Universitat
de les Illes Balears

First Analysis of the AVB's Stream Reservation Protocol in the Context of TSN

D. Bujosa, D. Cavka, I. Álvarez and J. Proenza

Departament
de Ciències Matemàtiques
i Informàtica

TECHNICAL REPORT
May/2019
A-02-2019

First Analysis of the AVB's Stream Reservation Protocol in the Context of TSN

Daniel Bujosa† Drago Čavka* Inés Álvarez† Julian Proenza†
†Department of Mathematics and Informatics, University of the Balearic Islands, Spain
†{daniel.bujosa, ines.alvarez, julian.proenza}@uib.es
*Faculty of Electrical Engineering, University of Banja Luka, Bosnia and Herzegovina
*drago.cavka@etf.unibl.org

ABSTRACT

The Audio Video Bridging (AVB) Task Group from the IEEE proposed a series of standards to provide Ethernet with soft real-time guarantees. Later on, the group was renamed to Time-Sensitive Networking and its scope was broadened to provide new services to more critical applications. The Stream Reservation Protocol (SRP) stands out among the projects developed by the groups, as it is key to bound the transmission delay and to minimize frame loss due to lack of resources. Nonetheless, SRP was originally designed for audio/video applications and does not take into account properties that are important for critical systems; such as termination. In this work we study the termination of the reservations using AVB's SRP. We used UPPAAL to model the protocol and to verify the property. We see that SRP does not provide termination, we discuss how the lack of termination can impact critical applications and we propose a series of solutions to provide termination using SRP.

1. INTRODUCTION

In 2005 the IEEE Audio Video Bridging Task Group started working in a new set of projects to provide Ethernet with soft real-time capabilities. The applications targeted by the group were related to audio and video streaming. Nevertheless, the interest in the work of the group moved to other application areas; such as automotive or automation. In 2012 the group was renamed and its scope broadened to address the needs of these new applications, becoming the Time-Sensitive Networking (TSN) Task Group [4]. The TSN Task Group aims at providing Ethernet with hard and soft real-time communications, flexibility of the traffic requirements and fault tolerance mechanisms for critical applications.

The number of projects carried out by the TSN Task Group is rapidly growing. Among these projects we find the Stream Reservation Protocol, which was originally standardized by the AVB Task Group in [2] and later extended by the TSN Task Group in [3]. Even though the revision of SRP included two new architectures, it still supports the distributed one proposed in AVB with no modifications.

SRP provides Ethernet with support for resource reservation along the path that connects a transmitter to its receivers. SRP is a key piece to support many of the projects developed in the scope of TSN. This is so because resource reservation prevents frame delays beyond predefined limits and losses due to buffer overflow. Furthermore, SRP allows to modify the traffic requirements in run-time, providing a certain degree of flexibility to the network.

Some of the applications targeted by the TSN Task Group are critical, as mentioned. The mechanisms developed for these applications must exhibit a series of properties to guarantee their correct operation. Moreover, these properties must be met in all the levels of the architecture, including the network. Nonetheless, the distributed version of SRP was not designed considering these applications and, thus, it was not designed to fulfill these properties.

In this work we study whether the distributed version of SRP can be deployed in critical systems. To do so, we model the protocol using the UPPAAL model checker [5] to check a series of properties that are typical in distributed systems for critical applications, *e.g.*, termination. We studied the termination of the reservation process; that is, we analyzed whether SRP guarantees that the reservation process is completed within a finite amount of time, regardless of the resources available in the network. We found that SRP does not provide termination. Moreover, we identify specific scenarios in which termination is not achieved, we discuss the consequences derived from the absence of termination and we provide a first overview of a series of mechanisms to enforce this property using SRP.

2. RELATED WORK

Due to the great relevance of the work carried out by the TSN Task Group, the community has carried out a significant amount of work related to their study, application and improvement, which has been surveyed in [6] and [8]. Many of these works are focused on the study of SRP's efficiency, such as the work presented in [10]. Moreover, some works present solutions to provide fault tolerance against permanent faults using SRP [7].

As we mentioned, the TSN Task Group has ammended the original SRP standard; in the IEEE Std 802.1Qcc [3]. This new standard describes two new architectures to carry out the management of the network in general, and the reservation of resources in particular. Specifically, these architectures rely in the centralization of the reservations to increase the efficiency in terms of resources and time required to perform the control of the network. In this same line, some works have proposed the use of Software Defined Networking techniques to manage the reservation of resources, such as those presented in [9] and [11].

Nevertheless, to the best of the authors' knowledge, there are no works related to the study of the fulfillment, by the distributed version of SRP, of properties that are fundamental for critical systems, such as termination or consistency. It is important to note that the distributed architecture is in-

cluded and supported by the new SRP standard. In this work we carry out a UPPAAL model of the resource reservation process using SRP, we study the termination of the reservations in different components and we propose mechanisms to provide termination using SRP.

3. SRP OVERVIEW

As anticipated in Section 1, the Stream Reservation Protocol [2] helps providing real-time guarantees to Ethernet-based communications; as it allows to ensure the availability of resources during the communication. This allows to bound the end-to-end delay of frames and to prevent packet losses. Furthermore, SRP allows to modify the traffic in run-time, providing certain degree of flexibility to the network.

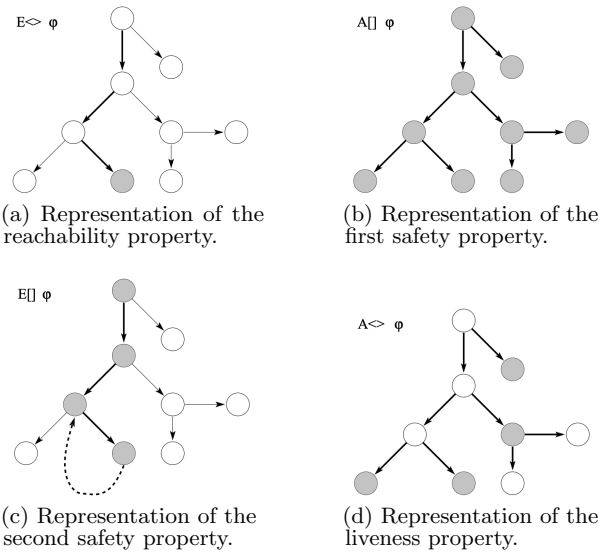
Currently, there are three different architectures proposed in the SRP standard [3]. The first architecture was originally proposed by AVB in [2], and consists in a distributed architecture, where resources are reserved in each network device using local information. The other two architectures rely on centralized components to manage the network. In this work we focus on the distributed architecture. Further details about the centralized architectures can be found in [3].

Either way, when using SRP the data communication is carried out through logical communication paths called *streams*. A stream is defined by a series of attributes, which represent the resources needed to transmit a specific set of messages. Moreover, SRP enforces a publisher-subscriber model, where the publisher is called *talker* and the subscribers are called *listeners*. In the distributed version of SRP, the talker is responsible for reserving the resources for the communication, i.e. the talker is responsible for creating the stream.

To create a stream the talker announces its intention to communicate by transmitting a special message called *talker advertise* (TA_A message). The TA_A message conveys information to identify the stream and the resources it needs. This message is processed by the bridges, which check whether there are enough resources in each one of their forwarding ports to create the stream. If there are enough resources in one port, the bridge forwards a TA_A message through said port; otherwise, the bridge forwards a *talker failed* message (TA_F message) through the port. A TA_F message conveys the same information as the TA_A message plus the reason for the failure in the reservation. It is important to note that, at this point, bridges register the talker's request, but do not carry out the reservation of resources.

When a listener receives a talker message, it processes the message and decides whether it wants to bind to the stream. If the listener does not want to bind, it does not perform any further actions nor informs the talker about it. On the other hand, if the listener receives a TA_A and wants to bind to the stream it checks whether it has enough resources to do so. If the listener has enough resources, it transmits a *listener ready* message (LA_R message). Otherwise, the listener forwards a *listener asking failed* message (LA_AF message). Finally, if a listener receives a TA_F and wants to bind to the stream, it will also forward an LA_AF message.

Bridges unify the responses from the listeners to transmit them towards the talker. To do so, bridges first process the responses received through each port and then generate a new response. When a bridge receives an LA_R through a port, it checks if the port has enough resources. If there are resources, the LA_R remains unaltered and the port locks the resources; otherwise the LA_R becomes an LA_AF. On the other hand, if a bridge receives an LA_AF it does nothing.



(a) Representation of the reachability property.

(b) Representation of the first safety property.

(c) Representation of the second safety property.

(d) Representation of the liveness property.

Figure 1: Properties that can be evaluated in UPPAAL based on a figure from [5]. Each figure shows the paths for which the state formulae holds; whereas the filled states are the ones where the state formulae is evaluated.

Once the bridge has processed the responses, it must unify them. If all ports have an LA_R, the bridge forwards an LA_R; while if all ports have an LA_AF, the bridge forwards an LA_AF. Finally, if the bridge has LA_R in some ports and LA_AF in some other ports, it forwards a so called *listener ready failed* message (LA_RF message).

When the talker receives an LA_R or an LA_RF message, it creates the stream and starts transmitting. Otherwise, if the talker receives an LA_AF, the stream is not created.

4. UPPAAL TOOL OVERVIEW

As we have already mentioned, we used the UPPAAL model checker to evaluate the termination of the reservations in SRP. UPPAAL is a tool to formally model real-time systems and to validate and verify their properties [5]. In UPPAAL a system is modelled as a network of interconnected *timed automatas* (finite-state machines extended with clocks that progress at the same pace) and data types. Moreover, UPPAAL provides a formal query language that allows to define the system's properties that are to be verified. The model checker explores all the possible execution paths of the model to check whether the properties hold. UPPAAL then informs the user about the result and, if a property does not hold, it shows an execution path in which the property is violated. We next describe the modelling tool and the query language in more detail.

4.1 The modelling tool

As we mentioned, in UPPAAL the system is modelled as a network of automatas. Each automata is specified as a *template* that can be extended with parameters to create instances of the same automata; each one called automaton. In turn, templates are constructed using *locations*, *edges*, local variables and local clocks. On top of that, different

templates can share global variables and global clocks, and can synchronize using *channels*. Specifically, there are two types of channel, *binary*, which synchronizes one sender and one receiver; and *broadcast*, which can synchronize a sender with an arbitrary number of receivers.

Each automaton progresses through a set of *locations*. The locations and variables of all automatas at a specific instant define the state of the system. There are three different types of location, *normal*, *committed* and *urgent*; depending on the residence time. Specifically, an automaton can indefinitely remain in a normal location, unless the residence time is upper bounded using a so called *invariant*. On the other hand, an automaton immediately leaves a committed or urgent location, i.e. the residence time is 0 and the time does not elapse in that location. Committed locations do not allow interleaving between different automatas; whereas urgent locations do allow interleaving. On top of that, the *initial* location represents the first location of the automaton.

As we said, the time an automaton remains in a normal location can be bounded using an invariant. An invariant is an expression that involves one or more clocks. Automatons progress through their locations using edges. Edges can be enabled or disabled using expressions called *guards*, which are defined using variables and clocks. Moreover, edges also allow to assign values to variables when they are taken.

4.2 The query language

UPPAAL provides a query language to express the properties that have to be verified for the system. Specifically, there are three different types of property, *reachability*, which is expressed with `E <>`; *safety*, which is expressed with `A []` or `E []`; and *liveness*, which is expressed with `A <>`. Figure 1 shows these properties, represented as a set of interconnected states. The figure shows the paths that the model checker follows, with bolder arrows, and the states that are evaluated, the highlighted circles.

Reachability (`E <>`) allows to evaluate whether a certain state in the model is eventually reached, e.g. the plane landed. Safety properties allow to evaluate two different scenarios. First, safety `A []` evaluates whether a certain condition is met in all the states, e.g. the temperature of the fuel is always lower than a predefined threshold. Second, safety `E []` evaluates whether there is always at least a path where the condition is always met, e.g. there is a path where the plane is still flying. Finally, liveness `A <>` allows to evaluate whether a state is always reached, independently of the path that is taken, e.g. a message transmitted through the network always reaches its destination.

Properties require what is called a *state formulae* to be expressed. The state formulae is code that allows to state conditions; e.g. i == 7 is true when variable i equals 7. Therefore, the state formulae allows to evaluate different properties in the model, such as i is always equal to 7, or it exists one path where i equals 7.

In the next section we describe the SRP model and the properties we evaluated, using the modelling tool and the query language described.

5. THE SRP MODEL

As we have mentioned in Section 1, we want to study whether SRP provides termination for the reservations. Thus, we modeled the SRP reservation process using three different templates. Each template models the relevant actions carried out by a talker, by a listener and by a bridge respectively. The talker template is instantiated once while the listener and bridge templates are instantiated three times. In this way the model is composed of one talker, three bridges and three listeners connected as shown in Figure 2.

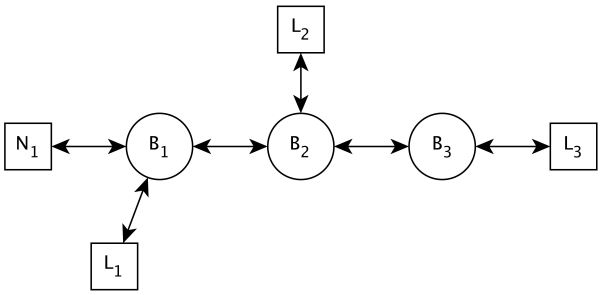


Figure 2: Network topology used for the model. It consists of one talker and three listeners. Each listener is attached to a different bridge, which are connected in a line topology.

Even though we could evaluate termination using a single talker and a single listener, such a setup would not allow to evaluate more complex scenarios, nor the complete behaviour of the protocol; e.g. how bridges merge different responses from the listeners. Thus, we decided to use three listeners. Moreover, each listener is attached to a different bridge, as it allows to study scenarios that are more pessimistic than having all nodes attached to the same bridge. Finally, we use a line topology as SRP relies on other protocols to eliminate loops in the network, such as the Rapid Spanning Tree Protocol [1], which establishes a logical line topology.

It is important to note that, as in any modelling process, we had to abstract part of the behaviour of the protocol. In this model we have abstracted the number of ports of the devices. Specifically, all devices count with a single reception port and a single transmission port. Nevertheless, each port can handle the reception and transmission of attributes to more than one device.

There are several reasons to abstract the number of ports in the model. First, modelling the logic of the bridge is not trivial and requires a significant amount of locations and variables. This complexity increases with the number of ports. Second, specifying a concrete number of ports reduces the generality of the model and the cases that can be explored. For instance, if we model bridges with two ports the number of attached devices would be limited to two. Finally, increasing the number of locations not only increases the complexity of the model, but also the number of states to be explored. This can lead to the explosion of the state space and the exhaustion of computational resources.

Thus, we had to adapt the model to handle the transmission of different attributes to different devices using a single port. Specifically, we included the possibility in reception to choose the type of attribute. More precisely, when a listener or bridge receives a TA_A, it can modify it to a TA_F, to emulate that the port in the transmitter did not count with enough resources. Nevertheless, if a device transmits a TA_F all other devices receive a TA_F.

We also abstracted the transmission of data frames. Concretely, the transmission of data frames is not modelled, as that would require modelling the different traffic shapers

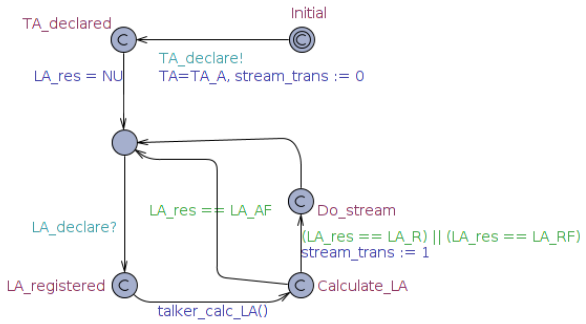


Figure 3: Talker template using the UPPAAL model checker.

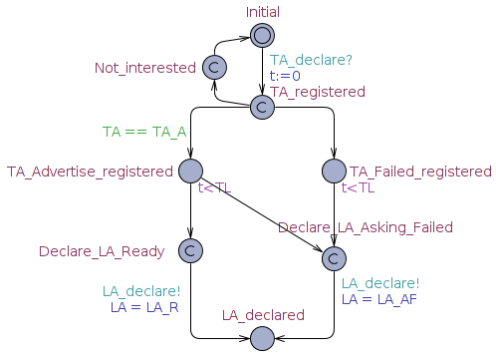


Figure 4: Listener template using the UPPAAL model checker.

proposed by the TSN Task Group. Instead, the transmission of data is represented with a location in the talker.

We next describe the basic operation of the templates used in the model.

5.1 Talker template

Figure 3 depicts the talker template. The operation of the talker starts in the `Initial` location, which is committed to prevent the model from being locked in the initial state forever. Thus, the model starts with the transition from the `Initial` location of the talker to the `TA_declared` location. This edge starts the transmission of the talker attribute message (TA message) with the channel `TA_declare`. After that, the automata waits to receive the listener attribute messages (LA messages) and calculates whether it can start transmitting or not. If the result of the calculation is a ready or a ready failed, the talker template reaches the location `Do_stream` which represents the start of the transmission. Otherwise, the talker waits to receive more listener attributes.

5.2 Listener template

Figure 4 depicts the listener template. The operation of the listener starts in the `Initial` location, where it waits for a TA message. As we described in Section 3, a listener may be interested in the stream or not. If it is not interested, the template moves to the `Not_interested` location and back to the `Initial` location. Otherwise, it processes the attribute.

If the attribute is a TA_F the template moves to the `TA_Failed_registered` location. On the other hand, if the attribute is a TA_A, the template can transition to the `TA_Advertise_registered` or to the `TA_Failed_registered` locations. Note that as we mentioned, we use a single port in the bridges and we emulate the transmission of different attributes to different receivers on reception. Thus, if the template moves to the `TA_Failed_registered` location, it emulates the lack of resources in the forwarding port of the bridge. Finally, if the listener has registered a TA_F it moves to the location `Declare_LA_Asking_Failed`. On the other hand, if the listener has registered a TA_A it moves to `Declare_LA_Ready` if it has enough resources for the stream and it moves to `Declare_LA_Asking_Failed` otherwise.

Thus, if the listener reaches the `Declare_LA_Ready` location, it represents that it received a TA_A and it wants to bind. Otherwise, if it reaches the `Declare_LA_Asking_Failed` location, it represents that the listener wants to bind, but the bridge or the listener did not have enough resources. Finally, the template transmits the proper LA message and moves to the `LA_declared` location using the `LA_declare` channel.

5.3 Bridge template

Figure 5 depicts the bridge template. The operation of the bridge starts in the `Initial` location, where it waits to receive a TA or an LA message.

If it receives a TA message it takes the left edge. The left edge registers the TA message, similarly to the listener. Specifically, if the bridge receives a TA_A, it can register a TA_A, emulating that the transmitter has enough resources; or it can register a TA_F, emulating that the transmitter does not have resources. After that, the bridge forwards the attribute and the automata returns to the `Initial` location.

On the other hand, if the bridge receives an LA message it takes the right edge. The right edge registers the LA messages and, according to the messages received and the resources of the bridge, it transmits the adequate LA message.

Specifically, when the bridge receives an LA message it registers the type of device which transmitted it in `prev_type`. It stores a `type_B` if the message comes from a bridge and a `type_L` if the message comes from the listener connected to the bridge. Then, it stores the type of LA message in `LA_prev` (a `LA_R`, a `LA_RF` or a `LA_AF`). After that, the bridge reaches the `Wait` location. In this location the bridge can receive more messages, restarting the loop by moving to the `LA_registered` location, or it can pass to the transmission phase, moving to the `Calculate_LA` location. In transmission, the bridge sends an LA message according to the LA messages received and the resources of the bridge; it moves to the `LA_declared` location and back to the `Initial` location.

6. EVALUATION OF THE TERMINATION

As we anticipated in Section 1, the distributed version of SRP was designed for audio/video applications and it was not designed taking into account the critical applications targeted by TSN. Critical applications have a series of requirements and the mechanisms used to support them must exhibit a series of properties, such as termination or consistency. In this work we focused on the termination of the reservations.

As we explained in Section 3, when listeners receive a TA_A message announcing a new stream, they send a message announcing their intention to bind to that stream. On the contrary, listeners do not inform the bridges nor the talker when they are not interested in the stream. We evaluate

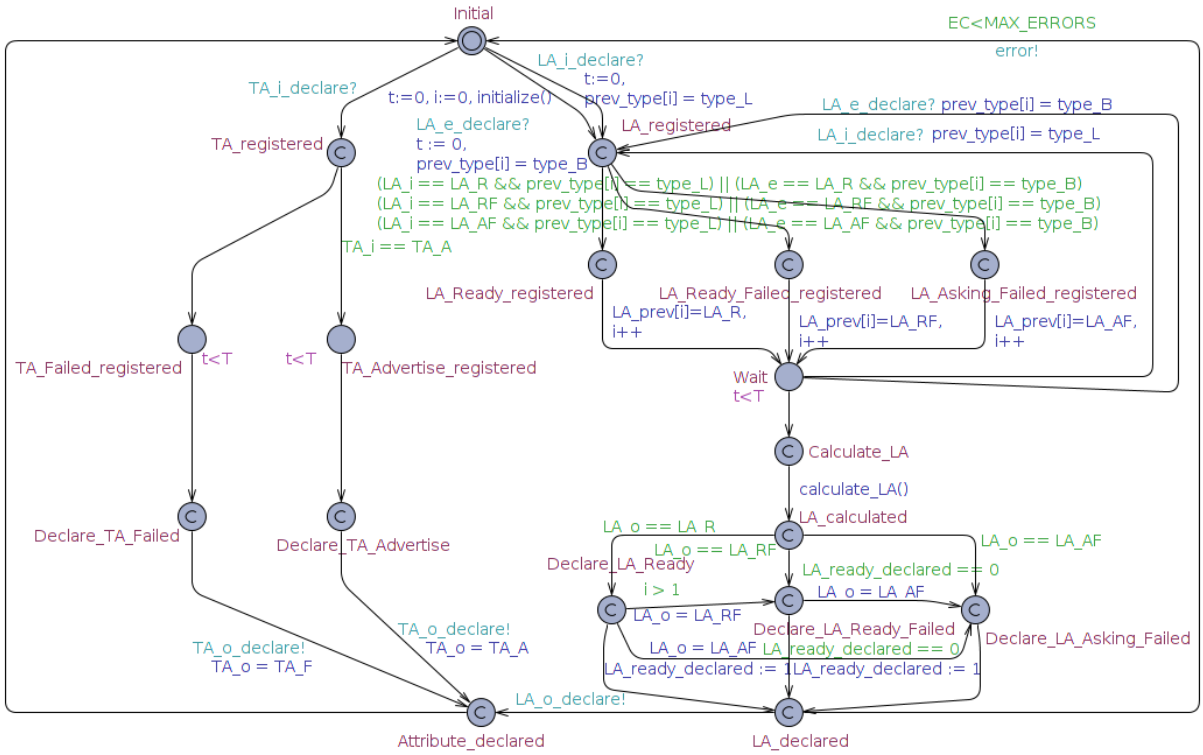


Figure 5: Bridge template using the UPPAAL model checker.

whether this behaviour causes scenarios where the reservation process is not concluded.

6.1 Termination in the Talker

We used the UPPAAL model to check whether SRP provides termination to the talkers. To do so, we used the query shown in Equation 1. This query evaluates whether the location `Talker.LA_registered` is always reached; i.e. it evaluates whether the talker always receives at least one response from a listener. This query is not satisfied in the model. Therefore, we can see that there are scenarios where the talker does not receive any response and, thus, SRP does not guarantee termination to the talker.

$$A <> \ Talker.LA\_registered \tag{1}$$

Many critical applications require to know the result of the reservations to make important decisions. Thus, the lack of termination can cause a malfunction of those applications. Not only it can block the decision process, but it can also lead to incorrect decisions due to the lack of knowledge. Moreover, delegating decisions related to the reservations to the application can cause inconsistencies between the view of the application and the state of the network.

A possible solution to this situation is to introduce timeouts in the talker. If the talker does not receive any listener attributes before the timeout expires, it should tear the stream down. The value assigned to these timeouts should be adjusted depending on the topology, to ensure that there is enough time for the listeners to communicate their intention to the talker. Concretely, the timeout must be adjusted to provide enough time to the furthest node in the network to announce its will to communicate.

This solution presents two main advantages. First, it solves a global problem locally, as it relies solely on information local to the talker. Second, this solution does not require the modification of SRP, as it can simply use the services already available in the protocol (announcing and deleting a stream). As a counterpart, it is not a general solution, as it depends on the network topology. Nevertheless, it would be possible to calculate the worst case time for the network, and use it for all reservations in a pessimistic approach.

6.2 Termination in Bridges

Similarly to the talker, a bridge that forwards a talker advertise message waits a response from a listener indefinitely. Also, bridges register talkers' attributes in all their ports, and they do so for all the talkers willing to transmit.

We used the queries shown in Equations 2 and 3 to evaluate the termination in bridges. These queries mean that locations `Bridge3.Attribute_declared` and `Bridge3.LA_registered` are always reached, respectively. As explained in Section 5, the first location is reached when the bridge receives and transmits a TA_A; while the second location is reached whenever the bridge receives an LA message. We see that the first query is satisfied but the second one is not. Therefore, we can see that there are scenarios where the bridge sends a talker advertise and does not receive any response.

$$A <> \ Bridge3.Attribute\_declared \tag{2}$$

$$A <> \ Bridge3.LA\_registered \quad (3)$$

This situation causes an unnecessary use of memory in bridges, as they store the information of streams that are never established. This is specially problematic in bridges, as the number of reservations they manage is considerably larger than in talkers. Thus, this can prevent the creation of new streams with listeners willing to bind.

It is important to note that this situation may not be solved by the solution proposed for the talkers. Let us assume we have the topology shown in Figure 2. Let us now assume that listeners L_1 and L_2 want to bind to the stream, but not listener L_3. The talker would receive the will to bind from L_1 and L_2, but bridge B_3 would wait indefinitely.

A possible solution is to introduce timeouts in the bridges. This mechanism would delete the stream registration in the memory of the bridge when none of the listeners connected to the bridge (directly or indirectly) want to bind to the stream. The value assigned to the timeout should be high enough to guarantee that all listeners can announce their will to bind to the stream before the registration is torn down. Furthermore, it should be high enough to guarantee that the talker can tear the stream down if no listeners want to bind.

The main advantage of this solutions if that it is local to each bridge. The counterpart is that it requires modifying SRP, as currently bridges can not create or eliminate declarations of streams. Moreover, the time a bridge must wait depends on the topology, even though it can be bounded using a worst case end-to-end delay calculation.

7. CONCLUSIONS

The AVB Task Group started a set of projects to provide standard Ethernet with soft real-time capabilities. The interest in the work of the group reached areas with tighter constrains in terms of timing guarantees and fault-tolerance. For this reason, the group was renamed to TSN and its scope broadened to provide hard and soft real-time guarantees, flexibility of the traffic and fault tolerance mechanisms.

Among the projects carried out by the groups we find the standardization of the Stream Reservation Protocol. SRP is key to provide timing guarantees, as it supports the reservation of resources. SRP proposes three different architectures to carry out the resource reservation. In this work we focused on the distributed architecture of SRP proposed in AVB.

We have studied whether this architecture is adequate for the critical applications targeted by TSN. Specifically, we focus on the termination in the reservations. We modeled the reservation process using the Uppaal model checker. Specifically, we modeled the behaviour of the talker, the listener and the bridge, each as a different template. We used a model composed of a talker, three bridges and three listeners to evaluate the termination of the reservations.

First, we evaluated whether SRP provides termination of the reservations in the talkers. We concluded that SRP does not provide termination. Specifically, if no listeners want to bind to the stream, the talker waits indefinitely for a reply, which can cause the malfunction of the application. We proposed to use timers to tear down the stream when no listeners want to bind. Second, we evaluated termination in bridges. Similarly to what happens in talkers, bridges wait to receive a reply from a listener indefinitely, which can have an impact in the following reservations. Again, we propose to use timers to tackle this issue and provide termination.

Acknowledgements

This work is supported in part by the Spanish Agencia Estatal de Investigación (AEI) and in part by FEDER funding through grant TEC2015-70313-R (AEI/FEDER, UE). Drago Čavka was supported by a scholarship of the EUROWEB+ Project, which is funded by the Erasmus Mundus Action II programme of the European Commission.

8. REFERENCES

[1] IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges. *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, pages 1–281, June 2004.

[2] IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP). *IEEE Std 802.1Qat-2010 (Revision of IEEE Std 802.1Q-2005)*, Sept 2010.

[3] IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements. *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)*, pages 1–208, Oct 2018.

[4] IEEE Audio and Video Bridging Task Group. https://1.ieee802.org/tsn/, April 2019.

[5] G. Behrmann, A. David, and K. G. Larsen. *A Tutorial on Uppaal*, pages 200–236. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[6] M. D. Johas Teener, A. N. Fredette, C. Boiger, P. Klein, C. Gunther, D. Olsen, and K. Stanton. Heterogeneous networks for audio and video: Using ieee 802.1 audio video bridging. *Proceedings of the IEEE*, 101(11):2339–2354, Nov 2013.

[7] O. Kleineberg, P. Fröhlich, and D. Heffernan. Fault-tolerant Ethernet networks with Audio and Video Bridging. In *ETFA2011*, pages 1–8, Sept 2011.

[8] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury. Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research. *IEEE Communications Surveys Tutorials*, 21(1):88–145, Firstquarter 2019.

[9] N. G. Nayak, F. Dürr, and K. Rothermel. Software-defined environment for reconfigurable manufacturing systems. In *2015 5th International Conference on the Internet of Things (IOT)*, pages 122–129, Oct 2015.

[10] D. Park, J. Lee, C. Park, and S. Park. New automatic de-registration method utilizing a timer in the IEEE802.1 TSN. In *2016 First IEEE International Conference on Computer Communication and the Internet (ICCCI)*, pages 47–51, Oct 2016.

[11] D. Thiele and R. Ernst. Formal analysis based evaluation of software defined networking for time-sensitive Ethernet. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 31–36, March 2016.