Self-Configuration of IEEE 802.1 TSN Networks

Marina Gutiérrez, Astrit Ademaj, Wilfried Steiner
{marina.gutierrez, astrit.ademaj, wilfried.steiner}@tttech.com
TTTech Computertechnik AG
Vienna, Austria

Radu Dobrin, Sasikumar Punnekkat
{radu.dobrin, sasikumar.punnekkat}@mdh.se
Mälardalen University
Västerås, Sweden

***Abstract*—Configuration processes of real-time networks are costly both in terms of time and engineering effort and require the system to be shutdown during the reconfiguration phase thus resulting in significant down time as well. The convergence of IT/OT technologies is bringing a whole world of possibilities for the configuration and management of real-time networks for the automation industry. With software defined networking (SDN) features like the separation of the data and control plane and standards like IEEE 802.1 developed with the goal of adding deterministic capabilities to traditionally dynamic switched Ethernet networks, the plug and play paradigm is almost around the corner. In this paper, we go one step further and start looking into the self-configuration of real-time networks. To achieve that we propose to introduce a Configuration Agent in the network, an entity that continuously monitors the network to detect changes and automatically update the configuration to adapt to such changes while maintaining desired quality of service. We present here the complete framework for the Configuration Agent that will provide self-configuration capabilities to real-time networks. The proposed framework has IEEE 802.1 as its core, but also shows how the set of standards need to be extended in order to achieve the self-configuration requirements. Concretely we examine the role of existing communication protocols like NETCONF and OPC-UA and propose the essential ingredients (managed objects) for a YANG model for the learning aspects in the bridges, including different working modes.**

I. Introduction

Emerging trends like Industrial Internet of Things (I-IoT), or Real-Time Internet of things (RT-IoT) are bringing into focus the need for efficient network management in Operation Technology (OT) fields like automotive or industrial automation. In those contexts the network is typically heavily engineered, tailored to satisfy the real-time requirements imposed by the physical environment. The configuration is then seen as a one-time event that occurs during the initialization of the network and any change during run-time requires often manual reconfiguration. Whereas this characteristic has not been such a problem in the past due to the static nature of OT networks, the online reconfiguration of the network is now becoming an urgent need as we are seeing larger and larger networks, transporting heterogeneous traffic that still require real-time guarantees.

To address the problem of the online reconfiguration of real-time networks, mechanisms from the Information Technologies (IT) field are being used in OT use-cases [1]. One of the main exponents of this IT/OT integration are the set of standards being developed by the Time-Sensitive Networking (TSN) task group of IEEE 802.1. These standards are adding real-time capabilities to regular switched Ethernet with features like system wide clock synchronization, scheduled traffic, redundancy or frame preemption. This, together with the already omnipresence of Ethernet in IT domains will make TSN to play the main role in the next generation of industrial networks. The focus of this paper is then how to add self-configuration capabilities to TSN networks.

978-1-5090-6505-9/17$31.00

The management and configuration of a network is adressed by TSN, although not fully covered. IEEE 802.1Qcc [2], currently under development, presents three different configuration models, from a fully distributed model approach, in which devices adjust their configuration with the information that they have locally available, to a fully centralized model in which there is a central entity that has a global view of the network and adjusts the configuration according to system-wide requirements. Additionally, the standards define so called "managed objects" for all their configurable aspects. This managed objects are meant to be adjusted to achieve the desired behavior. Although these configuration options provide many of the necessary elements, the self-configuration is not in scope of TSN. Thus, the contributions of this paper are i) the study of the requirements of a self-configuration approach for TSN networks and ii) the proposal of a complete self-configuration framework for TSN networks based on the notion of the Configuration Agent.

The Configuration Agent, presented in [3], describes the necessary elements for the self-configuration of a time-triggered network. The idea is to introduce intelligence in the network so it is able to first learn the characteristics of the traffic by means of monitoring and then to adapt the configuration of the network to meet quality of service (QoS) aspects, using the learned information as input. While the learning aspects have been studied in previous publications [4], in this paper the focus is on a concrete view of the Configuration Agent, as we discuss implementation details such as its architectural needs and the communication protocols used to transport its specific traffic. We show how the set of TSN standards is a good match for the Configuration Agent as it fulfills some of its requirements by design, and we also show how some aspects needs to be extended. Concretely we propose to introduce learning capabilities in the switches, with different working modes and present the managed objects that this features would need. Additionally, we also discuss

candidates for the communication protocols used to transport configuration information: NETCONF and OPC-UA.

The rest of the paper is structured as follows: we start in the Section II by deriving the requirements that a self-configuration approach shall satisfy. In Section III we explore the different solutions to address the self-configuration problem, both existing standards and protocols and new specific learning characteristics defined in this paper. Then in Section IV we map the solutions to the requirements and propose a complete self-configuration framework. Finally, we conclude the paper with some words on related work in Section V and conclusions, Section VI.

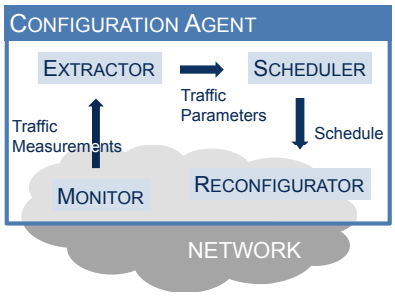

Fig. 1. Configuration Agent Overview [3].

II. Self-Configuration Requirements

The solution that we propose to ease the self-configuration of real-time networks is to introduce intelligence to the network in the form of a Configuration Agent [3]. The Configuration Agent is an autonomous entity that continuously monitors the network and extracts relevant information that will in time be used to modify the current configuration of the network. The functional elements that forms the Configuration Agent, depicted in Fig. 1, are the Monitor, Extractor, Scheduler and Reconfigurator and its functionality has been already defined in previous publications [4]. The Monitor gathers raw data from the network traffic, such as interarrival times of messages, and the Extractor uses that data to distill relevant traffic parameters. This process is the learning phase of the Configuration Agent. Once the Configuration Agent has enough information the Scheduler can produce a new schedule. Finally, the Reconfigurator is in charge of distributing and enforcing the new configuration in the network.

A. Architecture

An example target network for the Configuration Agent is depicted in Fig. 2. It is composed of end-stations and switches and links connecting them to each other. Next we explain how the functionality of the Configuration Agent is divided in such a network.

It is straightforward to conclude that the monitoring activities should be performed in the switches. Ideally, all the switches in the network are constantly monitoring the traffic that they forward. However to have that capability on every switch in the network could be challenging, specially in large

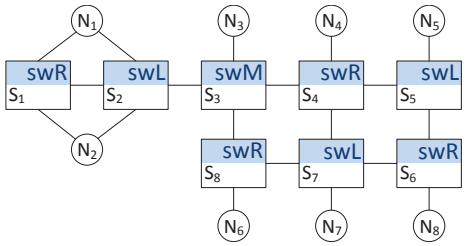

Fig. 2. Network Architecture Example.

legacy networks. Thus, our approach should be able to work with the sufficient number of monitors in the network. How many specifically and where in the network will depend strongly on the characteristics of the network and it is an open problem that we keep as future work.

The extraction of relevant traffic parameters from the data gathered by the monitors and the generation of a new configuration are activities that require a centralized view of the network. Thus, the functionality of the extractor and scheduler should be integrated in one of the switches of the network that will act as a "masterbox". However the amount of data gathered by the monitors can scale up easily introducing a significant network overhead. To alleviate that effect some of the data can be pre-processed already in the switches, applying the concepts of fog-computing or edge computing. From now on, we will refer to the Monitor and Extractor functions performed locally in the switch as the learning capability of the switch.

Finally, the role of the Reconfigurator is more spread among the elements of the network as it is composed of two functionalities: first it needs to distribute the new configuration through the network and second the elements that conform the network should be capable of understand the new configuration and change its own configuration accordingly. One example of this new configuration is a new schedule for the network. The switch should be able to implement this schedule. Thus, unlike with the learning capabilities, our approach demands that all the elements of the network have reconfiguration capabilities.

In summary, these are the types of devices that need to exist in a self-configurable network:

- swR: Simple switch with reconfiguration capabilities. It is still allowed to have some in the network.
- swRL: Switch with reconfiguration and learning (Monitor + local Extractor) capabilities. It is mandatory to have "enough" of them in the network.
- swRM: Masterbox. Includes capabilities of learning from the received data (centralized Extractor), generating new configuration (Scheduler) and distributing it. It is mandatory to have one in the network.

This materializes in the following architectural requirements:

R1 *The network shall implement minimum one swRM switch.*
R2 *The network shall implement swRL switches.*

R3 *The network may implement swR switches.*

B. Protocols

The Configuration Agent generates specific traffic workload. In this section we describe the type of messages that the Configuration Agent introduces in the network and specify the characteristics that the protocols for this need to provide.

The traffic introduced by the Configuration Agent can be classified in two types: learning messages and configuration messages. The learning messages are the ones generated in the learning phase of the Configuration Agent. They are transporting the relevant information that has been obtained in the switches after the learning process. Among those we can also separate them in two types: notification information, e.g., a new end-station is connected to the network and it is sending its communication needs and discovery information, i.e., traffic that has been added or removed from the network without properly notifying it. The learning messages shall be sent in the network with a priority proportional to the importance of the information that they are carrying but without detriment to existing traffic. These are the requirements for learning messages:

R4 *swRL switches shall forward learning messages to the swRM containing the information learned locally in the swRL.*
R5 *The learning messages shall be forwarded according to a communication protocol that is known by the swRLs and the swRM.*
R6 *The communication protocol for the learning messages shall prevent them from interfering with existing traffic.*

The configuration messages are the ones that transport the new configuration for the network. The configuration messages sent to a certain device shall contain the specific configuration information of that device. In addition, a re-configuration protocol shall be in place that defines a safe way of updating the configuration. Thus, these are the requirements for configuration messages:

R7 *The swRM shall forward configuration messages to the swRs and swRLs in the network.*
R8 *The configuration message shall be forwarded according to a communication protocol that is known to all swR, swRL and swRM switches.*
R9 *The swR and swRL switches shall implement a re-configuration protocol that defines a safe-way of updating the device configuration.*

III. Solution Space

To address the problem of the self-configurable real-time networks, it is clear that we need to look at wide-spread and/or standardized solutions that guarantee interoperability, especially in regards to the reconfiguration capabilities. Thus, in this section we give an overview of three network mechanisms that combined cover some of the re-configuration aspects that our approach requires. However we show that the existing solutions are not enough to cover all the self-configuration requirements and we propose a model for the learning capabilities of the switches.

A. IEEE 802.1 Time-Sensitive Networking

The Time-Senstitive Networking (TSN) task group of IEEE 802.1 focuses on the development of standards to provide deterministic communications over IEEE 802 networks. TSN continues the work of the Audio Video Bridging (AVB) task group widening the scope of the applications by increasing deterministic guarantees. Now the set of standards being developed by TSN are seen as key elements that will enable the use of Ethernet in fields such as industrial automation and automotive.

The four main pillars over which TSN is built are time synchronization (with IEEE 802.1AS, that includes a profile of IEEE 1588), reliability (with redundancy provided by IEEE 802.1CB and per-stream filtering by IEEE 802.1Qci), guaranteed end-to-end latency (scheduled traffic is defined in IEEE 802.1Qbv) and resource management (configuration in IEEE 802.1Qcc and YANG models in IEEE 802.1Qcp). These characteristics make TSN standards a good fit for our self configuration approach. Thus, next we analyze the configuration aspects of TSN.

1) Configuration of TSN Networks: The standard that deals with the configuration of TSN networks is IEEE 802.1Qcc. It is an ongoing project that at the time of this writing of this paper is in draft status, but the aspects discussed here are considered stable. IEEE 802.1Qcc is an enhancement of the Stream Reservation Protocol (SRP) (IEEE 802.1Qat) designed for the resource management in networks using the Credit Base Shaper (CBS) (IEEE 802.1Qav). CBS defines two traffic classes A and B and traffic of each class is allowed to be sent as long as there is enough credit for that traffic type. In that context SRP was a simple admission protocol in which the talker announces the traffic that it will send and depending on the available bandwidth it will be granted permission to do it, or not. With the inclusion of more complex traffic shaping mechanisms in TSN such as the Time Aware Shaper (TAS) (scheduled traffic) or frame preemption, an update for the SRP was needed.

One of the main elements for the configuration of TSN networks is the User Network Interface (UNI). On the user side of this interface are the talkers and listeners. On the network side are the bridges. The idea is that the user specifies the requirement for the streams that they want to transmit without having all the details about the network. The network then analyzes this information along with network capabilities and configures the bridges to meet the user requirements. To realize this configuration paradigm, IEEE 802.1Qcc defines three different configuration models with regards to their architecture:

Fully Distributed Model: In this model the UNI is situated between the talker/listener (user) and the bridge to which it is connected (network). The user transmits its requirements and the network propagates them through the relevant paths. The

management of the bridges is performed locally just with the information that is available to that bridge. This model can be used to configure the CBS and for that the SRP can be used as UNI. One limitation of this model is the lack of a centralized view with complete knowledge of the network that makes it not suitable for the configuration of the TAS.

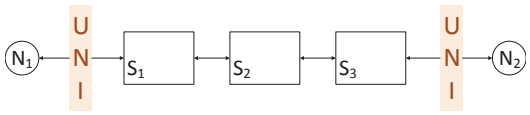

Fig. 3. IEEE 802.1Qcc Fully Distributed Model.

Centralized Network / Distributed User Model: This model comes to alleviate that limitation of the fully distributed model by introducing the Centralized Network Configuration (CNC). The UNI is still between the talkers/listeners and the bridges, but in this model the bridge communicates the user requirements directly to the CNC. The CNC has a complete knowledge of the network topology as well as the bridges capabilities and that enables it to perform more complex calculations needed to configure the TAS, frame replication and elimination or frame preemption. The management of the bridges is performed by the CNC using a network management protocol. The management of end-stations is not performed by the CNC. The CNC can exist either in an end-station or a bridge.

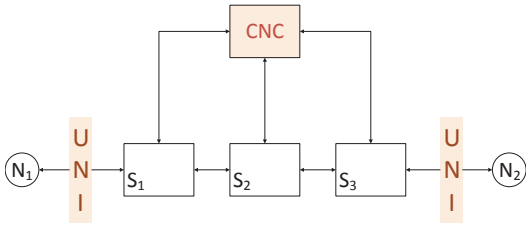

Fig. 4. IEEE 802.1Qcc Centralized Network / Distributed User Model.

Fully Centralized Model: In the two previous models, configuration of end-station was not addressed. However there are use-cases for highly critical applications, such automotive or industrial control, in which there are complex timing requirements that needs extra configuration. For those cases the notion of the Centralized User Configuration (CUC) is introduced in this model. The talkers/listeners communicate their requirements to the CUC and then the CUC exchange this information with the CNC through the UNI. The CNC and the CUC can be implemented in the same device or in separate devices. The definition of the communication protocol between the CUC and the end-stations is considered to be out of the scope if IEEE 802.1Qcc.

2) Managed Objects: As it can be seen these three models represent three different approaches of how configuration can be handled in a TSN network. The other big piece for the configuration of networks as defined in TSN are the *managed objects*. Managed objects are the configuration aspects of each functionality defined in the standards. For example, for the case of the TAS, managed objects allows to set and retrieve the state of the per port gates that handle the scheduled traffic. Managed objects can be read and modified using an appropriate configuration/management protocol in place but for now the definition of such protocols is not part of the TSN standards.

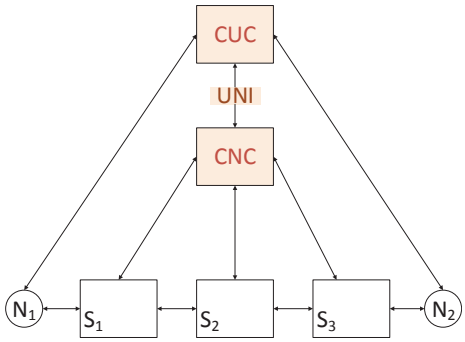

Fig. 5. IEEE 802.1Qcc Fully Centralized Model

B. NETCONF

Despite the protocol independent design of IEEE 802.1Qcc, there is an ongoing standardization effort to model managed objects using YANG. YANG is a data modeling language standardized by the IETF [5] to be used with the Network Configuration (NETCONF) protocol. NETCONF has been developed and standardized by IETF [6] with the aim of creating a configuration management protocol that alleviates some of the most common problems with existing network configuration protocols. Most of these protocols are vendor-specific and operate as command line interfaces which translates into low efficiency and low reliability. The main goal has been to move from the "network is the record" paradigm in which operators directly modify the configuration of devices to the "generate everything" approach in which a global knowledge of the network is used to generate new configurations that are pushed to the devices [7].

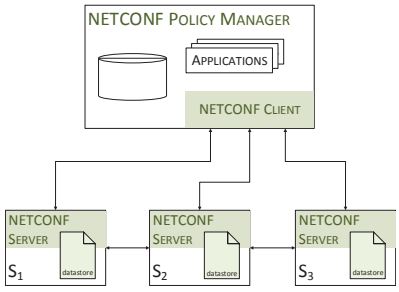

Fig. 6. NETCONF deployment.

In Fig. 6 a NETCONF deployment is depicted. The centralized architecture is needed to have a global knowledge of

the network and the Policy Manager is/act as the network manager. With regards to the NETCONF protocol we can see two roles: servers and clients. The network devices act as the configuration servers and the Policy Manager as the configuration client. NETCONF represents the configuration of a network device as a structured document that is called the "datastore". Datastores are stored in each network device and can be retrieved and edited. NETCONF supports different types of datastores: `running` datastore that contains the current configuration of the device. It should be always present. The other two datastores are optional: `start up` datastore with the initial configuration of the device. And `candidate` datastore that is a draft configuration for the device that can be edited for a later commit.

To interact with devices configuration NETCONF provides a series of operations such as `edit-config`, `get-config`, `copy-config` or `delete-config`. For large configuration datastores NETCONF also provides filtering mechanisms that allow the client to modify or retrieve just a subset of the whole configuration. NETCONF sees configuration changes as network-wide transactions, this means that if there is an error in the configuration, for example some inconsistency, the transaction will fail. The existence of those consistency checks and the all-or-nothing semantics guarantee a safe-way of updating the configuration.

C. OPC-UA

OPC UA uses a conventional client / server data acquisition mechanism, in which the user can retrieve data from nodes in an ad-hoc process. OPC UA Publish / Subscribe has recently been defined by the OPC Foundation in addition to the existing client / server mechanism and is already available in prototypes. This extension ensures that individual transmission nodes can initially publish data for several clients that have identified themselves as subscribers. This extension of the multicast data exchange will enable the communication between many sensors and the cloud as well as the coordination between machines. To transmit messages deterministically from broadcast nodes (publishers), two criteria shall be fulfilled:

A The Publish / Subscribe stack shall run in a real-time SW environment to ensure that data can be sent periodically. In order to allow optimal end-to-end latency for specific applications, OPC UA PubSub specifies the configuration option to transmit messages in a specific offset within a given period.

B The PubSub messages are to be transmitted via a deterministic network. TSN fulfills the demand for guaranteed deterministic communication and, as part of the IEEE Ethernet standard, also offers the same flexibility as OPC UA users are already used to have.

The data and configuration interface between OPC UA Pub / Sub and TSN is also defined by a workgroup of the OPC Foundation. It ensures that OPC receives UA data that requires guaranteed transmission, a deterministic path through the network. This can be achieved, among other things, by using suitable TSN configurations for the transmission of OPC UA data. The network configuration can be configured either statically (to the system design time) or dynamically. As a result, TSN provides different mechanisms for the dynamic TSN configuration (centralized, distributed and the combination of distributed and centralized approach), OPC UA specifies a defined interface to a service unit, the so-called PubSub TSN Configuration Broker (PTCB) (Fig. 7). This interface represents the TSN specific configuration mechanism of OPC UA devices transparently. All OPC nodes should therefore support this interface.

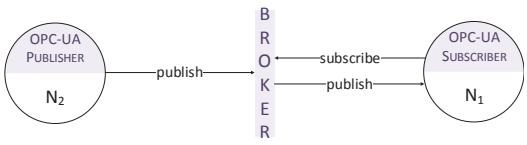

Fig. 7. PubSub TSN Configuration Broker (PTCB).

The PTCB is used to send OPC UA Publishers and Subscribers requests to dynamically configure the deterministic paths for TSN at runtime. The PTCB transmits these configuration requirements to the specific TSN configuration mechanism. If the central TSN configuration set is used, the PTCB will exchange the data with the Central Network Controller (CNC). The CNC then supplies the configurations of the nodes to the PTCB and also ensures that the TSN switches are automatically configured. The PTCB provides these configuration data, e.g. TSN Stream ID, back to the OPC UA nodes. Once the network is configured and a deterministic path is provided, the publisher starts publishing the data for real-time communication.

D. Managed Objects for the Switches Learning Capabilities

IEEE 802.1, NETCONF and OPC-UA cover many of the requirements shown in Section II. However there are requirements of the self-configuration approach that cannot be satisfied with existing solutions. Concretely, the learning capabilities assumed by the switches are currently not supported in the TSN standards. In this section we define managed objects for the Monitor and Extractor in TSN bridges.

To configure the self-configuration aspects of the networks, we propose to model the new capabilities with managed objects, the same way that configuration aspects of other standards are modeled in TSN. As it was explained in Section II-A the two Configuration Agent functions that should be implemented in the bridges are the Monitor and the Extractor. Here we have the option to model the two capabilities individually or together as a global learning capability. Modeling it as a single aspect will allow more freedom to the vendors to implement different learning algorithms, but modeling them as separate aspects will allow the CNC to have a fine grained control of the information that is and can be gathered, which would allow it to make smarter decisions. In this paper we consider the learning capability as a single aspect from the configuration point of view.

TABLE I
LEARNING MANAGED OBJECTS PER BRIDGE

name	operation supported
learningMode	RW
learningBufferSize	RW
maxLearningBufferSize	R
learningPeriod	RW
maxBurst	RW

Table I lists the managed objects for the self-configuration capabilities of the switches. There are two parameters that control the amount of learned data that the bridge can store. `maxLearningBufferSize` is the upper limit that is set by the bridge implementation. The second parameter is `learningBufferSize`, a configurable parameter that the CNC uses to control the traffic workload introduced by the learning mechanisms. It is important to notice that the learning data is the information extracted from the raw data gathered by the monitor and it is not the monitored data itself. For the monitored data, the bridge will have other implementation-specific buffer space.

The remaining parameters are related to the configuration of different learning modes. The learning modes are as follows:

- *DISCOVERY*: the learning mechanism is active and sending all information that it gathers to the CNC. The idea is to use this mode for the initial configuration of the network so the CNC can build its knowledge about the network.
- *NOTIFY*: the learning mechanism sends messages to the CNC just when it detects a change from its usual traffic. This is intended to be one of the normal modes of operation of the learning mechanism.
- *PERIODIC*: the learning mechanism sends periodic reports to the CNC. This is intended to be another normal mode of operation for the learning mechanism. In this case a change will be signaled to the CNC in the periodic report following that change. Periodic reports can be used as heart-beat to monitor the correct operation of the bridges.
- *DIAGNOSE*: the CNC requests from the bridge to send a report about some specif port/stream. It is intended to be used when the CNC notices anomalies and it can potentially detect faulty components or security breaches [8].
- *INACTIVE*: the learning mechanism is not active.

Fig. 8 depicts the learning modes satate machine. Flags `running_config` and `learning_enabled` signal the initial phase whether there is an existing configuration or if the learning mechanism is enabled. `enough_information` is the exit condition of the *DISCOVERY* mode and `something_is_wrong` is the trigger for the *DIAGNOSE* mode.

IV. SELF-CONFIGURATION FRAMEWORK

In the previous section the necessary elements for the configuration of a TSN network have been described in detail.

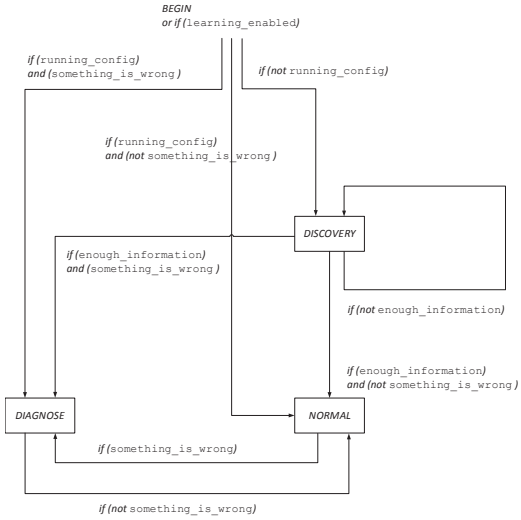

Fig. 8. Learning modes state machine.

Fig. 9 shows how these elements come together to satisfy the Configuration Agent requirements shown in Section II and form the complete self-configuration framework.

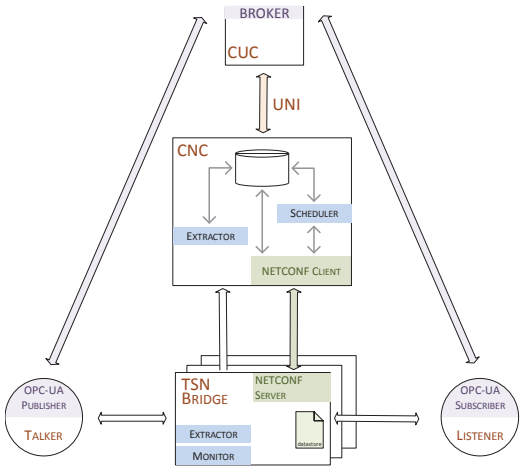

Fig. 9. Self-configuration Framework.

The centralized network models proposed in IEEE 802.1Qcc fit very well with the global view that the Configuration Agent needs to calculate the new configuration for the network. The inclusion of a CNC in these models satisfy requirement R1 from Section II. It is clear then, that the Extractor and the Scheduler should be implemented as part of the CNC. In fact, the CNC will always need to have some scheduling capability if it is meant to be used to configure the time aware shaper. The next decision is to chose between the centralized or distributed user model. Although the principles of the self-configuration can also be applied to the distributed user model, we think that the fully centralized model presents enough advantages

to favor it as it has been shown in Section III-A1.

The design choice of TSN for the network also guarantees the requirement R3 that demands reconfiguration capabilities, as any TSN compliant bridge that implements a certain standard, i.e., IEEE 802.1Qbv, can be configured in the same way. So the only issue that we have to enforce is that all the bridges implement the standards that we are interested. Once we have that, the interoperability provided by the use of the standard will do the rest.

The addition of the learning capabilities as described in Section III-D to a TSN bridge enables our self-configuration approach to fulfill requirements R2, R3, R4 and R7. For the communication protocols between the CNC and the bridges and the CUC and the end stations we chose to use NETCONF and OPC-UA respectively. The use of OPC-UA for the communication of learning messages between end-stations and the CUC satisfy requirements R5 and R6 as the PTCB alleviates the traffic workload. And the re-configuration mechanisms defined in NETCONF satisfy requirements R8 and R9.

A. Self-Configuration Workflow

Now we have defined all the elements involved in this last section we explain how we put everything together and how it will be the real workflow of the a self-configured network.

1) Initial Configuration: Given an existing legacy network to which a new CNC is connected, we assume that the network is a) not TSN or b) TSN with fully distributed configuration model. The assumption about the CNC is that it does not have previous information and that the end-stations do not actively notify the CNC about their needs. And that the CNC has the ability to assess its own knowledge about the network. Then initial self-configuration process is as follows:

1) The CNC establishes connection with the bridges using NETCONF protocol so it is aware of their capabilities.
2) The CNC activates the *DISCOVERY* mode in all the bridges that have the learning capability. It does so by modifying the bridges configuration using the NETCONF protocol.
3) The bridges send learning messages to the CNC, using the available bandwidth.
4) Once the CNC has enough information, it produces the initial configuration for the network.
5) The CNC distributes the new configuration to the bridges and the CUC. It uses NETCONF protocol to communicate the new configuration to the switches and the UNI interface to communicate it to the CUC. Part of that new configuration of the switches is the change of learning mode to one of the normal modes of operation (*NOTIFY* or *PERIODIC*).

For clarity two important aspects have been left out of the above step-by-step description of this process. The first one is in regards to step number 3). The learning messages are sent as best effort traffic using the available bandwidth. The reason for that is that without knowing enough about the network it does not make sense that the learning process interfere with existing traffic. However the question is then, would there be enough bandwidth available to transmit all the learning information? The answer to that question depends on a myriad of aspects, including not just the available bandwidth, but also the learning algorithm, both the one implemented locally in the switches and the one in the CNC, the size of the network, the number of messages, their length, their time characteristics...

TABLE II
EXAMPLE OF NETWORK UTILIZATION WITH LEARNING MECHANISM.

case	*n*	*p* (n_i)	*U*	$U + L_{BW}$
A		1 ms (15)		0.97
- small msgs (300B)	40	2 ms (13)	0.61	0.77
- variable period		3 ms (12)		0.71
B		1 ms (5)		0.73
- large msgs (1kB)	10	2 ms (3)	0.60	0.64
- variable period		3 ms (3)		0.63
C				
- small msgs (300B)	25	1 ms (25)	0.60	1.20
- same period				
D				
- large msgs (1kB)	8	1 ms (8)	0.64	0.88
- same period				

The second unanswered question is in regards to step 4). How long does it take for the CNC to receive all the learning messages? Again, this depends on many things, but in order to get an estimation we analyze the following case:

The assumptions made are based on the studied use case in [4], where all the traffic is considered periodic, with periods of 1, 2 and 3 ms. Table II shows the details of four different traffic profiles. In all of them the utilization, U is around 60%. Needles to say that if utilization of the network is already very high there will be very little bandwidth available for the learning messages to be sent, that is why we choose that utilization value. Thus, the bandwidth used by the learning mechanism, L_{BW}, should be:

$$L_{BW} < 1 - U(n, l_i, p_i) \tag{1}$$

where n is the number of messages, l_i are their length and p_i their periods. Again using a result of [4], we know that the period of a periodic message can be learned in the switch with an error lower than 1% after after 20 periods. Thus, the switch needs 20 ms to learn the period of a message whose period is 1 ms. That means that around 20-21 ms after the discovery mode has been enable the switch is ready to transmit this information to the CNC.

Now we need to start making assumptions on how the learning algorithm on the switch would send the learning messages. A smart learning algorithm could pack the information of different messages together to reduce bandwidth usage. Here, to have a worst-case scenario, we assume that the switch sends one learning message per stream. That means that the switch will send a number of learning messages at the same time. For example, for messages with $p_i = 1$ ms in *case A*, assuming length of learning messages of 300 bytes and network data rate of 100 Mbps, we calculate the learning bandwidth as follows:

$$L_{BW} = \sum_n \frac{1}{p_i} \frac{l_i}{dataRate} = 0.36 \tag{2}$$

Calculating L_{BW} like this for the other sets of messages we can evaluate equation (1) for all four cases. Last column of Table II shows those values. We see how for the cases in which the number of total messages is higher the bandwidth need for the learning mechanism is also higher, resulting even in non-feasible cases like *case C*. However that does not mean that the self-configuration approach does not work for those cases. It just means that it will take longer to send all the learning messages. With this small example we have just show how the number of streams in the network, its size and its period will impact the performance of the self-configuration approach proposed in this paper.

2) Handling of Changes: After the initial configuration has been distributed through the network, the switches ere now operating in one of their normal learning modes: *PERIODIC* or *NOTIFY*. If something changes in the network, i.e. changes in the traffic cause by the insertion or removal of devices, the CNC has now two ways of detecting. One option is that the end-station notifies of this change to the network. To do so it will communicate with the CUC using OPC-UA. The other option is through the learning capabilities of the switches that are in communication with the CNC using NETCONF protocol, periodically if the mode is *PERIODIC* and just when a change is detected if the mode is *NOTIFY*. These learning messages are sent as best effort traffic as discussed before. These two methods are not mutually exclusive, in fact their co-existence can be used by the CNC as redundant information for diagnostics purposes. In any case when a change is detected, steps 4) and 5) of the initial configuration process described in previous section will repeat.

V. Background

The configuration of real-time networks is a complex problem that has been addressed in the past, specially for the case scheduled traffic. One of the earliest works of dynamic configuration for time-triggered networks is the Flexible Time-Trigered (FTT) Ethernet protocol [9]. In FTT-Ethernet time is divided in Elementary Cycles (EC) at the beginning of which there is a time-slot reserved for the transmission of the Triggered Message. This control message synchronizes the network communicating the schedule for that EC to all devices in the network. A similar approach can be found in the Hard Real-Time Ethernet Switching (HaRTES) architecture [10] for which recently a distributed reconfiguration protocol has been introduced [11]. Whereas these works address the dynamic reconfiguration of the network, they do not propose any self-configuration mechanism. An automatic configuration approach is presented in [12] where the authors analyze its applicability to other real-time Ethernet variants such as Profinet, Ethernet/IP, Powerlink and Ethercat.

VI. Conclusions and Future Work

In this paper we have shown how by adding learning capabilities to the existing technologies for the configuration and management of real-time networks they can be used together to enable full network self-configuration. The set of standard developed by IEEE 802.1 TSN task group have been proved to be a good choice as the core of the self-configuration framework, and the communication protocols NETCONF and OPC-UA complements it covering aspects not defined in the standards. However self-configuration would not be possible without the specific learning mechanism that have been introduced in this paper, including the definition of managed objects for the switches and different learning modes.

As for the future work, there are many directions: from implementation aspects, like the computational requirements of the CNC and the switches, to more theoretical aspects like studying how many switches with learning capabilities are needed in a self-configured network or the specific learning algorithms both in the CNC and the switches.

Acknowledgment

The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007-2013/ under REA grant agreement 607727.

References

[1] W. Steiner, P. Gutiérrez Peón, M. Gutiérrez, A. Mehmed, G. Rodríguez Navas, E. Lisova, and F. Pozo, "Next Generation Real-Time Networks Based on IT Technologies," in *21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016.

[2] Institute of Electrical and Electronics Eingenieers, "802.1Qcc Stream Reservation Protocol (SRP) Enhancements and Performance Improvements," *http://www.ieee802.org/1/pages/802.1cc.html*, retrieved 16/04/2017, Draft 1.4.

[3] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat, "A Configuration Agent based on the Time-Triggered Paradigm for Real-Time Networks," in *11th IEEE World Conference on Factory Communication Systems (WFCS)*, 2015, Best Work-in-Progress Paper Award.

[4] ——, "Learning the parameters of periodic traffic based on network measurements," in *2015 IEEE International Workshop on Measurements & Networking (M&N)*, Oct 2015, pp. 1–6.

[5] M. Bjorklund, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)," RFC 6020, Oct. 2010. [Online]. Available: https://rfc-editor.org/rfc/rfc6020.txt

[6] R. Enns, M. Bjorklund, A. Bierman, and J. Schönwälder, "Network Configuration Protocol (NETCONF)," RFC 6241, Jun. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6241.txt

[7] J. Schonwalder, M. Bjorklund, and P. Shafer, "Network configuration management using netconf and yang," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 166–173, Sept 2010.

[8] E. Lisova, M. Gutiérrez, W. Steiner, E. Uhlemann, J. Åkerberg, R. Dobrin, and M. Björkman, "Protecting Clock Synchronization: Adversary Detection through Network Monitoring," *Journal of Electrical and Computer Engineering*, vol. 2016, 2016.

[9] P. Pedreiras, L. Almeida, and P. Gai, "The ftt-ethernet protocol: merging flexibility, timeliness and efficiency," in *Proceedings 14th Euromicro Conference on Real-Time Systems. Euromicro RTS 2002*, June 2002, pp. 134–142.

[10] R. Santos, M. Behnam, T. Nolte, P. Pedreiras, and L. Almeida, "Multi-level hierarchical scheduling in ethernet switches," in *2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*, Oct 2011, pp. 185–194.

[11] M. Ashjaei, Y. Du, L. Almeida, M. Behnam, and T. Nolte, "Dynamic reconfiguration in hartes switched ethernet networks," in *2016 IEEE World Conference on Factory Communication Systems (WFCS)*, May 2016, pp. 1–8.

[12] L. Dürkop, , J. Jasperneite, and A. Fay, "An Analysis of Real-Time Ethernets With Regard to Their Automatic Configuration," in *11th IEEE World Conference on Factory Communications Systems (WFCS)*, 2015.