Universitat
de les Illes Balears

Estudio de la idoneidad del *Stream Reservation Protocol* para dar soporte a Sistemas Empotrados Distribuidos Críticos Adaptativos

Daniel Bujosa Mateu

Memoria del Trabajo de Fin de Máster

Máster Universitario en Ingeniería Industrial
de la
UNIVERSITAT DE LES ILLES BALEARS

Curso Académico 2018/2019

September 20, 2019

Nombre del Tutor 1: Inés Álvarez Vadillo

Nombre del Tutor 2: Julián Proenza Arenas

Estudio de la idoneidad del *Stream Reservation Protocol* para dar soporte a Sistemas Empotrados Distribuidos Críticos Adaptativos

Daniel Bujosa Mateu
Supervisores: Inés Álvarez y Julián Proenza
Trabajo final del Máster en Ingeniería Industrial (MEIN)
Universitat de les Illes Balears
07122 Palma de Mallorca
daniel.bujosa@uib.es

Resumen

En 2005 el IEEE creó un *Task Group* con el objetivo de dotar a Ethernet de nuevos y mejores servicios de comunicaciones. Este grupo fue llamado *Audio Video Bridging* (AVB), y fue el responsable de proporcionar a Ethernet garantías de tiempo real suave a la vez que mantenía las fortalezas de Ethernet; bajo coste, gran ancho de banda, compatibilidad con Internet y su funcionalidad *plug&play*.

Para conseguir estos objetivos, el AVB *Task Group* comenzó tres proyectos. El primero fue el IEEE Std 802.1AS, dedicado a la sincronización de reloj; el segundo fue el IEEE Std 802.1Qav, que estandarizó el *Credit-Based Shaper*; y, finalmente, el IEEE Std 802.1Qat que estandarizó el *Stream Reservation Protocol* (SRP). Además, el grupo juntó los proyectos anteriormente mencionados en el IEEE Std 802.1BA-2011: *Audio Video Bridging Systems*.

En 2012 el AVB *Task Group* amplió su ámbito de trabajo a los sistemas críticos, como algunos usados en automatización, automoción y aviónica. Entonces pasó a llamarse *Time-Sensitive Networking* (TSN) y su lista de objetivos aumentó para incluir (i) soporte a tráfico de tiempo real duro, (ii) soporte a la reconfiguración del tráfico y (iii) soporte a aplicaciones altamente fiables por medio de tolerancia a fallos.

De todos los proyectos anteriormente mencionados podemos destacar el SRP. Este es un protocolo clave para proporcionar garantías de tiempo real a la vez que mantiene el comportamiento *plug&play* de Ethernet. Esto es así debido a que, por un lado, es capaz de comprobar que un tráfico que se quiere transmitir tendría suficientes recursos para ser transmitido y, por otro lado, permite modificar el tráfico durante el tiempo de ejecución.

En este trabajo analizaremos el SRP desde diferentes perspectivas y mediante diferentes técnicas. En primer lugar, desarrollaremos y utilizaremos una configuración experimental para estudiar el desempeño del protocolo; lo que nos dará una medida de la flexibilidad operacional que proporciona a los sistemas que lo usan, es decir, de su capacidad para dar soporte a cambios en el tráfico y en sus requisitos de transmisión en tiempo real sin para ello tener que interrumpir los servicios de comunicación. De esta manera, determinaremos la idoneidad del protocolo para dar soporte a sistemas adaptativos; que son aquellos capaces de ajustar de manera automática sus estrategias internas, para responder adecuadamente a cambios en un entorno dinámico.

En segundo lugar, modelaremos el SRP usando el UPPAAL *model checker* para comprobar si SRP proporciona propiedades que son importantes para los sistemas distribuidos críticos; como la terminación y la consistencia.

Abstract

In 2005 the IEEE created a Task Group to provide Ethernet with new and enhanced services. This Task Group was named Audio Video Bridging (AVB), and it was responsible to provide Ethernet with soft real-time guarantees, while keeping Ethernet's strengths; namely low cost, high bandwidth, Internet compatibility and plug&play functionality.

To achieve its goals, the AVB Task Group started three projects. The first project was the IEEE Std 802.1AS, devoted to clock synchronization; the second one was the IEEE Std 802.1Qav, which standardized the Credit-Based Shaper and finally, the IEEE Std 802.1Qat standardized the Stream Reservation Protocol (SRP). Moreover, the group created a profile that puts the aforementioned projects together, the IEEE Std 802.1BA-2011: Audio Video Bridging (AVB) Systems.

In 2012 the AVB Task Group broadened its scope to critical systems, such as those used in automation, automotive and avionics. The Task Group was renamed to Time-Sensitive Networking (TSN) and the number of objectives was increased to include (i) support for hard real-time traffic, (ii) support for the reconfiguration of the traffic and (iii) support for highly reliable applications by means of fault-tolerance.

From all the aforementioned projects, we can highlight the SRP. It is key to provide real-time guarantees while supporting the plug&play behaviour of Ethernet. This is so as it allows to ensure that the traffic has enough resources to be transmitted and it also allows to modify the traffic in run-time.

In this dissertation we will study SRP from different perspectives and using different techniques. First, we will develop and use an experimental setup to study the performance of the protocol; which will give us a measure of the operational flexibility that it provides to the systems that use it, that is, its ability to support changes in traffic and its transmission requirements in real time at run time. In this way, we will assess the suitability of the protocol to support adaptive systems; which are able to automatically adjust their internal strategies, to respond properly to changes in a dynamic environment.

Second, we will model SRP using the UPPAAL model checker to check whether SRP provides properties that are important for critical distributed systems; such as termination and consistency.

I. INTRODUCCIÓN

En 2005 el *Audio Video Bridging* (AVB) *Task Group* (TG) del *Institute of Electrical and Electronics Engineers* (IEEE) [1], centrado en las aplicaciones relacionadas con la transmisión de audio y vídeo, comenzó a trabajar en un nuevo conjunto de proyectos para crear un estándar que proporcionara a Ethernet propiedades de Tiempo Real (TR). Para conseguir este objetivo, el AVB TG comenzó tres proyectos. El primero fue el IEEE Std 802.1AS [2], dedicado a la sincronización de reloj; el segundo fue el IEEE Std 802.1Qav, que estandarizó el *Credit-Based Shaper* [3]; y, finalmente, el IEEE Std 802.1Qat que estandarizó el *Stream Reservation Protocol* (SRP) [4]. Además, el grupo juntó los proyectos anteriormente mencionados en el IEEE Std 802.1BA-2011: *Audio Video Bridging Systems* [5].

Sin embargo, el grupo comenzó a interesarse más adelante por otras áreas de aplicación como la automoción o la automatización. Por esta razón, en 2012 el grupo fue renombrado y paso a llamarse *Time-Sensitive Networking* (TSN) TG, a la vez que sus objetivos aumentaron para cubrir las necesidades que son habituales en estas nuevas aplicaciones. Concretamente, el TSN TG tiene como objetivo convertir Ethernet en un estándar con comunicaciones en TR duras y suaves, flexibilidad de los requisitos de tráfico y mecanismos de tolerancia a fallos.

A pesar de que el número de proyectos llevados a cabo por el TSN TG crece a gran velocidad, hay una serie de proyectos que pueden ser considerados el núcleo de la actividad del TG. Uno de estos proyectos es el SRP, que fue originalmente estandarizado por el AVB TG en [4] y posteriormente revisado por el TSN TG en [6].

SRP permite a Ethernet reservar recursos a lo largo del camino que conecta a un transmisor con sus receptores. Por esta razón, SRP es una pieza clave para muchos de los proyectos desarrollados en el ámbito de TSN. Esto se debe a que este protocolo autoriza la transmisión de nuevos mensajes solo después de comprobar que haya suficientes recursos en el camino que conecta a un transmisor y sus receptores durante las comunicaciones. Esto evita que en la transmisión de las tramas se produzcan retrasos mayores que los límites predefinidos y también que se den pérdidas de tramas debidas a desbordamientos de algún *buffer*. Además, SRP permite modificar los requisitos del tráfico durante el tiempo de funcionamiento, confiriendo un cierto grado de flexibilidad a la red.

Sin embargo, algunas de las aplicaciones a las que va dirigido TSN presentan estrictas restricciones con respecto al tiempo de respuesta y precisan de soporte para mecanismos de tolerancia a fallos. En estos casos, todos los protocolos de comunicación, como puede ser el SRP, deberían presentar ciertas propiedades que suelen ser básicas en sistemas distribuidos críticos adaptativos para garantizar el correcto funcionamiento del conjunto del sistema. Propiedades como pueden ser un tiempo de respuesta corto para los cambios de configuración del sistema y la terminación y la consistencia de esos mismos cambios.

En este trabajo analizaremos diferentes aspectos del SRP utilizando diferentes herramientas. Por un lado, estudiaremos el tiempo de respuesta a los cambios de configuración del sistema desde el punto de vista experimental y, por otro lado, analizaremos la terminación y consistencia mediante un entorno de modelado, validación y verificación llamado UPPAAL [7], el cual está especializado en modelar sistemas de TR. Así constataremos que SRP no tiene ni la propiedad de terminación ni la de consistencia. Además, mediante los resultados obtenidos con UPPAAL discutiremos las consecuencias derivadas de la ausencia de terminación y consistencia, y proporcionaremos una primera descripción general de una serie de mecanismos que proponemos en este trabajo para mejorar SRP de tal manera que presente las propiedades arriba indicadas.

El resto del documento está estructurado como sigue. La Sección II proporciona una visión general de los distintos trabajos relacionados. La Sección III describe los aspectos de SRP que han tenido más relevancia en nuestra investigación. A partir de este punto comienzan las secciones correspondientes al análisis experimental del tiempo de respuesta a los cambios de configuración del sistema. La Sección IV describe con cierto detalle el proceso de implementación de la red. La Sección V muestra la configuración de los experimentos y proporciona una breve justificación del porqué de cada uno de ellos. Por último, la Sección VI expone los resultados obtenidos mediante el proceso experimental y las conclusiones derivadas del análisis de dichos resultados. A partir de este punto comienzan las secciones correspondientes al análisis de las propiedades terminación y consistencia mediante la herramienta UPPAAL. La Sección VII proporciona una explicación general del funcionamiento del UPPAAL *model checker*. La Sección VIII describe el modelo de SRP realizado; mientras que la Sección IX discute la terminación y la Sección X discute la consistencia. Finalmente, la Sección XI cierra el documento con un resumen del trabajo realizado y una pequeña lista de trabajos pendientes de realizar en un futuro.

II. ESTADO DEL ARTE

Debido a la relevancia de los estándares de AVB, la comunidad ha realizado una cantidad significativa de trabajos

relacionados con su estudio, aplicación y mejora [8] [9]. Algunos de estos trabajos se centran en el estudio de la eficiencia del SRP, como el mostrado en [10], que busca mejorarla mediante la eliminación de tipos de mensajes. Mientras que otros presentan soluciones para proporcionar tolerancia a fallos permanentes mediante el uso del SRP [11].

Como se ha mencionado con anterioridad, el AVB TG pasó a llamarse TSN TG, aumentando también, en gran medida, el número de áreas de aplicación a las que apunta. Este renovado grupo desarrolló un nuevo estándar SRP; el IEEE Std 802.1Qcc [6]. Este nuevo estándar describe dos nuevas arquitecturas para la gestión de la red en general y la reserva de recursos en particular. Concretamente, estas arquitecturas se basan en la centralización de las reservas para mejorar la eficiencia en términos de recursos y tiempo requerido para completar las reserva.

Sin embargo, a pesar de la gran cantidad de trabajos realizados, este es el primer estudio que analiza el tiempo de respuesta para los cambios de configuración del sistema, la consistencia y la terminación de la versión de AVB (distribuida) del SRP. En este trabajo hemos implementado una red AVB y creado un modelo UPPAAL de la reserva de recursos usando el SRP, mediante los cuales hemos estudiado dichas propiedades. En las siguientes secciones (Secciones V, IX y X) se puede ver una explicación más detallada de la importancia de estas tres propiedades.

III. DESCRIPCIÓN GENERAL DEL SRP

Tal y como se anticipó en la Sección I, El SRP [4] es clave para proveer garantías de TR a las comunicaciones basadas en Ethernet; dado que permite comprobar que hay suficientes recursos para garantizar la respuesta en tiempo real de los mensajes que se quieren transmitir. Esto permite acotar el retraso *end-to-end* de las tramas y evitar la pérdida de paquetes debido al *buffer overflow*. Además, el SRP se puede utilizar para modificar el tráfico durante el tiempo de funcionamiento del sistema, confiriendo a la red cierto grado de flexibilidad. Como se ha dicho también con anterioridad (Sección II), actualmente existen 3 arquitecturas diferentes propuestas en el estándar de SRP [6]. Sin embargo, en este trabajo nos centraremos en la versión distribuida.

El SRP sigue el paradigma *publisher-subscriber*, donde el nodo *publisher* es llamado *talker* y los nodos *subscribers*, *listeners*; mientras que las comunicaciones de datos se realizan a través de *streams*. Un *stream* es un canal lógico de comunicaciones que transporta tráfico con características especiales, como por ejemplo, un cierto periodo o tamaño de trama. En la versión distribuida del SRP, el *talker* es el responsable de iniciar la creación del *stream*. A continuación se explicará en detalle el proceso de reserva de recursos del SRP, el cual también se encuentra descrito en la Fig. 1.

Para crear un *stream* el *talker*, representado en la Fig. 1 como un cuadrado con una 'T' inscrita, debe declarar su intención de comunicarse mediante la transmisión de un mensaje especial llamado *Talker Advertise* (TA_A), el cual es transmitido en modo *broadcast*. Es importante notar que

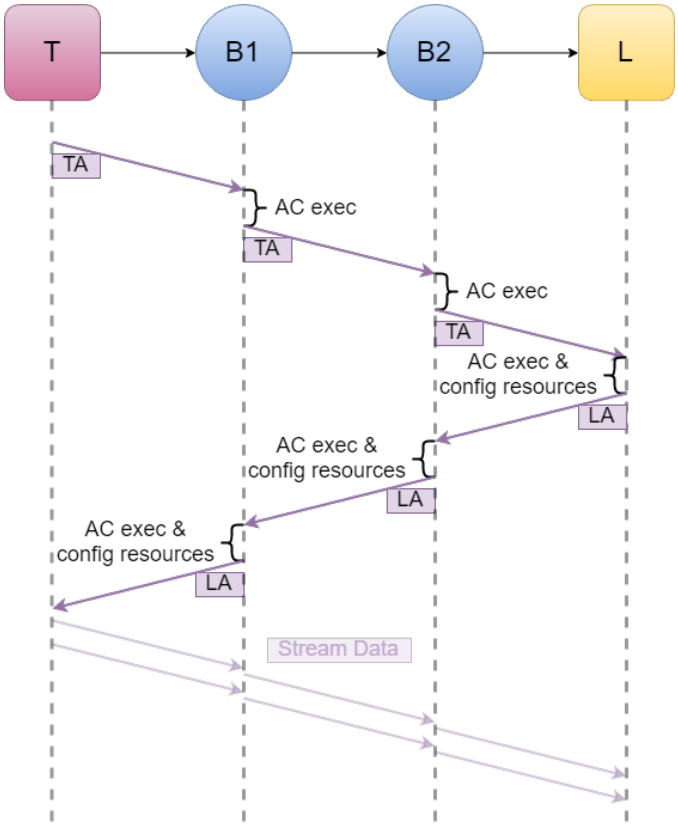


Fig. 1: Ejemplo de topología de una red AVB y esquema del mecanismo de reserva de recursos del SRP.

existen mecanismos que eliminan los bucles para evitar que las declaraciones circulen infinitamente en la red. El mensaje TA_A contiene información para identificar el *stream* y los recursos que este necesita. Los *bridges*, representados en la Fig. 1 como un círculos con una 'B' y un número identificativo inscrito, procesan el mensaje ("*AC exec*" en la Fig. 1) y comprueban si en los puertos de reenvío hay suficientes recursos para crear el *stream*. Si hay suficientes recursos en un puerto, el *bridge* reenvía el mensaje TA_A a través de él; de lo contrario, si el puerto no tiene recursos suficientes, el *bridge* transmite un mensaje *Talker Advertise Failed* (TA_F) a través de él. Un mensaje TA_F se transmite también en modo *broadcast* y contiene la misma información que un mensaje TA_A pero añadiendo el motivo por el que la reserva de recursos ha fallado. Es importante notar que, en este punto, los *bridges* registran la petición del *talker* pero aún no realizan la reserva de recursos.

Cuando un *listener*, representado en la Fig. 1 como un cuadrado con una 'L' inscrita, recibe una declaración del *talker* este decide si quiere unirse al *stream* o no. Si el *listener* no está interesado en el *stream* este no realizará ninguna acción al respecto ni informará a nadie acerca de su decisión. Por el contrario, si el *listener* sí está interesado en el *stream* se pueden producir 3 escenarios diferentes: (i) si el *listener* recibe un mensaje TA_A, el *listener* comprueba sus recursos

y, si tiene suficientes como para recibir el *stream*, transmite un mensaje *Listener Ready* (LA_R); (ii) si al comprobar sus recursos estos no son suficientes, el *listener* trasmitirá un mensaje *Listener Asking Failed* (LA_AF) y (iii) si el *listener* recibe un mensaje TA_F él también transmitirá un mensaje LA_AF.

Los *bridges* combinan las respuestas de los *listeners* para mandárselas a los *talkers*. Para poder llevar esto a cabo los *bridges* analizan las respuestas recibidas por cada puerto para luego generar la nueva respuesta que transmitirán al *talker* o a los *bridges* en dirección al *talker*. Cuando un *bridge* recibe un mensaje LA_R a través de un puerto, él comprueba que dicho puerto tenga recursos suficientes. Si hay suficientes recursos, el LA_R se mantiene inalterado y el puerto reserva los recursos necesarios ("*config resources*" en la Fig. 1); en caso contrario, el LA_R pasa a ser un LA_AF. Por otro lado, si el *bridge* recibe un mensaje LA_AF el valor se deja inalterado y el puerto no reserva los recursos.

Tras procesar las respuestas recibidas, cada *bridge* debe unirlas para su retransmisión. En este sentido, si todos los puertos de un *bridge* presentan un LA_R, ese *bridge* transmitirá un LA_R en dirección al *talker*; mientras que si todos los puertos presentan un LA_AF, el *bridge* transmitirá un LA_AF. Finalmente, si el *bridge* tiene LA_Rs en algunos puertos y LA_AFs en otros, este transmitirá un nuevo tipo de mensaje llamado *Listener Ready Failed* (LA_RF). Es importante notar que los *bridges* que reciban este tipo de mensaje lo procesarán como si se tratara de un mensaje LA_R.

Mientras el *talker* espera una respuesta, o recibe mensajes LA_AF, el *stream* no es creado. Sin embargo, una vez el *talker* reciba un LA_R o un mensaje LA_RF, el *talker* creará el *stream* y comenzará a transmitir. Una vez el *stream* ha sido creado el *talker* puede eliminarlo en cualquier momento. Esto puede hacerse gracias a un mecanismo denominado *unadvertise stream mechanism*, el cual consiste en la transmisión de un mensaje que elimina las reservas de recursos realizadas para ese *stream* de todos los elementos de la red. Este mensaje se transmite en modo *broadcast*, igual que el mensaje TA_A, para asegurar que todos los *bridges* y *listeners* lo reciben.

Por otro lado, tal y como hemos mencionado al principio de esta sección, los *streams* son canales lógicos de comunicaciones que transportan tráfico con características especiales. Una de estas características es la clase. Los *streams* pueden ser de clase A o B. Este parámetro solo influye en la *Quality of Service* (QoS) del *stream*, determinando aspectos como la prioridad. Sin embargo, no tiene un impacto notable en el SRP más allá de que un dispositivo puede o no tener recursos suficientes para un *stream* dependiendo de este parámetro, ya que cuanto menos exigente en recursos sea el *stream* más posibilidades hay de que los dispositivos tengan suficientes recursos.

Una vez explicados los diferentes aspectos generales del trabajo pasaremos a las secciones referentes al análisis experimental del protocolo.

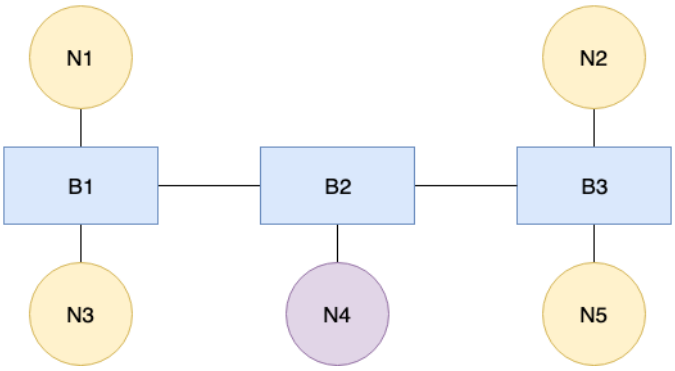


Fig. 2: Topología de la configuración experimental.

IV. Implementación Experimental

Una vez expuestas las generalidades del protocolo y el contexto en el que se desarrolló, procederemos a describir el análisis experimental del tiempo de respuesta a los cambios de configuración del sistema realizado en este trabajo. Tal y como se ha dicho anteriormente, el proceso y análisis experimental se encuentran descritos en 3 secciones: la presente sección (la Sección IV), que describe con cierto detalle el proceso de implementación de la red; la Sección V, que muestra la configuración de los experimentos y proporciona una justificación del porqué de cada uno de ellos; y la Sección VI, que expone los resultados obtenidos mediante la experimentación y las conclusiones derivadas del análisis de dichos resultados.

Por otro lado, en esta sección distinguiremos entre implementación hardware y software. En la subsección de implementación hardware (Subsección IV-A) explicaremos la topología utilizada en la experimentación y los distintos componentes utilizados; mientras que, en la subsección de implementación software (Subsección IV-B), explicaremos el desarrollo de los diferentes programas utilizados.

A. Implementación Hardware

Nuestra configuración experimental está inspirada en el trabajo de *Meyer et al.* [12], que incluye una arquitectura de red y una configuración del tráfico concreta, adaptándolo para cumplir con nuestras restricciones de hardware. Restricciones, como por ejemplo, el reducido número de puertos Ethernet en los *bridges* AVB. Específicamente, redujimos el número de nodos de 10 a 5 y, por lo tanto, tuvimos que reconfigurar los nodos de origen y destino para los *streams*. Además, solo se consideró el tráfico activado por eventos debido a la falta de soporte explícito para los *streams* activados por tiempo en AVB.

La configuración experimental, representada en la Fig. 2, la conforman 5 nodos (N1 a N5) y 3 *bridges* (B1 a B3), conectados en una topología lineal. Los nodos N1, N2, N3 y N5 se han implementado en PCs comunes y se conectan a la red mediante tarjetas de red Intel i210-T1 [13], que admiten todos los estándares AVB. El nodo N4, por el contrario, ha sido implementado en un PC empotrado, es decir, un PC con una capacidad de cómputo y memoria limitados dado que solo se

Fig. 3: Red real utilizada para realizar el análisis experimental.

necesitaba para transmitir mensajes *best-effort* (mensajes para los que la red no puede garantizar que los datos lleguen a su destino). Los *bridges* AVB, por otro lado, se basan en el SoC ARM9 Marvell 88E7251 de 400 MHz. Y, finalmente, los enlaces operan a 100 Mbps. La red real resultante es la que se muestra en la Fig. 3. En ambas figuras el amarillo señala los nodos implementados en los PCs (nodos N1, N2, N3 y N5); el lila, el nodo implementado en el PC empotrado (nodo N4); y el azul, los *bridges* AVB.

Además, hemos utilizado un Arduino para poder reiniciar los *bridges* mediante comandos de la consola de Linux que se ejecuta en uno de los ordenadores y nos permite controlar la experimentación. Finalmente hemos usado un analizador de red para monitorizar el tráfico en la red. En cuanto al analizador de red, hemos utilizado el Hilscher netANALYZER NANL-B500G-RE [14], que captura tramas en múltiples enlaces y las almacena con una resolución de nanosegundos.

B. Implementación Software

Por un lado, el software de esta implementación experimental está basado en un código de OpenAvnu [15] llamado *"simple talker and listener"*. Este código está validado por Avnu Alliance [16] y, como su nombre indica, fue creado para que un único *talker* transmitiera un único *stream* a un único *listener*. En consecuencia, el código tuvo que ser considerablemente modificado para ajustarse a los requisitos de flexibilidad y configuración de los experimentos.

En el código OpenAvnu, los nodos ejecutaban, entre otros códigos, uno llamado *simple_talker*, en el caso de los *talkers*, o uno llamado *simple_listener*, en el caso de los *listeners*. Estos se encargaban de crear el *stream* en el caso del *talker* y los mecanismos de recepción en el caso del *listener*. El primer cambio que se debía aplicar consistía en conferir a cada nodo la capacidad de transmitir o recibir más de un *stream*. Esto se consiguió mediante la adición de una capa superior en la que se crean múltiples threads, donde cada uno de estos threads ejecuta un *simple_talker* o *simple_listener*. Esta nueva capa de código requirió realizar muchos cambios en el código original. Básicamente, ciertas inicializaciones tuvieron que ser movidas a la nueva capa, se añadieron semáforos, algunas variables globales tuvieron que pasar a ser locales, etc. Es importante notar que este fue un proceso largo y difícil debido a la longitud y complejidad del código original.

Por otro lado, tal y como se ha mencionado con anterioridad, AVB presenta dos clases de tráfico, la clase A y la clase B. Dado que la versión original del código solo permitía transmitir un tipo de *stream* con periodo, *payload* y clase prefijados (clase A), fue necesario implementar algunos cambios que confirieran configurabilidad en cuanto a las características de los *streams* transmitidos. El cuanto al periodo y *payload* bastó con modificar algunas variables teniendo en cuenta las restricciones y condiciones del protocolo AVB. Sin embargo, capacitar a los nodos para transmitir mensajes de clase B supuso realizar cambios en funciones situadas en diversas capas inferiores del código.

Por último, en la capa más alta creada se implemento una función de configuración de las comunicaciones. Esta lee un documento que presenta la estructura mostrada en la Fig. 4 y contiene la información necesaria para configurar los diferentes *streams* de los diferentes nodos que conforman la red. Concretamente, el fichero de configuración indica a cada nodo qué *streams* debe escuchar como *listener* (los dos números tras el "*listen to:*" conforman el identificador del *stream* a escuchar); tras lo cual, especifica las características de cada uno de los *streams* (periodo, *payload*, *delay*, clase y *reset*). Esta función de configuración resultó ser muy útil ya que evitaba tener que compilar el código cada vez que se tenía que hacer un cambio en la configuración de las comunicaciones, lo cual agilizó mucho el proceso de experimentación. Especialmente si tenemos en cuenta que cada modificación en las comunicaciones suponía compilar el código en, como mínimo, dos ordenadores diferentes y, como máximo, en cuatro.

Por otro lado, durante la fase de experimentación nos percatamos de algunas restricciones de AVB que impedían el buen funcionamiento de los nodos. Sin embargo, gracias a los estándares y al análisis del código pudimos resolver todos los problemas, de los cuales daremos algún ejemplo a

```
nodeID 1:
    listen to: 0 0
    listen to: 0 1
    :
    listen to: 2 4
nodeID 2:
    listen to: 0 2
    listen to: 2 0
    :
    listen to: 3 1
:
nodeID 9:
    listen to: 4 0
    listen to: 4 1
    :
    listen to: 4 9
stream 0 0:
    period =  250000
    payload = 32
    delay =   0
    class =   b
    reset =   0
stream 0 1:
    period =  125000
    payload = 64
    delay =   1000000
    class =   a
    reset =   0
:
stream 4 9:
    period =  250000
    payload = 32
    delay =   1000000
    class =   b
    reset =   1
```

Fig. 4: Ejemplo del fichero de configuración.

continuación.

Un ejemplo de error detectado durante la experimentación consistía en que algunos nodos solo podían transmitir uno de los diversos *streams* que debían transmitir. Esto se debía a ciertas restricciones relacionadas con las direcciones MAC y los identificadores de los *streams*. Este error fue fácil de solucionar mediante la implementación de nuevas funciones que determinaran las MACs de los nodos e identificadores de los *streams* en el caso de los *talkers* y nuevos filtros para detectarlos en los *listeners*.

Otro problema se produjo a causa de una de las primeras configuraciones propuestas para los experimentos. Con dicha configuración nuestra red no era capaz de reservar los recursos necesarios para todos los *streams*, debido a la falta de recursos, lo cual resultaba bastante extraño teniendo en cuenta que un análisis de dicha configuración mostraba que la utilización de los recursos debería ser inferior al 75% con dicha configuración. Analizando los estándares de AVB nos dimos cuenta de que, en cuanto al periodo de los *streams*, el mensaje TA_A solo informa de cuantos mensajes se van a transmitir en un periodo concreto. Por ejemplo, el periodo de los *streams* clase A es 125 μs, por lo que, si los datos de un *stream* clase A deben ser transmitidos con un periodo de 62.5 μs, el mensaje TA_A informará de que el *stream* desea transmitir *dos* mensajes por periodo. Si por el contrario, el *stream* quiere transmitir con un periodo de 25 μs el mensaje TA_A informará de que el *stream* desea transmitir *cinco* mensajes por periodo. Lo mismo ocurre con la clase B, solo que el periodo asignado a esta clase es 250 μs. Dado que los periodos en esta primera configuración eran mayores que los previamente mencionados, el código indicaba en el mensaje TA_A que se enviaría *un* mensaje (el mínimo posible) cada 125 μs para los *stream* de clase A, cuando en realidad el periodo de estos mensajes debería ser de 500 μs. Como resultado, para los *bridges* la cantidad de recursos necesaria para cada *stream* era mucho mayor, lo que hacía no planificable esta configuración. La solución consistió en cambiar la configuración de las comunicaciones de forma que se mantuviera la misma utilización de recursos pero evitando utilizar periodos más grandes de 125 μs en el caso de *streams* de clase A y periodos más grandes de 250 μs en el caso de *streams* de clase B.

A continuación se muestra una pequeña descripción de los códigos resultantes.

1) ***Talker****:* A este código se le debe pasar el nombre de la interfaz de comunicaciones utilizada, el identificador del nodo y el protocolo de transporte utilizado.

En el *multiStr_talker* se crean las variables a compartir entre hilos y se ejecutan las funciones de inicialización que solo deban ejecutarse una vez por nodo. Un ejemplo es la variable de la interfaz y su inicialización. En esta capa también se lee el documento de configuración de los *streams* en función del identificador del nodo. En base a esta información se crean diferentes hilos encargados de transmitir cada uno un tipo de *stream* concreto con unas características concretas extraídas del documento de configuración.

Cada hilo crea las variables locales que necesita y ejecuta las funciones de inicialización propias de su *stream*. A continuación, crea una estructura para los mensajes, transmite el TA_A y, cuando recibe la respuesta, entra en un bucle en el que rellena los mensajes y los transmite. Cuando el *listener* se desconecta, este código recibe un mensaje indicando dicho evento tras lo cual el programa se cierra.

Además, se ha incluido un *delay* previo al TA_A para tener mayor flexibilidad a la hora de elegir el momento en que el *stream* es declarado en los experimentos. Este es uno de los parámetros de configuración tal y como se ha visto anteriormente en esta subsección y en la Fig. 4. También hay un *clock* que calcula el tiempo transcurrido entre que se transmite el TA_A y que se recibe una respuesta de algún *listener*. Este *clock* se utiliza para calcular el tiempo que tarda el mecanismo de reserva de recursos de SRP en ejecutarse. Estos datos se guardan en un documento .csv para su posterior análisis mediante Excel y/o Matlab.

2) ***Listener****:* A este código se le debe pasar el nombre de la interfaz de comunicaciones utilizada, el identificador del nodo y el nombre del fichero en el que se guardarán los mensajes recibidos.

De forma similar a como ocurre en el *multiStr_talker*, en el *multiStr_listener* se crean las variables a compartir entre hilos y se ejecutan las funciones de inicialización que solo deban ejecutarse una vez por nodo. Un ejemplo, es la variable de la

TABLA I: Características del tráfico utilizado en los experimentos. T es el periodo y P el *payload* de las tramas.

ID	*Talker*	*Listener*	Clase	T (μs)	P (bytes)
F1	N1	*Broadcast*	*Best-effort*	250	256
F2	N1	N2	Clase B	250	64
F3	N3	N2	Clase B	250	64
F4	N3	N2	Clase A	125	32
F5	N1	N5	Clase A	125	32
F6	N1	N2	Clase A	125	32
F7	N1	N2	Clase A	125	32
F8	N4	*Broadcast*	*Best-effort*	250	256

interfaz y su inicialización. También, se lee el documento de configuración de los *streams* en función del identificador del nodo y, en base a esta información, se crean diferentes hilos encargados de recibir cada uno un tipo de *stream* concreto con unas características concretas extraídas del documento de configuración.

Cada hilo crea las variables locales que necesita y ejecuta las funciones de inicialización propias de su *stream*. A continuación el *listener* espera a la recepción de su *stream*, para esto hubo que añadir filtros para que cada hilo acepte solo los *streams* a los que está suscrito. Una vez recibido el TA_A transmite el LA_R. Tras esto, el código entra en un bucle en el que el hilo recibe los mensajes de su *stream* y los escribe en un documento previamente creado. Finalmente, al cerrar el programa del *listener* este manda un mensaje a los *talkers* anunciando su desconexión de las comunicaciones por lo que estos dejan de transmitir y también se cierran. Cuando todos los hilos han sido cerrados se apaga el programa.

V. Configuración de los experimentos

La configuración usada en este trabajo emulaba diferentes situaciones de uso en un sistema automovilístico para evaluar el rendimiento del SRP de AVB en diferentes escenarios. Decidimos utilizar la industria automovilística como ejemplo ya que este es un sector donde se tiene prevista la aplicación del protocolo AVB y/o TSN. Además, es uno de los sectores que podrían beneficiarse del desarrollo de sistemas distribuidos críticos adaptativos, ya que estos pueden dar soporte de manera natural a vehículos autónomos. Los sistemas adaptativos como estos deben responder a las solicitudes de reconfiguración dentro de un tiempo determinado que generalmente está dictado por la dinámica de una aplicación concreta. Por ejemplo, las transmisiones de audio y vídeo de visión trasera están obligadas por la *National Highway Traffic Administration* (NHTSA) [17] de los EEUU a estar disponibles dentro de los 2 segundos posteriores al arranque del automóvil. Por esta razón el rendimiento del SRP es un aspecto relevante a analizar en este ámbito.

A continuación, describiremos la configuración experimental utilizada y los experimentos realizados.

La Tabla I muestra los parámetros utilizados para definir el tráfico de los experimentos. Los *streams* F1 y F8 representan tráfico *best-effort* mientras que los *streams* F2 a F7 representan *streams* producidos por, por ejemplo, sensores y cámaras, y, por tanto, están sujetos a condiciones de respuesta en TR. Para evaluar el tiempo de respuesta a los cambios de configuración del sistema del SRP bajo una alta carga de red, todos los *talkers* han sido ubicados en un extremo de la red (conectados al B1 que se muestra en la Fig. 2), mientras que los *listeners* han sido ubicados en el extremo opuesto (conectados al B3 que se muestra en la Fig. 2). Es importante notar que ningún *stream* presenta un periodo mayor a 250 μs, tal y como se ha mencionado previamente en la Sección IV.

TABLA II: Características del tráfico usado para medir el impacto que el tamaño de la red y la carga de la misma tienen sobre el tiempo de creación de los *streams*. P es el *payload* de los mensajes de datos.

Experimento	*Stream*			Interferencia	
	Talker	*Listener*	Clase	Carga	P (bytes)
Tamaño de la red	N1	N3	Clase A	0%	-
	N1	N4	Clase A		
	N1	N5	Clase A		
Carga de la red	N1	N2	Clase A	0%	-
				50%	256
					1000
				90%	256
					1000

En este trabajo, en primer lugar llevamos a cabo una serie de experimentos preliminares para estudiar el comportamiento del SRP bajo diferentes condiciones de operación. Este primer estudio nos permitió entender un poco mejor el protocolo antes de su análisis en escenarios más complejos. Concretamente, estudiamos cómo el SRP se ve afectado por tres factores diferentes: (i) el tamaño de la red; (ii) la carga de la red; y (iii) el número de *streams* que intentan ser declarados a la vez.

Para analizar los efectos de cada uno de estos factores realizamos experimentos que aislaban dichos efectos. En este sentido, para medir el impacto del tamaño de la red medimos el tiempo que tardaba en crearse solo el *stream* F7 con diferentes *listeners*. Cada pruebas situaba el *listener* un *bridge* más alejado del *talker*. Concretamente, se realizaron 3 pruebas en las que se utilizó el nodo N1 como *talker* y los nodos N3, N4 y N5 respectivamente como *listeners* (Fig. 2). Para estudiar el efecto que la carga de la red tiene sobre el SRP, medimos el tiempo que tardaba en crearse el *stream* F7 definido en la Tabla I con una carga de red del 0%, 50% y 90%. Es importante mencionar que utilizamos el nodo N3 transmitiendo tráfico *best-effort* para crear la carga. Además, utilizamos dos tamaños de trama para este tráfico de carga, unas tramas con un *payload* de 256 bytes y otras tramas con un *payload* de 1000 bytes. La Tabla II muestra un resumen de las características del tráfico utilizado en estos experimentos.

Finalmente, para medir el impacto de declarar varios *streams* a la vez, medimos el tiempo requerido por el *bridge* B1, mostrado en la Fig. 2, para procesar el mensaje TA_A del *stream* F7 con diferente número de *streams* intentando ser declarados a la vez (F1+F2, F1+F2+F3+F4, F1+F2+F3+F4+F5+F6+F8; donde los *streams* del F1 al F8

presentan las características mostradas en la Tabla I).

Tras la realización de estos experimentos preliminares se procedió a analizar el protocolo en escenarios más complejos inspirados en las diferentes situaciones a las que un sistema de estas características se puede ver sometido. Concretamente, el SRP fue estudiado en 3 escenarios diferentes: (i) los nodos solicitan simultáneamente la admisión de los *streams* F2 a F7 de la Tabla I (*Escenario 1*); (ii) los nodos solicitan la admisión de los mismos 6 *streams* pero de manera secuencial, una solicitud tras otra en orden F2, F5, F6, F7, F3, F4 dejando un segundo de espera entre solicitud y solicitud(*Escenario 2*); y, finalmente, (iii) las solicitudes se dividen en dos grupos de 3 *streams* que serán solicitados a la vez, G1 = {F5,F3,F4} y G2 = {F2,F6,F7}, dejando 10 segundos de espera entre las solicitudes de G1 y G2 (*Escenario 3*). Tal y como hemos dicho, estos escenarios emulan situaciones reales. Concretamente, el escenario 1 evalúa el rendimiento en situaciones de arranque del sistema, por ejemplo, cuando se enciende el automóvil, mientras que el escenario 2 analiza la variación en el tiempo de respuesta para una solicitud bajo diferentes cargas de red. Por último, el escenario 3 emula situaciones en las que múltiples sistemas deben activarse simultáneamente, por ejemplo, el encendido de la cámara y sensor trasero al activar la marcha atrás.

VI. Resultados Experimentales

En esta última sección dedicada al análisis experimental, mostraremos y analizaremos los resultados obtenidos en los experimentos previamente descritos. En la Tabla III se pueden ver los diferentes datos obtenidos a partir de los experimentos preliminares, mientras que en las Fig. 5, 6 y 7 se pueden ver las gráficas de probabilidad acumulada del tiempo requerido para la reserva de recursos de los *streams* para dichos experimentos preliminares. Por otro lado, en la Fig. 8 se pueden ver las gráficas de probabilidad acumulada de los tiempos requeridos para la reserva de recursos de los *streams* para los 3 escenarios. A continuación, analizaremos cómo este tiempo se ve afectado por los 3 factores estudiados: (i) el tamaño de la red, (ii) la carga de la red y (iii) el número de *streams* que intentan ser declarados a la vez; y el comportamiento del protocolo en los 3 escenarios analizados: (i) arranque del sistema, (ii) activación de un *stream* con diferente carga de red y (iii) activación de un subconjunto de *streams* de manera simultánea.

A. Tamaño de la red

Tal y como era esperable y se puede ver en los resultados mostrados en la Tabla III y la Fig. 5, tanto la media como la desviación del tiempo requerido para la reserva de recursos de los *streams* aumenta con el número de *bridges* que deben atravesar las peticiones para ir del *talker* al *listener*.

Sin embargo, analizando en detalle los resultados se puede observar como, aunque efectivamente el tiempo medio de reserva de recursos aumenta con cada *bridge*, no lo hace de forma lineal, sino que cada *bridge* adicional afecta en menor medida al tiempo de respuesta del protocolo. Esto podría indicar una estabilización del tiempo requerido para la reserva de recursos de los *streams* para redes grandes. Sin embargo, la desviación sí que parece aumentar exponencialmente, lo cual podría limitar la escalabilidad de este tipo de redes.

B. Carga de la red

En cuanto a los efectos provocados por la carga de la red en el tiempo requerido para la reserva de recursos de los *streams*, los resultados mostrados en la Tabla III y en la Fig. 6 permiten observar diversas características interesantes del SRP.

En primer lugar, se observa como dicho tiempo aumenta con la carga. No obstante, este aumento solo es significativo cuando la carga es muy baja. Para cargas altas de la red, el impacto del aumento de la carga es menor.

En segundo lugar, también es posible observar una ligera reducción en el tiempo requerido para la reserva de recursos de los *streams* cuando el *payload* del tráfico de carga es mayor. Esto podría deberse a que el uso de *payloads* mayores supone una menor cantidad de mensajes por segundo, lo que deja espacios de ancho de banda más grandes en los que poder transmitir las tramas de datos de los *streams* y, por tanto, facilita el proceso de planificación.

C. Número de streams declarados simultáneamente

En la Tabla III y la Fig. 7, podemos ver cómo este factor afecta de forma diferente a como lo hacían los otro factores. En los experimentos anteriores, a medida que se aumentaba el factor considerado, el tiempo requerido para la reserva de recursos de los *streams* aumentaba en menor proporción. Sin embargo, en este caso el efecto es exponencial, provocando aumentos más grandes del tiempo requerido para la reserva de recursos de los *streams* a medida que aumenta el número de *streams* compitiendo por ser declarados.

D. Escenarios 1, 2 y 3

Por último, en la Fig. 8 podemos ver el comportamiento de la red en los distintos escenarios explicados anteriormente. Tal y como podemos observar, el tiempo requerido para la reserva de recursos de los *streams* es mayor en el escenario 1, seguido del escenario 3 y siendo el escenario 2 el escenario con valores de tiempo más pequeños. Esto se corresponde con lo observado en el experimento de la Subsección VI-C, donde el tiempo aumentaba con el número de *streams* que eran declarados a la vez. Además, el aumento de tiempo exponencial con el número de *streams* declarados simultáneamente también se observa en estos escenarios ya que: la media de tiempo en el escenario 2, donde solo se declara un *stream* a la vez, es de unos 200 ms; mientras que en el escenario 3, donde se declaran 3 *streams* a la vez, es de 300 ms; y en el escenario 1, donde se declaran hasta 6 *streams* a la vez, el tiempo medio es de 600 ms.

En cuanto a los otros factores, vemos que estos no son apenas apreciables en ninguno de los 3 escenarios, a excepción del efecto de aumento de carga el cual puede apreciarse ligeramente en el escenario 3, donde vemos que en presencia de poca carga los *streams* se crean más rápido. Sin embargo, de nuevo vemos como el tiempo termina estabilizándose a medida que los *streams* son creados.

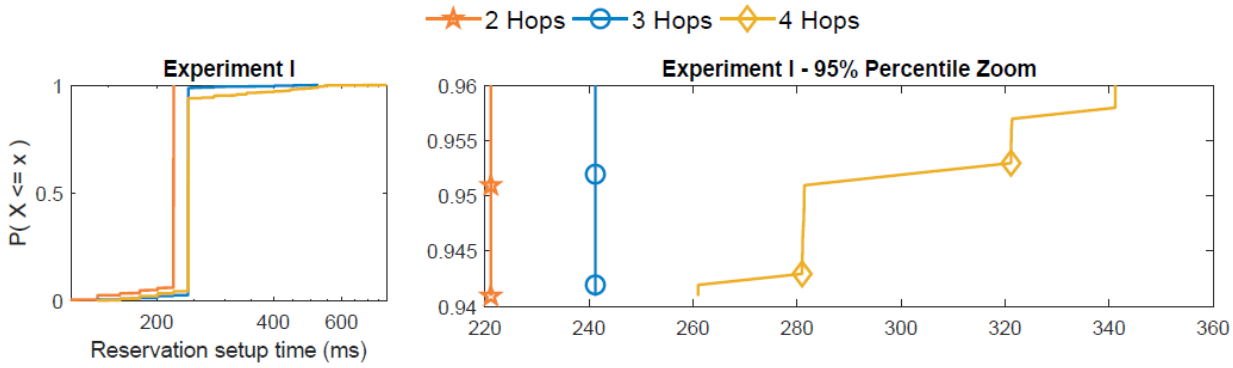


Fig. 5: Tiempo requerido para la reserva de recursos de los *streams* en función del tamaño de la red.

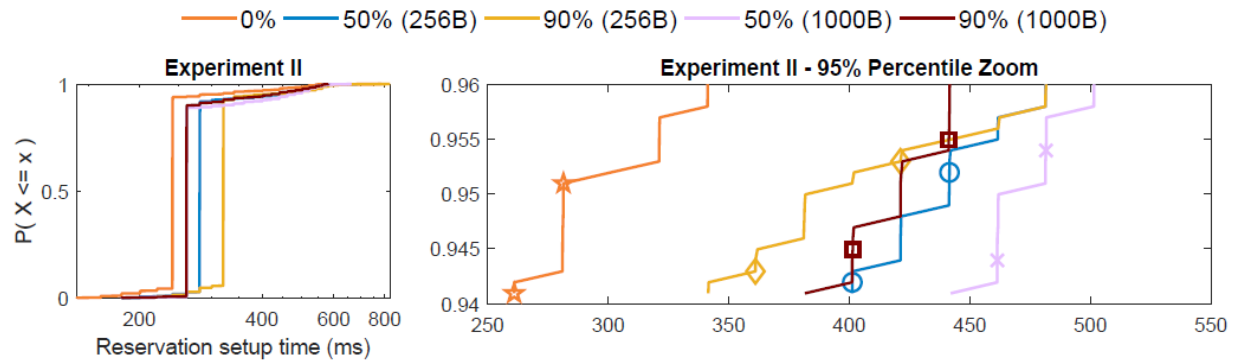


Fig. 6: Tiempo requerido para la reserva de recursos de los *streams* en función de la carga de la red.

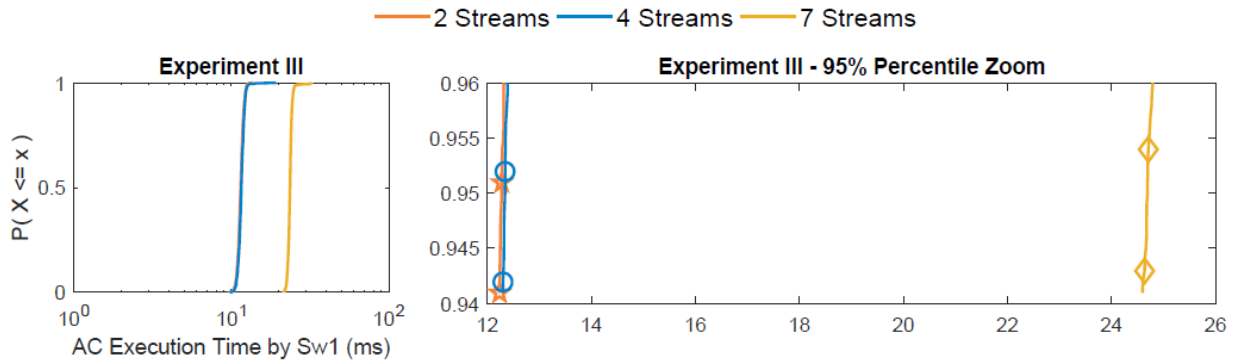


Fig. 7: Tiempo requerido para la reserva de recursos de los *streams* en función del número de *streams* que intentan declarar a la vez en la red.

VII. Descripción general de Uppaal

A continuación, procederemos a exponer el análisis mediante Uppaal realizado en este trabajo. Tal y como se ha dicho anteriormente, el proceso y análisis mediante Uppaal se encuentra descrito en 4 secciones: la presente sección, la Sección VII, que proporciona una explicación general del funcionamiento del Uppaal *model checker*; la Sección VIII, que describe el modelo de SRP realizado; y las Secciones IX y X discuten la terminación y la consistencia respectivamente.

Uppaal es una herramienta para modelar, validar y verificar de manera formal propiedades en sistemas de TR [7]. En Uppaal los sistemas se modelan como una red de *timed automata* interconectados. Entendiendo por *timed automaton* una máquina de estados finitos extendida mediante: relojes que evolucionan simultáneamente y tipos de datos. Además, Uppaal proporciona un lenguaje formal de *queries* que permite definir las propiedades del sistema que deben ser verificadas. El *model checker* usa el modelo del sistema y las *queries* como inputs para llevar a cabo una comprobación exhaustiva de las propiedades. Es decir, el *model checker* explora todas las rutas de ejecución posibles del modelo para verificar si las propiedades se cumplen o no. Tras esto Uppaal informa

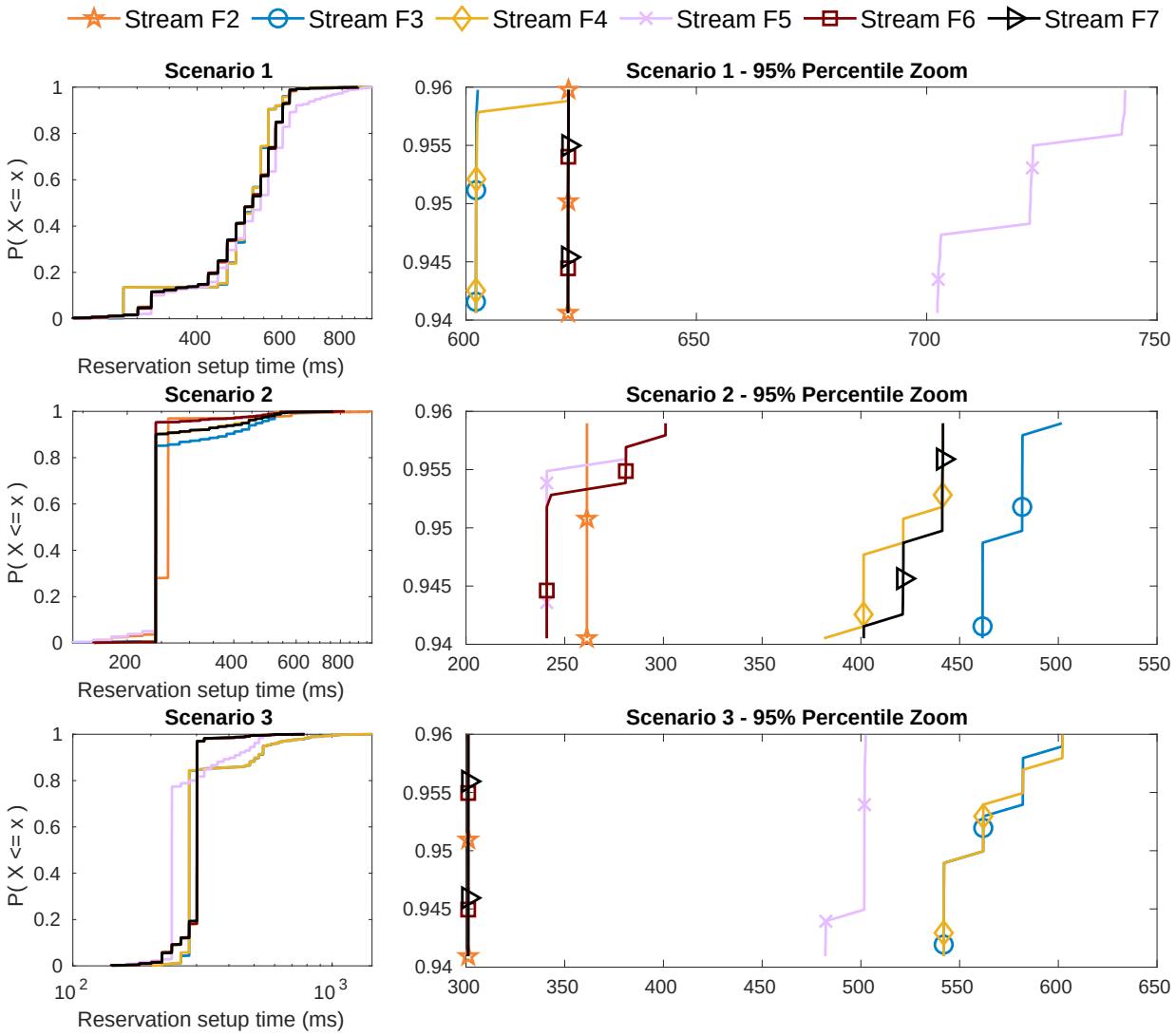


Fig. 8: Tiempo requerido para la reserva de recursos de los *streams* en los 3 escenarios analizados.

TABLA III: Resultados Experimentales.

Condiciones Experimentales	Tiempo de respuesta (ms) Min.	Media	Máx.	σ
Experimento I - Núm. de *Bridges*				
1 *Bridges* (F7:N1→N3)	120.42	217.34	221.14	15.28
2 *Bridges* (F7:N1→N4)	140.48	240.91	521.78	19.65
3 *Bridges* (F7:N1→N5)	140.48	248.75	782.64	49.28
Experimento II - Carga de la red				
Sin carga	140.48	248.75	782.64	49.28
256 PL & 50% CR	180.85	294.03	682.34	56.28
256 PL & 90% CR	220.63	329.35	822.31	52.59
1000 PL & 50% CR	200.54	281.31	662.12	65.48
1000 PL & 90% CR	180.53	276.61	581.94	56.74
Experimento III - Núm. *Streams* creados				
2 *Streams*	9.94	11.47	13.73	0.48
4 *Streams*	9.78	11.58	19.21	0.56
7 *Streams*	21.52	23.59	32.39	0.91

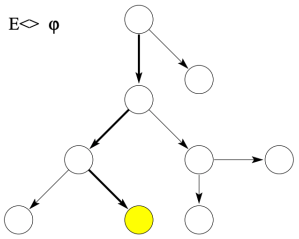
Values obtained from 1000 samples per experiment condition.

sobre el resultado del análisis y, si una propiedad no se cumple, el programa proporciona un ejemplo de ruta de ejecución en la que se viola la propiedad. A continuación explicaremos la herramienta de modelado y el lenguaje de *queries*.

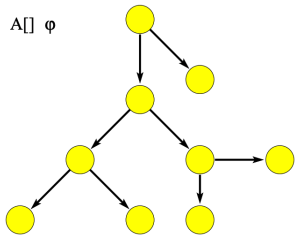
A. *Herramienta de modelado*

Tal y como se ha mencionado anteriormente, en UPPAAL los sistemas se modelan como una red de autómatas. Cada autómata se especifica mediante un *template* que puede ser instanciado varias veces. A su vez, los *templates* se construyen mediante *locations*, *edges*, variables locales y *clocks* locales. Además de esto, diferentes *templates* pueden compartir variables y *clocks* globales, y pueden sincronizarse mediante diferentes tipos de *channels*. Específicamente, en este proyecto se utilizan dos tipos de channels, *binary*, que sincronizan un emisor y un receptor; y *broadcast*, que sincronizan un transmisor con un número arbitrario de receptores.

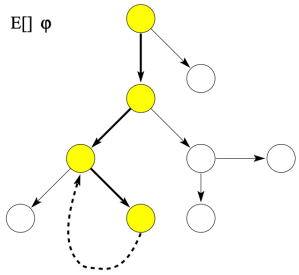
Cada autómata evoluciona a través de un conjunto de *locations*. Hay 3 tipos de *locations*, *normal*, *committed* y *urgent*,



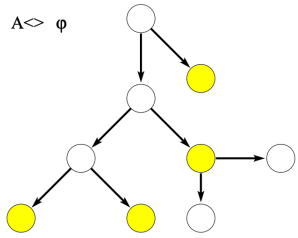
(a) Representación de la propiedad *reachability*.



(b) Representación de la propiedad *safety*.



(c) Representación de la segunda propiedad *safety*.



(d) Representación de la propiedad *liveness*.

Fig. 9: Propiedades que UPPAAL puede evaluar basado de una figura de [7]. Cada figura muestra los diferentes estados a los que el sistema puede evolucionar; mientras que los estados marcados representa cual debe ser la distribución de los estados en los que la *state formula* se cumple para poder afirmar que la propiedad (*reachability*, *liveness* o *safety*) también se cumple.

en función del tiempo que un autómata puede permanecer en ella. Específicamente, un autómata puede permanecer indefinidamente en una ubicación *normal*, a menos que el tiempo de permanencia en dicha *location* esté limitado por el uso de *invariants*. Por otro lado, un autómata deberá abandonar inmediatamente cualquier *location* del tipo *committed* o *urgent*, es decir, el tiempo que un autómata puede permanecer en una *location* de este tipo es 0 y, por tanto, el tiempo no transcurre en este tipo de *locations*. En este sentido, la diferencia entre *commited* y *urgent* es que las primeras son atómicas y las segundas no. Por atómica entendemos que no se ven afectadas por las acciones llevadas a cabo por otros autómatas con *locations* del mismo tipo. Por otro lado, una *location* de cada autómata deberá ser, además, de tipo *initial*, lo cual representa la *location* en la que empezará a funcionar el autómata.

Tal y como hemos dicho, el tiempo que un autómata puede permanecer en una *normal location* puede ser acotado mediante *invariants*. Siendo un *invariant* una expresión que involucra uno o más *clocks*. Los autómatas pueden avanzar por sus ubicaciones usando los *edges*. Los *edges* pueden ser habilitados o deshabilitados mediante el uso de expresiones llamadas *guards*, que son definidas usando variables y *clocks*. Además, los *edges* permiten la asignación de valores a variables cuando estos son tomados.

Finalmente, los autómatas pueden sincronizarse entre ellos tomando ciertos *edges* a la vez. Esto puede hacerse mediante el uso de los ya mencionados *channels*. En estos casos un autómata es el transmisor y uno o mas autómatas conforman los receptores. En el caso de los *binary channels*, los dos, el transmisor y el receptor, esperan a que el otro este en disposición de tomar el *edge* de sincronización. Por otro lado, en los *broadcast channels*, los receptores esperan a que el transmisor tome el *edge*, mientras que el transmisor puede tomar el *edge* en cualquier momento, haya o no autómatas esperando.

B. Lenguaje de queries

UPPAAL presenta un lenguaje a base de *queries* para definir las propiedades que se quieren verificar. Concretamente, existen 3 tipos de propiedades diferentes, *reachability*, la cual es expresada mediante el símbolo `E<>`; *safety*, expresada mediante los símbolos `A[]` o `E[]`; y *liveness*, expresada mediante el símbolo `A<>`. La Fig. 9 muestra estas propiedades mediante un conjunto de estados interconectado. En la figura cada circulo representa un estado del sistema, teniendo en cuenta que los estados están definidos por la *location* en la que se encuentra cada autómata y el valor de sus variables y *clocks*. Además, los círculos amarillos representan los estados en los que se cumple el *state formula* (), tal y como explicaremos más adelante.

La *reachability* (`E<>`) permite evaluar si el modelo alcanza eventualmente un estado concreto, por ejemplo, el avión aterriza. Las propiedades de *safety* permiten evaluar dos tipos de escenarios. El primero, *safety* `A[]`, evalúa si una condición concreta se cumple en todos los estados que el modelo puede alcanzar, por ejemplo, la temperatura del combustible es siempre inferior a un cierto valor. El segundo, *safety* `E[]`, evalúa si siempre existe al menos un camino a través del cual puede evolucionar el sistema en el que cierta condición se cumple, por ejemplo, existe al menos un camino a través del cual el sistema puede evolucionar en el que el avión no se ha estrellado. Finalmente, la *liveness* `A<>` permite determinar si siempre, independientemente del camino recorrido por el modelo, se puede alcanzar un estado que cumple una condición concreta, por ejemplo, un mensaje transmitido a través de una red siempre alcanza su destino.

Tal y como se ha mencionado anteriormente, las propiedades requieren de un *state formula* para ser expresadas. Estas *state formulaes* no son más que código que define las condiciones de los estados de las que hablábamos en el párrafo anterior. Un ejemplo de *state formulae* es i == 7 condición que será verdadera cuando la variable i sea igual a 7. En este sentido, la *state formula* permite evaluar diferentes propiedades en el modelo, tales como i vale siempre 7 o existe al menos un estado en el que i es igual a 7.

En la siguiente sección describiremos el modelo SRP y las propiedades evaluadas mediante la herramienta de modelado y el lenguaje de *queries*.

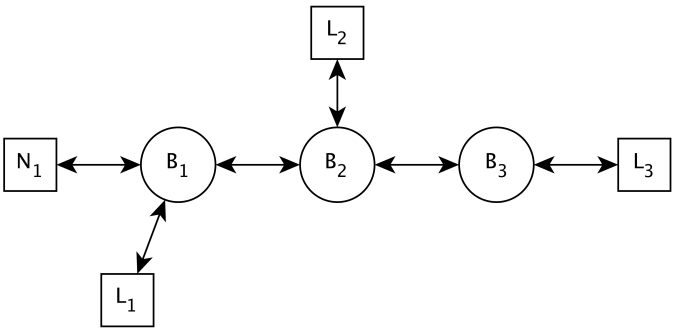


Fig. 10: Topología de la red utilizada en el modelo. Esta consiste en un *talker* y 3 *listeners* conectados a 3 *bridges* en una topología lineal.

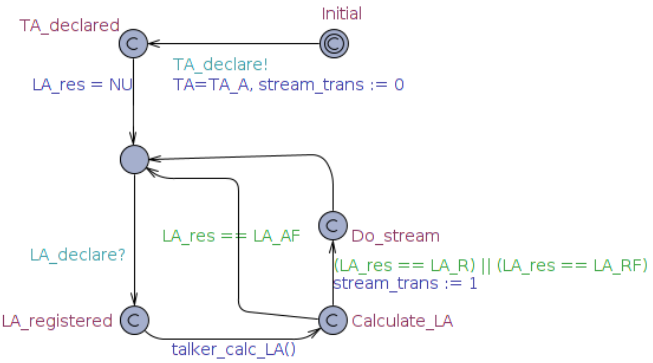


Fig. 11: *Talker template* usado en el UPPAAL *model checker*.

VIII. EL MODELO DE SRP

Nuestro modelo del SRP esta compuesto por 3 *templates* diferentes y esta basado en una versión previa creada por otro alumno de TFM. Estos *templates* llevan a cabo las diferentes acciones relevantes al protocolo realizadas por los *talkers*, *bridges* y *listeners*. Estos *templates* están instanciados una vez en el caso del *talker* y 3 veces en el caso de los *bridges* y los *listeners*, creando una red de comunicaciones tal y como se muestra en la Fig. 10.

Esta topología es suficiente para evaluar la terminación y la consistencia. Específicamente, elegimos 3 *listeners* dado que es habitual hacer triple replicación de ciertos nodos el redes altamente fiables para proveerlas de tolerancia a fallos. En estos casos los 3 nodos ejecutan las mismas operaciones simultáneamente y votan para acordar las acciones a realizar. Decidimos conectar cada *listener* a un *bridge* diferente ya que esto permitía estudiar escenarios más pesimistas que si todos los *listeners* hubieran estado conectados al mismo *bridge*. Finalmente, utilizamos una topología lineal dado que SRP depende de otros protocolos que eliminan los bucles en la red, tales como el *Rapid Spanning Tree Protocol* [18] o el *Shortest Path Bridging protocol* [19], generando una topología lógica lineal desde el punto de vista del SRP.

Es importante notar que, como en cualquier proceso de modelado, tuvimos que abstraer parte del comportamiento del protocolo. En este modelo abstrajimos el número de puertos de cada componente. Concretamente, todos los dispositivos cuenta con un único puerto de transmisión y otro de recepción. Sin embargo, cada puerto es capaz de manejar la recepción y transmisión de atributos de más de un dispositivo.

Hay diversas razones para justificar la abstracción del número de puertos. En primer lugar, modelar la lógica de un *bridge* no es trivial y requiere de un número significativo de *locations* y variables, y esta complejidad aumenta con el número de puertos. En segundo lugar, especificar un número concreto de puertos reduce la generalidad del modelo y el número de casos en los que puede ser utilizado. Por ejemplo, si hubiéramos modelado el *bridge* con tan solo dos puertos no habríamos podido conectar a este un *talker*, un *listener* y otro *bridge* tal y como finalmente hicimos. Por último, aumentar el número de *locations* no solo aumenta la complejidad del modelo, sino que además aumenta el número de estados que el programa debe analizar. Esto puede llevar a una explosión del espacio de estados y, por consiguiente, un consumo de recursos que puede no ser apto para todo tipo de ordenadores.

Por lo tanto, para modelar la transmisión de diferentes atributos a dispositivos diferentes, nosotros incluimos la posibilidad de modificar el tipo de atributo recibido. Por ejemplo, cuando un *listener* o un *bridge* recibe un TA_A, este puede modificarlo y convertirlo en un TA_F, emulando que el puerto a traves del cual se transmitió el mensaje no tenía suficientes recursos. No obstante, si un dispositivo transmite un TA_F los dispositivos que lo reciban no podrán modificarlo recibiendo así el TA_F transmitido.

Otra abstracción consiste en que nosotros no modelamos la transmisión de frames de datos, dado que esto habría supuesto el modelado del *Credit Based Shaper* [3] y otros mecanismos que aumentarían la complejidad, desvirtuarían el modelo y no aportarían mayor precisión al análisis del protocolo. En su lugar, la transmisión de datos viene representada mediante una *location* en el *talker*.

A continuación describiremos cada uno de los templates utilizados en el modelo.

A. Talker template

La Fig. 11 muestra el template del *talker*. El *talker* comienza en una *location commited*. Esto se debe a que, si la *location* no fuera de este tipo, el *template* podría permanecer en en dicha *location* de forma indefinida generando un bloqueo en el modelo. Por tanto, el modelo comienza con la transición desde la *location commited initial* a la *location TA_declared* en el *talker*. Este *edge* inicia la transmisión del mensaje *Talker Atribute* (TA) mediante el *binary channel TA_declare*. Tras esto el autómata ingresa en un bucle en el que recibirá los mensajes *Listener Atribute* (LA) y calculará si debe comenzar con la transmisión del *stream* o no. Si el resultado del cálculo es un LA_R o LA_RF el *template* del *talker* pasará a la *location Do_stream*, la cual indica el comienzo de la transmisión de datos.

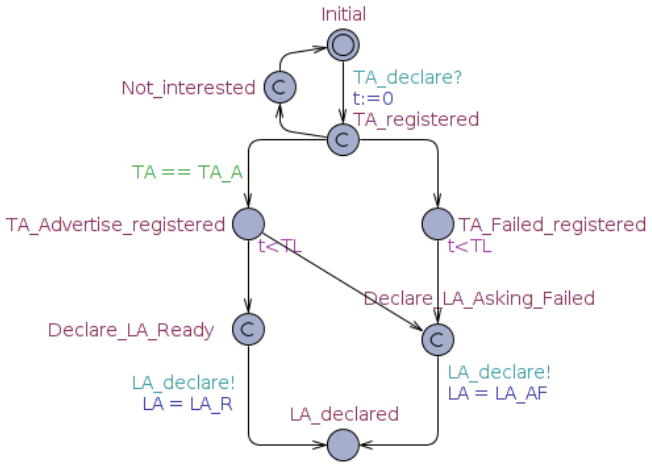


Fig. 12: Listener template using the UPPAAL *model checker*.

B. Listener template

La Fig. 12 muestra el template del *listener*. Inicialmente el *listener* espera hasta que recibe un TA. Tal y como se explicó en la Sección III, cuando el *listener* recibe un mensaje TA este lo registra. Después, el *listener* puede: ignorar el mensaje TA sino está interesado en el *stream*, registrar el mensaje TA si recibió un mensaje TA_A y además está interesado en el *stream* o registrar un mensaje TA_F si estaba interesado en el *stream* y recibió un mensaje TA_F. Finalmente, el *listener* interesado en el *stream* solo transmitirá un mensaje LA_R si registro un mensaje TA_A y, además, tiene suficientes recursos. De lo contrario, transmitirá un mensaje LA_AF.

Al principio de esta sección mencionamos que cambiamos el lugar donde se toman las decisiones relacionadas con los recursos del puerto como herramienta de abstracción. En este sentido, la recepción de este *template* es un buen ejemplo de ello. Tal y como veremos en la siguiente sección, los *bridges* transmiten lo mismo que recibieron. Por tanto, es en la recepción del *listener* donde el TA puede cambiar. Si el *listener* recibe un mensaje TA_A él puede registrar un TA_A o un TA_F dependiendo de los recursos del puerto del *bridge* conectado al *listener* en cuestión. Sin embargo, si el *listener* recibe un TA_F este siempre registrará dicho valor.

C. Bridge template

La Fig. 13 muestra el *template* del *bridge*. El *bridge* espera en la *initial location* hasta que recibe un TA o un mensaje LA. Si recibe un TA el autómata tomará el *edge* de la izquierda. Este *edge* registra el mensaje TA recibido y, en función de su valor, retransmitirá un TA_A o un mensaje TA_F. Una vez el mensaje ha sido transmitido, el autómata volverá a la *initial location*. Este camino recién explicado presenta otro caso de simplificación. Y es que, en la recepción, el *bridge* puede cambiar el valor del mensaje recibido de un TA_A a un TA_F representado la falta de recursos en el puerto que transmitió el mensaje a este *bridge*, el cual puede estar situado en el *talker* o en otro *bridge* que se encontraba entre este *bridge* y el *talker*. Además, el cambio de valor en la transmisión no se realizará en este *bridge* sino en la recepción de los *bridges* o *listeners* conectados al mismo. Si estando en la *initial location* el *bridge* recibe un mensaje LA el autómata tomará el *edge* derecho. Este *edge* registra el mensaje LA y, dependiendo de los mensajes recibidos y los recursos del *bridge*, este transmitirá un mensaje LA_R, LA_RF o LA_AF. Este camino es muy importante para entender la verificación de las propiedades por lo que a continuación sera explicado con más detalle.

En primer lugar, una vez registrado el mensaje LA el autómata entra en un bucle de recepción de los mensajes LA. En este bucle el *array prev_type* registra el origen del mensaje. Él registra el valor *type_B* si el mensaje proviene del siguiente *bridge* en dirección a los *listeners* o un *type_L* si el mensaje proviene del *listener* conectado al *bridge*. A continuación el *array LA_prev* registra el contenido del mensaje (un LA_R, LA_RF o LA_AF). Tras esto el *bridge* llegará a la *location* llamada *Wait*. En esta el *bridge* puede: o continuar recibiendo más mensajes que reinician el bucle previamente explicado o puede pasar a la fase de transmisión. En esta última fase el *bridge* transmitirá un mensaje LA diferente dependiendo de los mensajes LA recibidos y de los recursos del *bridge*.

IX. EVALUACIÓN DE LA TERMINACIÓN

Tal y como se introdujo en la Sección I, la terminación acostumbra a ser una propiedad muy importante en los sistemas críticos distribuidos. Así pues, el SRP debe cumplir esta propiedad si quiere poder ser utilizado en este tipo de sistemas. En este trabajo se diferencia dos niveles de terminación.

El primero se corresponde con la terminación para la aplicación. Muchas aplicaciones críticas requieren conocer el estado de la reserva de recursos para tomar importantes decisiones. En este sentido, la falta de terminación puede causar el mal funcionamiento de aplicaciones de este tipo. A este nivel, SRP debe asegurarse de que el proceso de reserva acaba (creando el *stream* o no) en un tiempo acotado.

El segundo se corresponde con la terminación a nivel de infraestructura involucrada en la reserva. Entendemos como infraestructura al conjunto de *bridges* y nodos de la red. Puesto que estos dispositivos toman decisiones relacionadas con el proceso de reserva de recursos, es importante proveer a estos terminación para evitar efectos imprevistos e indeseables en futuras reservas.

Sin embargo, a pesar de la importancia de esta propiedad, en este trabajo encontramos problemas de terminación en ambos niveles. Es importante notar que gran parte de los problemas detectados se debían al hecho de que en el SRP los *listeners* que no están interesados en un *stream* no informan ni a los *bridges* ni al *talker* de esta situación. A continuación se expondrán los diferentes problemas detectados así como diferentes posibles soluciones.

A. Terminación para la Aplicación

En primer lugar discutiremos la terminación a nivel de aplicación. En este trabajo se observaron escenarios en los

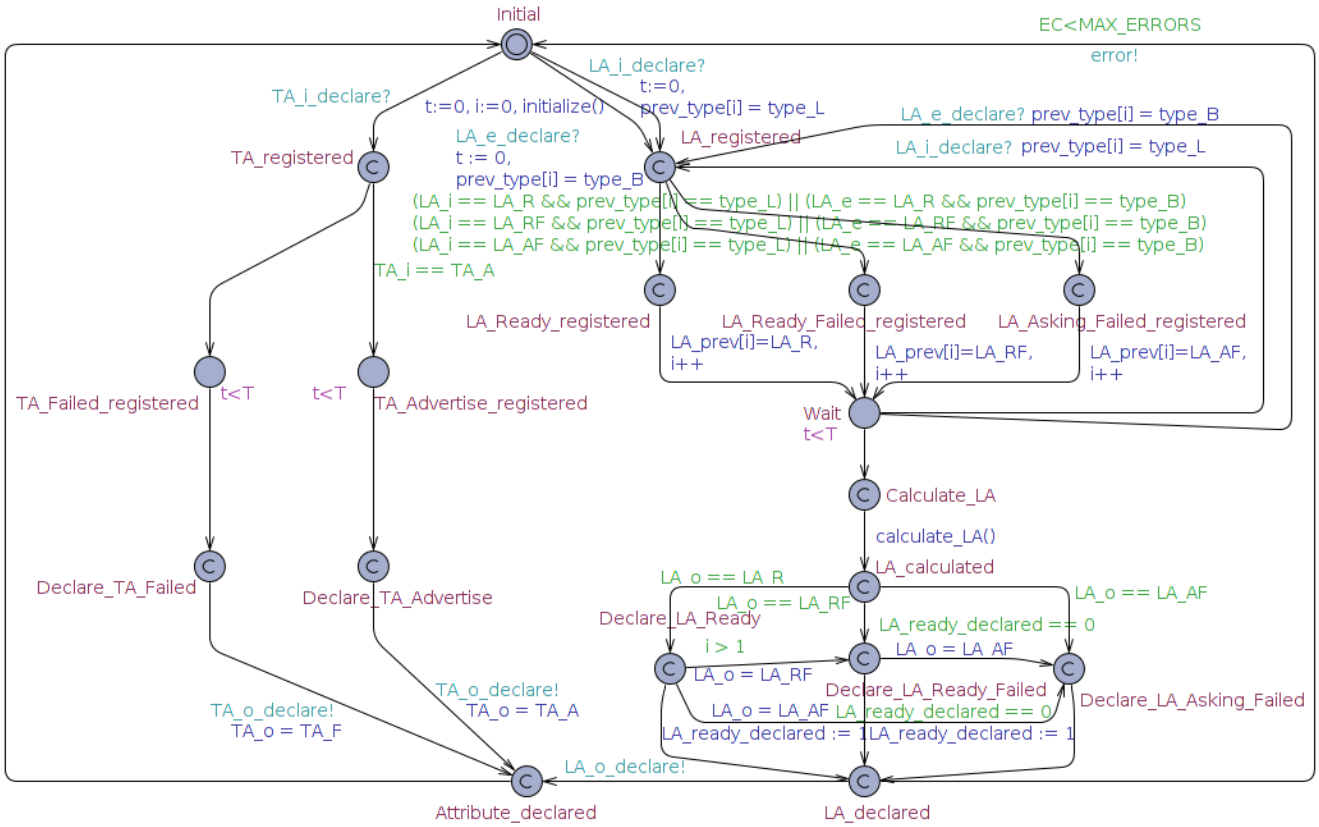


Fig. 13: *Bridge template* usado en el UPPAAL *model checker*.

que el *talker* no recibía ninguna respuesta por parte de los *listeners*, incluso en ausencia de fallos, de forma que permanecía esperando indefinidamente. En un análisis más detallado de dicho escenario nos dimos cuenta de que este se producía cuando no había ningún *listener* interesado en el *stream*. Tal y como se mencionó anteriormente, muchas aplicaciones críticas requieren conocer el resultado de la reserva para tomar decisiones importantes. En este sentido, la falta de terminación puede causar un mal funcionamiento de este tipo de aplicaciones, como bloquear el proceso de decisión o conducir a decisiones incorrectas debido a la falta de conocimiento del estado de la reserva de recursos.

Una posible solución podría consistir en introducir un *timer* en el *talker*. En este sentido, si el *talker* no recibe ninguna respuesta por parte de los *listeners* antes de que el *timer* expire, él deberá eliminar el *stream* usando el mecanismo de *unadvertise stream* disponible en SRP explicado en la Sección III. El valor asignado al *timer* debe ser ajustado acorde a la topología de la red, debiendo asegurar que el tiempo será lo suficientemente largo como para asegurar que la respuesta del *listener* más lejano, en caso de que dicha respuesta se produzca, podrá llegar al *talker* antes de que el *timer* expire.

B. Terminación para la Infraestructura

A continuación discutiremos el segundo nivel de terminación, el cual se corresponde con la terminación para la infraestructura. Un *bridge* que reenvía la solicitud transmitida por un *talker* se mantendrá a la espera de la respuesta de los *listeners* de forma indefinida. Además, los *bridges* registran los atributos del *talker* en todos sus puertos de reenvío y esto lo hacen para todos los *talkers* que quieran transmitir. Mediante el modelo se detectaron escenarios en los que el *bridge* enviaba un TA_A y no recibía ninguna respuesta, porque ninguno de los *listeners* conectados al *bridge* (directa o indirectamente) estaban interesados en dicho *stream*. Además, estos escenarios podían llegar a producirse incluso en escenarios en los que se alcanzaba terminación para la aplicación. Este tipo de problemas en la terminación, aunque no son tan crítico como los producidos a nivel de aplicación, puede causar un uso innecesario de la memoria de los *bridges* y, a largo plazo, impedir la creación de *streams* para los que sí existen *listeners* interesados.

Nosotros hemos propuesto dos soluciones diferentes a este problema. El primero consiste en introducir un *timer* por cada *stream* en cada *listener* y en cada puerto de cada *bridge*. En estos casos, la expiración del *timer* provocará la eliminación del registro del *talker*. El valor del *timer* en los puertos de los *bridges* debe ser suficientemente grande como para asegurar que estos podrán recibir la respuesta de los *listeners* interesados antes de que la petición sea eliminada. Por otro lado, en el caso de los *listeners* el tiempo asignado a los *timers* únicamente debe ser suficientemente grande como para asegurar que el *listener* podrá responder a la petición antes de

que el registro del *stream* sea eliminado.

La segunda solución consiste en introducir *timers* únicamente en los puertos de los *bridges*, y no en los *listeners*. En su lugar, las declaraciones en los *listeners* serán eliminadas cuando estos reciban un tipo de mensaje especial transmitido por los *bridges*. Concretamente, el *bridge* más cercano al *talker* que no haya recibido ninguna respuesta deberá eliminar el registro de su memoria y iniciar la transmisión de un mensaje que elimine el registro en los diferentes dispositivos conectados a él. Esto sería similar a lo que el *talker* hace cuando ejecuta el mecanismo de *unadvertise stream* mencionado en la Sección III. Igual que en la primera solución, el valor del *timer* en los puertos de los *bridges* debe ser suficientemente grande como para asegurar que estos podrán recibir la respuesta de los *listeners* interesados en el *stream* antes de que la petición sea eliminada. Sin embargo, existe un nuevo requisito, y es que el primer *timer* en expirar debe ser el del *bridge* que se encuentre más cerca del *talker*. De lo contrario, más de un *bridge* podría iniciar la transmisión del mensaje encargado de la eliminación de los registros del *stream*. Es importante notar que establecer el mismo tiempo de espera a todos los *timers* sería una configuración simple que cumple los dos requisitos mencionados, aunque no es ni la óptima ni la única configuración posible.

La ventaja de la segunda solución, con respecto a la primera, es que los nodos no requieren de un *timer* por cada *stream* que vayan a recibir. Sin embargo, los *bridges* deberán ser capaces de crear mensajes capaces de eliminar los registros del *stream* en otros dispositivos.

X. Evaluación de la Consistencia

Tal y como se introdujo en la Sección I, la consistencia también tiende a ser una propiedad importante en los sistemas críticos distribuidos. En este trabajo diferenciaremos dos niveles de consistencia. El primer nivel se refiere a la consistencia para la aplicación. Los sistemas distribuidos suelen ejecutar aplicaciones en las que se requiere que diferentes nodos realicen acciones coordinadas. En estas aplicaciones la consistencia en las comunicaciones es clave para garantizar el correcto funcionamiento del sistema. En este sentido, el primer paso para conseguir comunicaciones consistentes es reservar los recursos de la red de forma consistente. Por tanto, a este nivel, SRP debe garantizar que todos, o ninguno, de los *listeners* interesados tengan reservados los recursos necesarios para la comunicación.

El segundo nivel se refiere a la consistencia para la infraestructura, incluso si la reserva en sí no es consistente. Este segundo nivel fue puesto en consideración para preservar el comportamiento del SRP, es decir, permitir que el *talker* pueda comunicarse con algunos *listeners*, incluso si no todos los *listeners* interesados en el *stream* pueden recibir. En este escenario, nosotros proponemos proporcionar una visión consistente de las reservas a todos los dispositivos y relevar a la aplicación la decisión de hacer uso o no del *stream* en cuestión.

Nosotros encontramos algunos problemas de consistencia en ambos niveles incluso en ausencia de fallos. Los problemas detectado se deben principalmente al hecho de que la información relacionada con la reserva de recursos se propaga unidireccionalmente. En otras palabra, los mensajes TA transmitidos por los *talkers* siempre son transmitidos en dirección a los *listeners*, mientras que la respuesta de los *listeners* y *bridges* solo se transmite en dirección al *talker*. Esto provoca que no todos los dispositivos involucrados en la reserva de recursos reciban la misma información.

A. Consistencia para la Aplicación

Como se dijo anteriormente, en este nivel de consistencia es importante que todos los *listeners* interesados en el *stream* reciban lo mismo. Esto es especialmente relevante cuando se trata de nodos que realizan acciones coordinadas. Sin embargo, tal y como se explicó en la Sección III, en el SRP los recursos pueden ser reservados solo para un subconjunto de *listeners* interesados, los cuales tiene recursos suficientes, mientras que otros *listeners* interesados, debido a la falta de recursos, no podrán recibir los mensajes de datos de dicho *stream*. En este caso se genera una inconsistencia en el intercambio de datos. Esto muestra que, para el SRP, es más importante comenzar las comunicaciones, aunque sea con solo unos pocos de los *listeners* interesados, que realizar las comunicaciones de forma consistente, recibiendo o todos los *listeners* interesados o ninguno.

Además, detectamos que, incluso cuando todos los *listeners* interesados en el *stream* tenían recursos suficientes, no se garantizaba la consistencia durante todo el proceso de transmisión de datos. Esto se debe a dos razones. La primera es que entre el *talker* y los diferentes *listeners* existen diferencias en cuanto a la longitud y el retraso *end-to-end*; y la segunda, es que el *talker* comienza a transmitir tan pronto como recibe la respuesta favorable de al menos un *listener*. Por lo tanto, algunos *listeners* interesados en el *stream*, y con recursos suficientes, pueden dejar de recibir los primeros mensajes de datos debido a que no ha habido tiempo suficiente para que la reserva de recursos necesaria para que estos *listeners* reciban se realice.

Una posible solución a este problema consistiría en, de nuevo, utilizar un *timer* en el *talker*. Este debería ser ajustado en función de la topología, para garantizar que el *listener* más alejado tenga tiempo suficiente para comunicar al *talker* su intención de recibir el *stream*. Cuando el *timer* expire el *talker* deberá comprobar el resultado de la reserva. Si la reserva de recursos a fallado para al menos un *listener* (el *talker* recibe algún mensaje LA_RF o LA_AF) o si ningún *listener* está interesado en el *stream* (el *talker* no recibe ninguna respuesta) el *talker* deberá eliminar el *stream* utilizando el ya mencionado mecanismo de *unadvertise stream* explicado en la Sección III. Por otro lado, si la reserva se realizó correctamente para todos los *listeners* (el *talker* solo recibe mensajes LA_R) este comenzará la transmisión del *stream*. De esta forma es posible garantizar la consistencia para la aplicación ya sea porque todos los *listeners* han podido reservar correctamente

los recursos o porqué, por el contrario, ninguno a podido. Es importante notar que el hecho de que el *talker* elimine el *stream* también en el caso en que ningún *listener* este interesado en el *stream*, hace que esta solución incluya a su vez la solución propuesta para solventar el problema del nivel de terminación para la aplicación, solución mostrada en la Subsección IX-A.

B. Consistencia para la Infraestructura

Aunque la solución anterior elimina las inconsistencias para la aplicación, también restringe ciertas características de SRP que pueden ser interesantes para cierto tipo de aplicaciones. Por esta razón, a partir de ahora abandonaremos la perspectiva todo o nada de la subsección anterior y nos centraremos en garantizar que todos los dispositivos compartan la misma visión de la red, es decir, que todos los dispositivos sepan que *listeners* pueden reservar recursos para el *stream* y cuales no. Es importante notar que SRP no confiere este nivel de consistencia nunca ya que, aunque el *talker* puede saber que hay *listeners* interesados en el *stream*, este no sabe ni cuantos ni quienes son los interesados.

Además, no todos los dispositivos reciben la misma información. Esto también afecta a los *bridges* dado que ellos toman muchas decisiones usando principalmente información interna del mismo y comparten muy poca información acerca de las decisiones tomadas con el resto de dispositivos que conforman la red. Esto puede llevar a inconsistencias en las reservas de los *bridges*, dado que la información que dos *bridges* reciben de otro *bridge* varía mucho dependiente de si estos se encuentran en el camino hacia el *talker* o hacia el *listener*.

Utilicemos un ejemplo para entender el tipo de problemas que este tipo de inconsistencia en los *bridges* puede causar en la red. Asumamos que tenemos un *talker* conectado a un *listener* a través de dos *bridges* en una topología lineal (T-B1-B2-L). Cuando el *listener* responde a la declaración del *stream*, y el *bridge* B2 tiene suficientes recursos, este último los reserva. Sin embargo, si el *bridge* B1 no tiene suficientes recursos no puede reservarlos lo que, unido al hecho de que B1 no informa de esta decisión al B2, provoca que el *bridge* B2 este reservando recursos incluso si el *stream* finalmente no ha sido creado.

Una solución podría consistir en introducir dos listas en las respuestas de los *listeners*, en los *bridges* y en los *talkers*. Una de las listas contendría los identificadores de los *listeners* que pueden recibir (de aquí en adelante *List of Nodes with Resources* (LNR)) mientras que la otra contendrá los identificadores de los *listeners* que no pueden recibir (de ahora en adelante *List of Nodes with No Resources* (LNnR)). Los *listeners* interesados en en el *stream* incluirían su identificador en la respuesta en la lista que le correspondiera, en función de sus recursos y el mensaje recibido en la declaración del *talker* (TA_A o TA_F).

Los *bridges*, por su parte, actualizarían sus listas internas cada vez que recibieran una respuesta de un *listener*. Si la respuesta del *listener* no es modificada en el *bridge* (ver Sección III para más detalles) el *bridge* actualizará sus listas internas con los identificadores recibidos tal y como se han recibido, es decir, los identificadores que se encontraran en la LNR del mensaje se guardarían en la LNR del *bridge* y los identificadores que se encontraran en la LNnR del mensaje se guardarían en la LNnR del *bridge*. Por el contrario, si un *bridge* modifica la respuesta de un *listener* por una falta de recursos en el puerto de recepción del mensaje, este añadirá los identificadores presentes en ambas listas del mensaje recibido en su propia LNnR interna. En este sentido, cada vez que el *bridge* tenga que enviar la respuesta combinada de las respuestas recibidas, este incluirá sus propias listas al mensaje. En cuanto al *talker*, este también actualizará sus listas internas igual que hace el *bridge* cuando tiene suficientes recursos en el puerto de recepción. De esta forma, el *talker* podrá saber que *listeners* podrán recibir las tramas de datos del *stream* y cuales no.

Finalmente, el *talker* propagaría esta información a través de la red. Específicamente, este utilizaría un *timer* para esperar a la respuesta de todos los *listeners*. El valor dado a este *timer*, igual que los otro vistos en otras soluciones, debe garantizar que ha transcurrido suficiente tiempo como para asegurar que todas las respuestas de los *listeners* han llegado al *talker*. Una vez el *timer* expire, el *talker* transmitirá sus propias listas al resto de dispositivos de forma que estos tengan la misma visión de la red. Esto solucionaría la asimetría en la información que diferentes dispositivos reciben. De esta forma, la aplicación puede tomar decisiones teniendo una visión completa del estado de la red y los *bridges* pueden eliminar sus registros y reservas de recursos cuando estos no vayan a ser utilizados.

Sin embargo, esta solución puede ser compleja y requiere aplicar muchos cambios al SRP. Así pues, si no es necesario que los nodos tengan una visión consistente de la red, pero queremos evitar que los *bridges* malgasten recursos, es posible utilizar una solución más sencilla. Básicamente la solución consistiría en que, si un *bridge* recibe una respuesta de un *listener*, indicando que hay suficientes recursos, pero este no tiene suficientes recursos en el puerto de recepción, que informe a los dispositivos de los cuales a recibido el mensaje de que no tiene suficientes recursos y que por tanto no van a recibir el *stream*.

XI. Conclusiones y Trabajo Futuro

El SRP es uno de los protocolos más importantes de AVB y TSN, dos conjuntos de estándares que están revolucionando el mundo de las comunicaciones empotradas e industriales, en especial TSN, que es un firme candidato a liderar la siguiente revolución industrial en el contexto de la nueva Industria 4.0 [20].

El análisis de estos estándares y sus protocolos no solo sirve para (i) conocerlos mejor y ver para qué y para qué no pueden ser útiles, sino que también puede (ii) ser una herramienta para buscar defectos y hallar soluciones que los haga adecuados para el mayor número posible de aplicaciones. En este sentido, en este trabajo hemos hecho ambas cosas: (i) analizar el protocolo en diferentes escenarios en los que

el tiempo de respuesta era un factor clave para dar soporte a la adaptabilidad; y (ii) buscar problemas y soluciones en propiedades clave para los sistemas críticos como son la consistencia y la terminación.

Por un lado, el análisis experimental muestra que el SRP presenta tiempos de respuesta a los cambios de configuración del sistema que varían poco cuando las redes son grandes o presentan mucha carga en la red. Esto hace que las redes AVB o TSN puedan escalar en tamaño y carga de la red sin demasiados perjuicios al tiempo de reconfiguración. En otras palabras, si SRP es capaz de cumplir los requisitos de TR en una red grande y con mucha carga de red, aumentar el tamaño o la carga de dicha red no debería suponer ningún problema. Sin embargo, se ha observado que la concurrencia de declaraciones de *streams* puede afectar drásticamente a los tiempos de respuesta a los cambios de configuración del sistema. Por esta razón, aumentar el número de *streams* que puedan ser declarados simultáneamente es arriesgado ya que los tiempos de configuración crecen rápidamente y pueden llegar a incumplir los requisitos de TR del sistema. Además de estas conclusiones, en este trabajo hemos calculado los tiempo de reconfiguración del sistema (desde 200 ms hasta 600 ms en función, principalmente, del número de *streams* declarados simultáneamente) los cuales pueden servir como referencia a un usuario de sistemas empotrados para determinar si AVB o TSN son aptos para su aplicación.

Por otro lado, las observaciones realizadas sobre el modelo del SRP en UPPAAL muestran que dicho protocolo no presenta ni la propiedad de terminación y ni la de consistencia. Esto se debe principalmente a la falta de respuesta por parte de los *listeners* que no están interesados en el *stream* declarado y a la transmisión unidireccional de las decisiones tomadas en el proceso de reserva de recursos respectivamente. En este sentido, en este trabajo, además, hemos planteado soluciones que podrían permitir a SRP adquirir estas propiedades en diferentes niveles (terminación y consistencia a nivel de aplicación y/o infraestructura).

En un futuro buscaremos implementar las soluciones planteadas para solucionar los problemas de terminación y consistencia en el modelo UPPAAL, con el fin de verificar formalmente su correcto funcionamiento. Por otro lado, desde hace un tiempo estamos en colaboración con un grupo de la Universidade de Aveiro (Portugal) para realizar una comparación entre los tiempos de respuesta de los cambios de configuración realizados con SRP (analizado en este trabajo) y los del *Hard Real-Time Ethernet Switch* (HaRTES) [21], un protocolo de comunicaciones con características similares. Como resultado de esta colaboración estamos acabando un *paper* de revista, cuya referencia se indica al final del Apéndice A.

APÉNDICE A: PUBLICACIONES DE ESTE TRABAJO

D. Bujosa, D. Cavka, I. Álvarez, J. Proenza. *First Analysis of the AVB's Stream Reservation Protocol in the Context of TSN*. **Presentación oral sin publicación en las actas** en EUROMICRO 17th International Workshop on Real-Time Networks (RTN 2019), Barcelona, 2019

D. Bujosa, I. Álvarez, D. Cavka, J. Proenza. *Analysing Termination and Consistency in the AVB's Stream Reservation Protocol*. **Publicación en las actas** del IEEE 24th International Conference on Emerging Technologies and Factory Automation (ETFA 2019), Zaragoza, 2019.

I. Álvarez, L. Silva, **D. Bujosa**, P. Pedreiras, J. Proenza, L. Almeida. *Comparison of the Performance of the Admission Control Process in HaRTES and AVB*. **Artículo en preparación para su envío a una revista.**

APÉNDICE B: ACRÓNIMOS

AVB *Audio Video Bridging*
HaRTES *Hard Real-Time Ethernet Switch*
IEEE *Institute of Electrical and Electronics Engineers*
LA *Listener Atribute*
LA_AF *Listener Asking Failed*
LA_R *Listener Ready*
LA_RF *Listener Ready Failed*
LNnR *List of Nodes with No Resources*
LNR *List of Nodes with Resources*
NHTSA *National Highway Traffic Administration*
QoS *Quality of Service*
SRP *Stream Reservation Protocol*
TA *Talker Atribute*
TA_A *Talker Advertise*
TA_F *Talker Advertise Failed*
TG *Task Group*
TR Tiempo Real
TSN *Time-Sensitive Networking*

AGRADECIMIENTOS

Este trabajo ha sido financiado en parte por la Agencia Estatal de Investigación (AEI) y en parte por el Fondo Europeo de Desarrollo Regional (FEDER) a través de la subvención TEC2015-70313-R (AEI/FEDER, UE).

REFERENCIAS

[1] (2019, March) IEEE Audio and Video Bridging Task Group. https://1.ieee802.org/tsn/.

[2] "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks," *IEEE Std 802.1AS-2011*, pp. 1–292, March 2011.

[3] "IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams," *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pp. C1–72, Jan 2009.

[4] "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.

[5] "IEEE Standard for Local and metropolitan area networks–Audio Video Bridging (AVB) Systems," *IEEE Std 802.1BA-2011*, pp. 1–45, Sept 2011.

[6] "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)*, pp. 1–208, Oct 2018.

[7] G. Behrmann, A. David, and K. G. Larsen, *A Tutorial on Uppaal*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 200–236. [Online]. Available: https://doi.org/10.1007/978-3-540-30080-9_7

[8] 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*, vol. 101, no. 11, pp. 2339–2354, Nov 2013.

[9] 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*, vol. 21, no. 1, pp. 88–145, Firstquarter 2019.

[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)*, Oct 2016, pp. 47–51.

[11] O. Kleineberg, P. Fröhlich, and D. Heffernan, "Fault-tolerant Ethernet networks with Audio and Video Bridging," in *ETFA2011*, Sept 2011, pp. 1–8.

[12] P. Meyer, T. Steinbach, F. Korf, and T. C. Schmidt, "Extending IEEE 802.1 AVB with Time-Triggered Scheduling: A Simulation Study of the Coexistence of Synchronous and Asynchronous Traffic," in *2013 IEEE Vehicular Networking Conference*, Dec 2013, pp. 47–54.

[13] (2019) Adaptador intel ethernet i210-t1 para servidores. https://ark.intel.com/content/www/es/es/ark/products/68668/intel-ethernet-server-adapter-i210-t1.html.

[14] (2019) Nanl-b500g-re. https://www.hilscher.com/products/product-groups/analysis-and-data-acquisition/ethernet-analysis/nanl-b500g-re/.

[15] (2019) Openavnu. https://github.com/AVnu/OpenAvnu/blob/master/README.rst.

[16] (2019) Avnu alliance. https://avnu.org.

[17] A. Gothard, R. Kreifeldt, and C. Turner, "AVB for Automotive Use White Paper," AVnu Alliance, Tech. Rep., Oct 2014. [Online]. Available: \url{https://avnu.org/wp-content/uploads/2014/05/2014-11-20_AVnu-Automotive-White-Paper_Final_Approved.pdf}

[18] "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)*, pp. 1–281, June 2004.

[19] "IEEE Standard for Local and Metropolitan Area Networks–Media Access Control (MAC) Bridges and Virtual Bridges," *IEEE Std 802.1Q, 2012 Edition, (Incorporating IEEE Std 802.1Q-2011, IEEE Std 802.1Qbe-2011, IEEE Std 802.1Qbc-2011,IEEE Std 802.1Qbb-2011, IEEE Std 802.1Qaz-2011, IEEE Std 802.1Qbf-2011,IEEE Std 802.1Qbg-2012, IEEE Std 802.1aq-2012, IEEE Std 802.1Q-2012*, pp. 1–1782, Dec 2012.

[20] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE industrial electronics magazine*, vol. 11, no. 1, pp. 17–27, 2017.

[21] R. Santos, R. Marau, A. Vieira, P. Pedreiras, A. Oliveira, and L. Almeida, "A Synthesizable Ethernet Switch with Enhanced Real-Time Features," in *2009 35th Annual Conference of IEEE Industrial Electronics*, Nov 2009, pp. 2817–2824.