIs given by it and ik, and finally, $leave?$ is used to close the session.

Algorithm 1 takes as input a set $\mathcal{T} = \{\pi^1, \cdots, \pi^m\}$ of traces produced by the observation of $\mathcal{S}$ and returns an automaton $\mathcal{A}_m$ able to accept all the behaviors in $\mathcal{T}$. To do this, it creates an initial automaton $\mathcal{A}_0$ and uses function $compose$, defined in Algorithm 2, to successively expand this automaton incorporating the behavior displayed by each trace in $\mathcal{T}$.

Algorithm 1: Construction of the automaton accepting a set of traces $\mathcal{T}$

Data: A set of traces $\mathcal{T} = \{\pi^1, \cdots, \pi^m\}$
Result: An automaton $\mathcal{A}$ that accepts all traces of $\mathcal{T}$
1 $\mathcal{A}_0 := \langle l_{init}, \{l_{init}\}, \emptyset, \emptyset, f_0 \rangle$;
2 **for** $i := 1 \ldots m$ **do**
3 $\quad \mathcal{A}_i := Compose(\mathcal{A}_{i-1}, \pi^i)$;
4 **return** $\mathcal{A}_m$;

The role of function $compose$ in Algorithm 2 is to extend a given automaton $\mathcal{A}$ so that it is able to accept a given

trace $\pi = \pi_0 \cdot \ldots \cdot \pi_{n-1}$. In the algorithm, automata are tuples such as $\langle l_{init}, Locs, ELocs, Arcs, f_{\mathcal{A}} \rangle$ where l_{init} is the initial location; $Locs$ is the set of all automaton locations that correspond to *correct* states (both traffic and KPIs have the intent values); $ELocs$ is the set of error locations, meaning that when the system reaches them either traffic or some KPIs have invalid values; $Arcs$ is the set of automaton edges of the form (l, e, l'), where l/l' are, respectively, the source/target locations of the arc, and e is its label. We only write the edge label when the transition has been produced by an event, otherwise, we use symbol "-". Finally, function $f_{\mathcal{A}}$ associates each location with the 3-tuple $(conf, ik, ok)$ containing the configuration, intent traffic and KPIs applied when the system $\mathcal{S}$ is at this location.

Given $\mathcal{A} = \langle l_{init}, Locs, ELocs, Arcs, f_{\mathcal{A}} \rangle$, we make use of the following notations and functions to simplify the description of Algorithm 2. Sets $\mathcal{A}(locs)$, $\mathcal{A}(arcs), \cdots$ (as in lines 10,11,12) refer to the components $Locs$, $Arcs, \cdots$ of $\mathcal{A}$. Function $loc(\mathcal{A})$, as in line 5, is the set of all locations of $\mathcal{A}$, i.e., $loc(\mathcal{A}) = Locs \cup ELocs$. Given a set of locations, function get (lines 5 or 17) returns the location, if it exists, having exactly the configuration, and intent traffic and KPIs given in its parameters. If the location does not exist, get returns $None$. Function $newLocation(\mathcal{A})$ (for example in line 9) returns a new location name to be used as a new automaton state. Notation $\mathcal{A}(locs) := \mathcal{A}(locs) \cup c, \mathcal{A}(arcs) := \mathcal{A}(arcs) \cup c, \cdots$ (e.g. in lines 10,11 and 12) is used to add c to the corresponding automaton components.

Algorithm 2 proceeds as follows. First, it initiates the current location $cloc$ to l_{inic} and the current configuration $cConf$ to the unknown value $\perp$ in line 1. Then, it iterates through all trace $\mathcal{S}$-states π_i checking if the current automaton location accepts π_i. In case it does not, the algorithm creates a new location to force the automaton to accept the state. When all the trace states have been traversed, the new automaton built is returned in line 52.

The $switch$ sentence of line 3 considers all possible cases for state π_i. Lines 4-12 deal with the case that π_i is a message to initiate a new session. If the automaton already contains a location loc' representing the same request, the current location is updated to loc'. Otherwise, a new location is created, and the automaton is conveniently updated (lines 9-12). Lines 13-24 manage the case when π_i is a request to update the configuration. If the current location is the initial one, the request is discarded. Otherwise, variable $cConf$ records the new configuration, and the algorithm searches for a location with the new configuration and the same values for the intent traffic and KPIs as the current location. If there exists such a location loc' in $\mathcal{A}$, the current location is updated to loc'. Otherwise, a new location with the required values is added to the automaton (lines 18-22). The case when π_i is a request to leave the session (lines 25-27) is simpler. It only updates the automaton to start again from the initial location l_{init}.

The most challenging case is when π_i contains a tuple corresponding to a network $\mathcal{S}$-state observed during a session (lines 28-51). We first analyze, using function $correct$, whether the observed traffic and KPIs fit the intent values (line 29). If it is so, the algorithm searches for a location loc' in the automaton with the configuration and intent traffic and KPIs stored in π_i (line 30). If this location loc' is found, the automaton is updated linking the current location $cloc$ with loc', and the current location of the algorithm jumps to loc' (lines 31-33). If no location loc' is found, a new location to match state π_i is added to the automaton (lines 34-39). If, on the contrary, the observed values of π_i does not meet the intent ones, the algorithm uses function $error$ to calculate the deviation degree (we use the third parameter of it and ik to do this) (line 41). Function $error$ returns a new pair of intent values representing how far π_i is from reaching the intent values it and ik. Next, the algorithm proceeds as in the previous case. If the automaton already contains a location loc' with the new intent traffic and KPIs (and the current configuration) (lines 43-45), the current location of the algorithm jumps to loc'. Otherwise, a new location is created, with the values needed to accept π_i.

IV. Example

Now, we show how the algorithm incrementally generates a network automaton by learning a small subset of traces obtained from the testbed presented in the next section. In particular, we use 3 traces that track the KPIs of a TSN stream between two end points. Each trace represents different network configurations and traffic patterns. In addition, in each trace the network is reconfigured with different MCS values (from 0 to 10 in steps of 2). Clearly, learning 3 traces does not produce a complete digital twin of the 5G NPN network and more traces must be learned.

As described in the previous section, specifically the construction of the learned automaton, traces register 3 types of events: `join` and `leave` are thrown one end point when it starts or stops sending data, and `config` is triggered to change the network configuration. Figure 4 shows the automaton after the first trace has been partially learned. We use some functions (e.g. `initStructures` and `initCnx`) to make the model clearer. In the first transition, from `initial` to `idle`, the automaton initializes all internal structures. Then, the TSN end point joins to the network with some traffic requirements, that are obtained by calling `initCnx` function. From this state, the automaton fixes a network configuration and transits to `Conf1`. When the algorithm completely learns the first trace, we obtain an automaton with the states and transitions shown on the right side of Figure 8, where each state represents a different network configuration and the relation between the intended parameters and the observed ones. The full automaton shown in Figure 8 is the result of learning the 3 traces. The structure of states and transitions reveals that, in the 3 traces learned, the TSN streams start and end with `mcs = 0`, the automaton reflects that it is only possible to join and leave the stream with this `mcs` value. Clearly, the algorithm should learn more traces in which it is possible to join and leave the stream with different `mcs`

Algorithm 2: *Compose*

Data: An automaton $\mathcal{A} = \langle l_{init}, Locs, ELocs, Arcs, f_{\mathcal{A}} \rangle$ and a trace $\pi = \pi_0 \cdot \ldots \cdot \pi_{n-1}$
Result: Automaton $\mathcal{A}$ extended to accept π

$cloc := l_{init}$; $cConf := \bot$;
for $i := 0 \ldots n-1$ **do**
  switch π_i **do**
    case $join(nit, nik)?$ **do**
      $loc' := loc(\mathcal{A}).get(cConf, nit, nik)$;
      if $loc' \neq None$ **then**
        $cloc := loc'$
      else
        $nloc := newLocation(\mathcal{A})$;
        $\mathcal{A}(locs) := \mathcal{A}(locs) \cup \{nloc\}$;
        $\mathcal{A}(arcs) := \mathcal{A}(arcs) \cup \{(cloc, join(nit, nik)?, nloc)\}$;
        $\mathcal{A}(f_{\mathcal{A}}) := \mathcal{A}(f_{\mathcal{A}}) \cup \{(nloc, (cConf, nit, nik))\}$;
    case $config(nc)?$ **do**
      if $cloc \neq l_{init}$ **then**
        $cConf := nc; (conf, it', ik') := f_{\mathcal{A}}(cloc)$;
        if $conf \neq nc$ **then**
          $loc' := loc(\mathcal{A}).get(nc, it', ik')$;
          if $loc' = None$ **then**
            $nloc := newLocation(\mathcal{A})$;
            $\mathcal{A}(locs) := \mathcal{A}(locs) \cup \{nloc\}$;
            $\mathcal{A}(arcs) := \mathcal{A}(arcs) \cup \{(cloc, -, nloc')\}$;
            $\mathcal{A}(f_{\mathcal{A}}) := \mathcal{A}(f_{\mathcal{A}}) \cup \{(nloc, (nc, it', ik'))\}$;
          else
            $cloc := loc'$
    case $leave?$ **do**
      $\mathcal{A}(arcs) := \mathcal{A}(arcs) \cup \{(cloc, -, l_{init})\}$;
      $cloc := l_{init}$
    case $\langle ts, conf, ((it, ot), (ik, ok)) \rangle$ **do**
      if $correct((it, ot), (ik, ok))$ **then**
        $loc' := loc(\mathcal{A}).get(conf, it, ik)$
        if $loc' \neq None$ **then**
          $\mathcal{A}(arcs) := \mathcal{A}(arcs) \cup \{(cloc, -, loc')\}$;
          $cloc := loc'$;
        else
          $nloc := newLocation(\mathcal{A})$;
          $\mathcal{A}(locs) := \mathcal{A}(locs) \cup \{nloc\}$;
          $\mathcal{A}(arcs) := \mathcal{A}(arcs) \cup \{(cloc, -, nloc)\}$;
          $\mathcal{A}(f_{\mathcal{A}}) := \mathcal{A}(f_{\mathcal{A}}) \cup \{(nloc, (conf, it, ik))\}$;
          $cloc := nloc$;
      else
        $(it', ik') := error((it, ot), (ik, ok))$;
        $loc' := loc(\mathcal{A}).get(conf, it', ik')$;
        if $loc' \neq None$ **then**
          $\mathcal{A}(arcs) := \mathcal{A}(arcs) \cup \{(cloc, -, loc')\}$;
          $cloc := loc'$
        else
          $eloc := newLocation()$;
          $\mathcal{A}(elocks) := \mathcal{A}(elocks) \cup \{eloc\}$;
          $\mathcal{A}(arcs) := \mathcal{A}(arcs) \cup \{(cloc, -, eloc)\}$;
          $\mathcal{A}(f_{\mathcal{A}}) := \mathcal{A}(f_{\mathcal{A}}) \cup \{(eloc, (conf, it', ik'))\}$;
          $cloc := eloc$;
return $\mathcal{A}$;

values. In this case, the automaton shown in Figure 8 will be extended with new transitions but not new states.

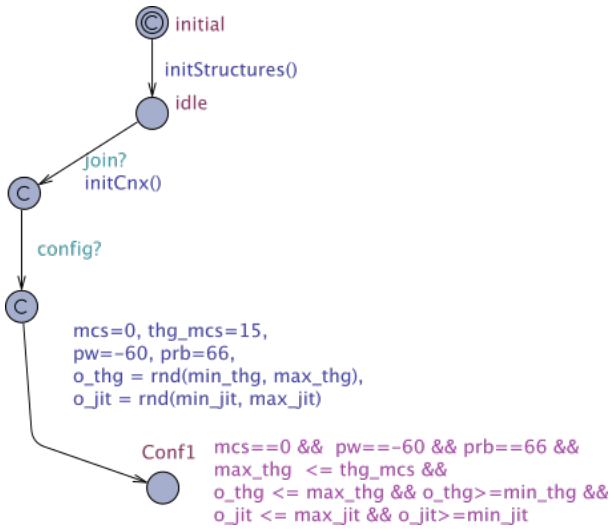

Fig. 4: Network automaton - 1 trace partially learned

The automaton shown in Figure 8 has a non-deterministic nature. For instance, after receiving a configuration request in the `idle` state, it can set 3 different network configurations that lead to different states (`Conf1`, `Conf7` or `Conf13`). Thus, for a given traffic requirement, more than one network configuration may be possible. MBT is in charge to find (if it exists) a state (or equivalently a network configuration) that satisfies the traffic requirements. To this end, MBT algorithm has to exhaustively explore the state space of the network automaton in parallel with an automaton, which represents the TSN end point, that sets up the traffic requirements and stimulates the network automaton with the `join`, `leave` and `config` events.

V. Testbed and Results

The evaluation of the proposal for automata learning based configuration has been carried out in a 5G testbed at University of Malaga. Figure 5 depicts the whole TSN over 5G testbed. On the one hand, the blue part represents the fixed TSN network, including two Relyum TSN endpoints and one Relyum TSN bridges. On the other hand, the green part shows the TSN over 5G NPN, combining the TSN bridge with the Keysight UXM5G E7515B mobile network emulator and the Keysight anechoic chamber where the 5G antennas (28GHz) and 5G commercial smarphone are located. In the middle of the testbed we connect a computer for monitoring and control part of both domains.

One of the cornerstones of the integration of 5G and TSN networking resides in the definition mechanisms to enable synchronization between the 5G and the TSN network time domains. That synchronization is required as the traditional TSN synchronization is dependent of an actual physical link where HW timestamping is used for very accurate time alignment. However, in a radiated environment as 5G, such

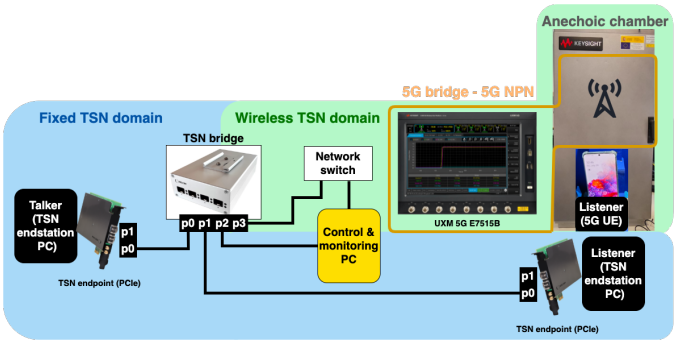

Fig. 5: TSN over 5G testbed

synchronization needs to be provisioned by different means. As there are not currently available TSN translators, in our initial approximation we have not yet addressed the time synchronization at the mobile node. However, we have adopted the approach to characterize the downlink traffic metrics as received by a mobile device where the traffic source is generated in a TSN network adapter. The traffic traverses a TSN bridge that is configured with synchronized time windows consistent with the traffic source. At a later stage, the intention is to develop a software translator based on the timestamping capabilities of the networking cards and incorporate commercial components for translation at both the network and mobile side as they become available.

The control and monitoring computer allows to access to the dashboard in order to configure the TSN bridge and endstations. Additionally, it can retrieves the monitoring files of different parts of the testbed. In order to have a first version of the monitoring and configuration part, we have developed a toolkit based on the Keysight TAP automation framework that automatically configures the 5G emulator, runs a test scenario simulating the users traffic pattern and captures the traffic in different interfaces (step 1). Then, this information is processed using a Java program that generates a log file that summarizes the behavior of the network KPIs over time (step 2). These log files are used to produce a sequence of abstract states in order to learn how the network KPIs evolved in the different configurations. The files are used to build the UPPAAL models of the 5G private network described in section Example.

As per reference configurations for industrial traffic types, we have used different payload sizes (100 bytes and 1 Kbytes and multiple cycle periods for testing. The monitorization performed at different parts of the system allows to obtain a consistent results. In this case the testing consist in the realization of several single test, which are performed considerable times, in the same manner, in order to get the final results.

Figure 6 and figure 7 are representative as a example of these results. The interval ID represents a four seconds lapse, when the average calculation of each KPI is executed. The blue and orange dots shows the results obtained for throughput and jitter in these interval IDs. In addition the MCS value is also represented in gray, where is observed worse results (lower

throughput and higher jitter) due to some packet loss when its value is 10.

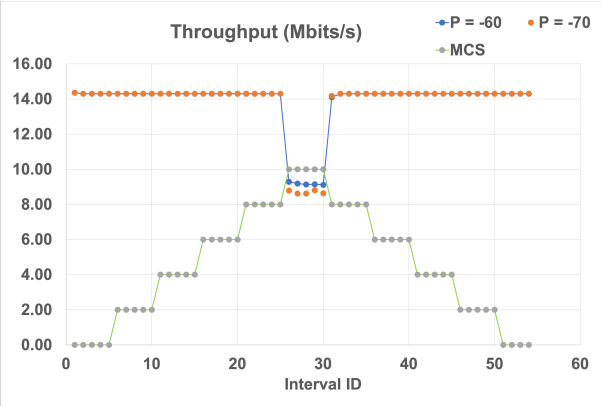

Fig. 6: Throughput results

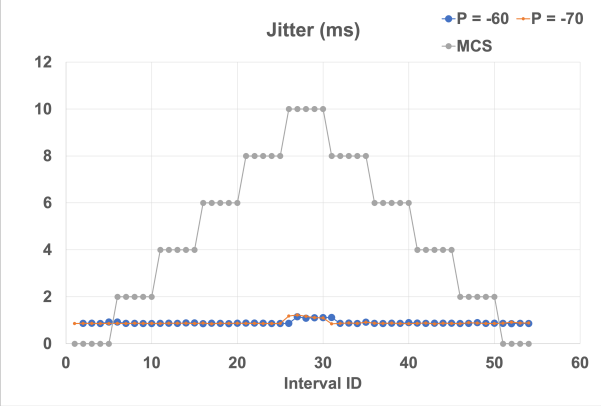

Fig. 7: Jitter results

VI. Conclusions

In this article we have presented a novel approach to zero touch self configuration of private 5G networks to transport TSN traffic in industry. The use of automata learning to record and abstract behaviour of network as state machine offers interesting features, like the ability to search potential configurations for a given traffic and expected KPIs using (automated) model based testing. Such features is very relevant in the direction of providing explicable decisions in such critical networks. In the experiments to build an example automata learned from the network we select three parameters to be configured in the radio access network and three different patterns of TSN traffic. The results with a realistic testbed confirmed the suitability of the approach As future work, we are expanding the learning algorithms and the testbed to support more configuration parameters, more users and a bigger 5G NPN in order to make real demonstrations with industrial companies.

Acknowledgment

This work is supported by EVOLVED5G and AFFORDABLE5G projects (European Union Horizon 2020) under grant agreements No.101016608 and No.957317 and by RFOG Project (Spanish Goverment) under grant agreement RTI2018-099777-B-I00.

References

[1] C. Mannweiler, B. Gajić, P. Rost, R. S. Ganesan, C. Markwart, R. Halfmann, J. Gebert, and A. Wich, "Reliable and deterministic mobile communications for industry 4.0: Key challenges and solutions for the integration of the 3gpp 5g system with ieee," 2019.

[2] I. S. Comsa, S. Zhang, M. E. Aydin, P. Kuonen, Y. Lu, R. Trestian, and G. Ghinea, "Towards 5g: A reinforcement learning-based scheduling solution for data traffic management," *IEEE Transactions on Network and Service Management*, vol. 15, pp. 1661–1675, 2018.

[3] J. Kaur, M. A. Khan, M. Iftikhar, M. Imran, and Q. Emad Ul Haq, "Machine learning techniques for 5g and beyond," *IEEE Access*, vol. 9, pp. 23 472–23 488, 2021.

[4] M. S. Mollel, A. I. Abubakar, M. Ozturk, S. F. Kaijage, M. Kisangiri, S. Hussain, M. A. Imran, and Q. H. Abbasi, "A survey of machine learning applications to handover management in 5g and beyond," *IEEE Access*, vol. 9, pp. 45 770–45 802, 2021.

[5] T. Joshi, D. Ahuja, D. Singh, and D. P. Agrawal, "Sara: Stochastic automata rate adaptation for ieee 802.11 networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1579–1590, 2008.

[6] A. Sarigiannidis, P. Nicopolitidis, G. Papadimitriou, P. Sarigiannidis, and M. Louta, "Using learning automata for adaptively adjusting the downlink-to-uplink ratio in ieee 802.16e wireless networks," in *2011 IEEE Symposium on Computers and Communications (ISCC)*, 2011, pp. 353–358.

[7] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, 2018.

[8] J. Sen and K. Vinodha, "A Weighted Learning Automata-Based Multicast Routing Protocol for Wireless MANET," *INT. J. OF ENGINEERING RESEARCH & TECHNOLOGY*, vol. 2, no. 6, 2013.

[9] J. E. Preciado-Velasco, J. D. Gonzalez-Franco, C. E. Anias-Calderon, J. I. Nieto-Hipolito, and R. Rivera-Rodriguez, "5g/b5g service classification using supervised learning," *Applied Sciences*, vol. 11, no. 11, 2021.

[10] P. Nicopolitidis, G. I. Papadimitriou, A. S. Pomportsis, P. Sarigiannidis, and M. S. Obaidat, "Adaptive wireless networks using learning automata," *IEEE Wireless Communications*, vol. 18, no. 2, pp. 75–81, 2011.

[11] H. X. Nguyen, R. Trestian, D. To, and M. Tatipamula, "Digital twin for 5g and beyond," *IEEE Communications Magazine*, vol. 59, no. 2, pp. 10–15, 2021.

[12] M. Utting and B. Legeard, *Practical Model-Based Testing - A Tools Approach*. Morgan Kaufmann, 2007.

[13] G. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, September 2003.

[14] B. Aichernig, W. Mostowski, M. Mousavi, M. Tappler, and M. Taromirad, *Model Learning and Model-Based Testing*, ser. LNCS. Springer Nature, Jul. 2018, pp. 74 – 100.

[15] M. Gallardo, P. Merino, L. Panizo, and A. Salmerón, "Integrating river basin dsss with model checking," *Int. J. Softw. Tools Technol. Transf.*, vol. 20, no. 5, pp. 499–514, 2018.

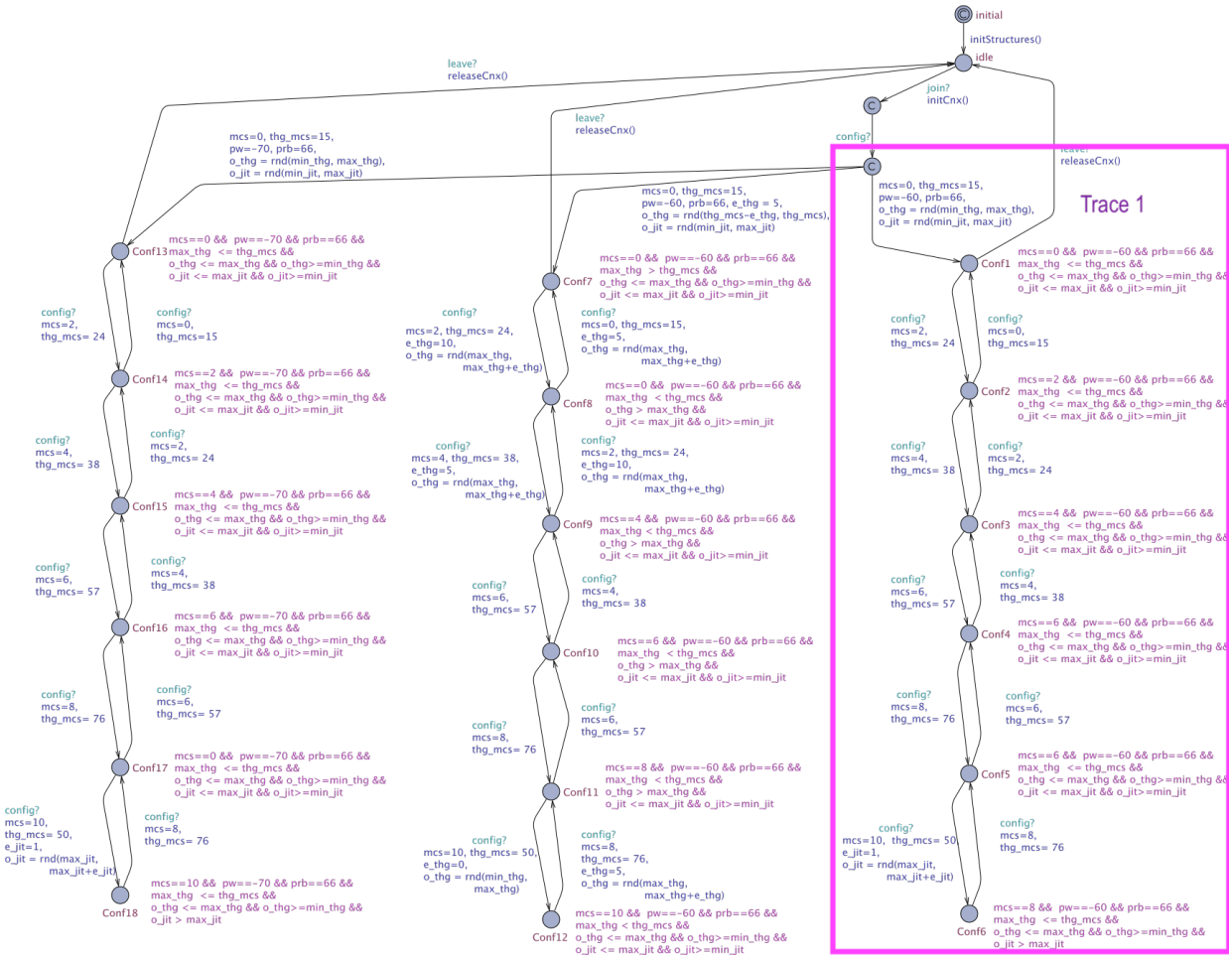

Fig. 8: Network automaton - 3 traces learned