Universitat de les Illes Balears

TRABAJO DE FIN DE GRADO

Implementación y verificación de mecanismos para tolerar fallos en el canal de comunicaciones de manera dinámica en sistemas distribuidos empotrados de tiempo real críticos basados en Ethernet

Marc Costa Márquez

Grado de ingeniería en electrónica industrial y automática

Escuela politécnica superior

Año Académico 2020-21

Implementación y verificación de mecanismos para tolerar fallos en el canal de comunicaciones de manera dinámica en sistemas distribuidos empotrados de tiempo real críticos basados en Ethernet

Marc Costa Márquez

Trabajo de Fin de Grado

Escuela politécnica superior

Universidad de las Illes Balears

Año Académico 2020-21

Palabras clave del trabajo:

DFT4FTT,

Nombre Tutor/Tutora del Trabajo Alberto Ballesteros

Índice

Resumen

Desde el 1980 están apareciendo soluciones tecnológicas que han ido acercando a las redes de ordenadores al concepto de sistemas distribuidos. Desde los 80 que aparecen los primeros sistemas operativos distribuidos, así como las redes LAN, hasta a hoy en día, que una fuerte entrada en el mercado de los microprocesadores, permite conseguir software y hardware dedicado y distribuido a bajo coste haciendo que los sistemas distribuidos estén en sistemas de control de fábricas, comunicaciones multimedia, la industria o la conducción automática. Esta demanda tan diversa hace que algunas veces los requisitos de los sistemas sean más exigentes. En este trabajo nos centraremos en explicar las exigencias de la comunicación y fiabilidad más críticas dentro de estos sistemas, empezaremos explicando el paradigma FTT, fundamental para aquellos sistemas que quieran funcionar con tiempo real, hablaremos de la evolución de los protocolos más relevantes para nuestro trabajo hasta llegar a la infraestructura DFT4FTT, en la que nos centraremos para implementar nuestro mecanismo de sincronización (SECSM) así como en validar y evaluar el mecanismo para mejorar la tolerancia a fallos de la red.

Lista de figuras

Acrónimos

DES Distributed embeded sistems

FTT Flexible Time-Triggered

HaRTES Hard Real-Time Ethernet Switch

FTTRS Flexible Time-Triggered Replicated Star

DFT4FTT Dynamic Fault Tolerance for Flexible Time-Triggered

EC Elementary Cycle

TM Trigger Message

FTT-SE Flexible Time-Triggered Switch Ethernet

SECSM Slave Elementary Cycle Synchronization Mechanism

GPIO General Pin Input Output

1. Introducción

En esta sección empezaremos motivando el trabajo, así como contextualizando el área de estudio, enumeraremos los objetivos más específicos del trabajo y comentaremos las tareas realizadas para conseguir estos objetivos.as para conseguir estos objetivos.

1.1. Contexto y motivación del trabajo

Los sistemas de control empotrado son sistemas informáticos diseñados para controlar otro sistema diferente. Debido al bajo coste de los componentes electrónicos, hoy en día podemos encontrar sistemas empotrados por todas partes, desde usos industriales como en fábricas hasta usos particulares como en casas y coches. Estos sistemas suelen estar limitados por su tamaño o energía y debido a su naturaleza son sistemas de *tiempo real*, esto significa que han de reaccionar en un periodo de tiempo limitado. Si el sistema responde más tarde del plazo máximo establecido se considera que ha fallado. Dependiendo de las consecuencias de este retraso podemos hablar de: *tiempo real débil*, si las perdidas no son graves, como en la transmisión o recepción de un archivo multimedia; y *tiempo real fuerte*, si conlleva perdidas graves como en el control del tráfico aéreo o en un sistema médico donde se podría perder la vida de una persona.

Además de esto, si el sistema es crítico, se requiere que éste proporcione ciertas garantías de *fiabilidad*. Esto se puede lograr mediante tolerancia a fallos. Básicamente añadiendo redundancia para que, en caso de que ocurra un fallo, el sistema sea capaz de seguir proporcionando su servicio.

Con frecuencia estos sistemas suelen estar distribuidos, es decir, son sistemas empotrados y distribuidos (Distributed Embedded Systems, DESs). Un DES consta de varios nodos de computación interconectados mediante una red de comunicaciones.

Normalmente, las condiciones de operación de este tipo de sistemas se conocen de antemano. Cuando esto sucede, su configuración hardware y software (por ejemplo, el número de nodos, los programas que ejecutan, y los mensajes que se intercambian) puede ser estática (no cambia). Sin embargo, en los últimos años muchos DESs deben operar en entornos dinámicos, donde las condiciones de operación cambian de forma impredecible. Estos sistemas deben ser capaces de reconfigurarse de forma autónoma para *adaptarse* a los cambios del entorno.

El proyecto Dynamic Fault Tolerance for Flexible Time Triggered (DFT4FTT) tiene como objetivo proporcionar una infraestructura sobre la que construir DES con requisitos de tiempo real fuerte, alta fiabilidad y adaptabilidad. A nivel de las comunicaciones DFT4FTT utiliza Flexible Time Triggered (FTT). Este paradigma de comunicaciones permite a los nodos de un DES

intercambiarse mensajes con requisitos de tiempo real. Además, al ser flexible, permite que estos requisitos puedan ir cambiando mientras el sistema está operando. FTT se basa en un modelo de comunicaciones maestro/esclavo donde el *maestro* gestiona las comunicaciones de los *esclavos* (los nodos del DES). Para ello el maestro organiza las comunicaciones en rondas de tiempo de tamaño fijo llamadas Elementary Cycles (ECs) y al principio de cada EC el maestro envía un mensaje llamado Trigger Message (TM) que contiene la lista mensajes que los esclavos deben enviar durante ese EC.

El TM es un mensaje crítico ya que, si un fallo corrompe dicho mensaje, el sistema deja de responder durante un EC. Es por ello que el diseño de DFT4FTT contempla el uso de un mecanismo de tolerancia a fallos llamado Slave Elementary Cycle Synchronization Mechanism (SECSM) que se asegura que el TM se transmita a los esclavos de forma fiable. Este mecanismo se dividide en dos partes.

El presente trabajo tiene como objetivo la implementación de este mecanismo, la integración de dicha implementación dentro del prototipo en desarrollo de DFT4FTT, así como su verificación y evaluación.

1.2. Objetivos específicos

En esta sección se listan y explican los objetivos específicos en los que se pueden dividir el objetivo principal de este trabajo.

Primero nos centraremos en familiarizarnos con la tecnología y el entorno de desarrollo.

Estudiar el paradigma FTT [1].

Estudiar el diseño de la transmisión fiable de TM.

Identificar y entender los ficheros y funciones de la implementación heredada que son relevantes para llevar a cabo las tareas del trabajo.

Implementar en el maestro la transmisión fiable de TM.

Modificar el TM para que contenga todos los datos necesarios.

Implementar la transmisión múltiple de TM, habilitando también la modificación del número de replicas dinámicamente.

Comprobar el correcto funcionamiento de la implementación.

Implementar en los esclavos la recepción fiable de los TM.

Determinar el inicio de la ventana síncrona a partir de los múltiples TM recibidos.

Comprobar el correcto funcionamiento de la implementación.

Evaluar la exactitud de la transmisión/recepción fiable de los TM.

Implementar una inyección de fallos controlada en el sistema.

Medir la sincronización entre los esclavos.

Interpretación de los datos de los experimentos.

2. Fundamentos y trabajo previo

En esta sección hablaremos sobre el trabajo previo realizado en el campo de las comunicaciones que es más relevante para el trabajo descrito en este documento. Primero explicaremos con más detalles el paradigma FTT así como sus implementaciones. En concreto, nos centraremos en dos implementaciones de FTT sobre switched Ethernet, una llamada Hard Real-Time Ethernet Switch (HaRTES) y su versión tolerante a fallos Flexible Time-Triggered Replicated Star (FTTRS). Después se explicará Dynamic Fault Tolerance for the Flexible Time-Triggered (DFT4FTT), que es la infraestructura sobre la cual se ha desarrollado este trabajo y que a su vez utiliza FTTRS como subsistema de comunicaciones.

2.1. Paradigma FTT y sus implementaciones

Flexible Time-Triggered (FTT) es un paradigma de comunicaciones que garantiza la comunicación de *tiempo real flexible* y *flexibilidad operacional* para DESs.

Por un lado, que FTT garantice comunicaciones de tiempo real, significa que asegura que los nodos del DES intercambian cada uno de sus mensajes dentro de un plazo temporal acotado y determinista. Que además sea flexible, significa que las comunicaciones soportan tráficos con diferentes requisitos de tiempo real: básicamente fuerte y débil. Por otro lado, la flexibilidad operacional significa que se pueden cambiar los requisitos de tiempo real y la planificación del tráfico de forma *online* (sin parar el sistema), para proporcionar una comunicación adecuada a medida que cambian los requisitos operacionales del sistema.

Este paradigma permite la transmisión híbrida de mensajes periódicos y aperiódicos. Los mensajes periódicos son aquellos que se envían constantemente cada cierto tiempo. Un ejemplo de mensaje periódico es aquel que transporta las lecturas de un sensor de temperatura. Por otro lado, la transmisión de los mensajes aperiódicos está activada por un evento y pueden ocurrir en cualquier momento como, por ejemplo, una alarma.

FTT se basa en un modelo de comunicación entre maestro/esclavo, donde un nodo privilegiado llamado maestro gestiona las comunicaciones entre el resto de los nodos, llamados esclavos. Para ello el maestro organiza la comunicación en fragmentos temporales llamados ciclos elementales, EC de sus siglas en inglés. Cada EC se divide en 3 ventanas como se muestra en la figura 1.

Fig. 1. composición de un EC.

Durante la primera ventana denominada Trigger Message window el maestro envía el Trigger Message (TM). El TM es un mensaje periódico que se envía al inicio de cada EC y cumple 2 funciones: que todos los esclavos sepan cuando es el inicio de la ventana síncrona para poder sincronizarse, y comunicar a cada esclavo el conjunto de mensajes que tienen que enviar en ese EC. Esta planificación de mensajes se hace teniendo en cuenta diferentes requisitos de tiempo real que tiene el sistema durante su funcionamiento y este es el principal mecanismo por el cual FTT proporciona flexibilidad. La segunda, es la ventana síncrona, donde se transmiten todos aquellos mensajes periódicos que han cumplido su periodo. Y la tercera, la ventana asíncrona, para los mensajes aperiódicos en los que su evento haya ocurrido.

Existen varias implementaciones del paradigma FTT. Algunos ejemplos son: FTT-CAN, sobre el protocolo de comunicaciones Controller Area Network; FTT-Ethernet sobre Ethernet con una topología en forma de bus; y FTT-SE sobre Ethernet con una topología en forma de estrella con un switch comercial.

En este trabajo es relevante la implementación de FTT Hard Real-Time Ethernet Switch (HaRTES), que utiliza Ethernet con una topología en estrella con un switch hecho a medida. Se considera la evolución de FTT-SE, ya que este último tiene ciertas limitaciones debidas al uso de un switch comercial [2]. El switch de HaRTES solventa las limitaciones de FTT-SE y dota al sistema de ventajas adicionales. Básicamente podemos mencionar las siguientes mejoras:

- En primer lugar, el switch de HaRTES incluye el FTT master. Este hecho permite aumentar la integridad del sistema, simplificar la gestión del tráfico asíncrono y disminuir la variabilidad temporal durante el envío de señales digitales conocido como *jitter*.
- En segundo lugar, el switch de HaRTES incluye y gestiona colas para mensajes con diferentes propiedades de tiempo real. En particular, esta característica permite conectar no sólo nodos FTT, sino nodos comerciales que envían tráfico sin garantías de tiempo real.
- Por último, el switch incluye un *Port Guardian* por puerto. Cada Port Guardian supervisa el tráfico que se recibe por su puerto, descartando los mensajes que sean erróneos desde el punto de vista temporal, por ejemplo, mensajes que un nodo FTT envíe más frecuentemente de lo permitido según la planificación de las comunicaciones.

Sin embargo, HaRTES no es adecuado para implementar la red de comunicaciones de un DES crítico. Esto es debido a que HaRTES es vulnerable frente a fallos permanentes que afecten al canal. El switch, que recordemos tiene dentro el maestro, representa un punto único de avería porque si éste falla todas las comunicaciones entre esclavos fallarán. Además, HaRTES también es vulnerable frente a fallos temporales que afecten al canal. Por ejemplo, un pico de radiación podría modificar el valor de los bits de un mensaje siendo transmitido, lo que provocaría que el receptor recibiría un mensaje corrupto.

Flexible Time-Triggered Replicated Star (FTTRS) toma como base HaRTES e incluye replicación tanto temporal como espacial para solventar los problemas de fiabilidad previamente descritos. Por un lado, la replicación espacial provee tolerancia a fallos permanentes y, tal y como se muestra en la figura 2, consiste en duplicar la red. Ahora la red consta de dos switches interconectados, cada uno con su respectivo maestro, y cada esclavo se conecta a los dos switches mediante enlaces dedicados. Por otro lado, la replicación temporal permite tolerar los fallos transitorios y consiste en retransmitir de manera proactiva los mensajes críticos. Es decir, que un mismo mensaje es enviado por el canal varias veces sin necesidad de que el nodo receptor lo pida. Todo esto dota a HaRTES de mayor fiabilidad, manteniendo las propiedades del paradigma FTT

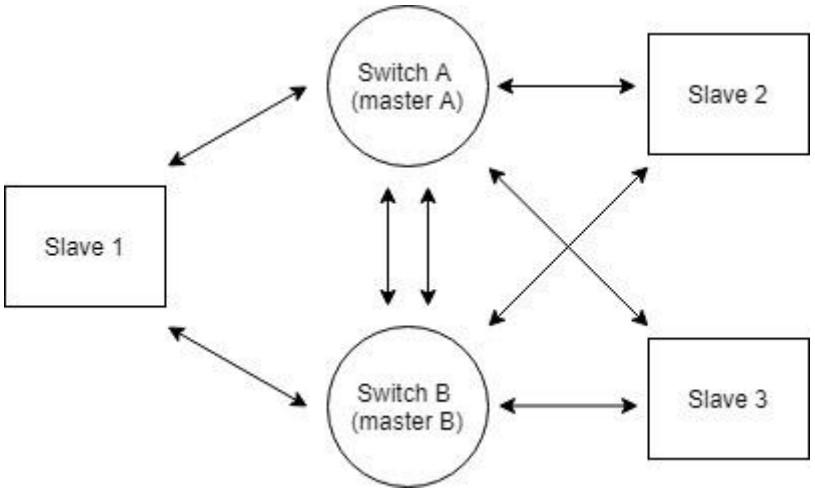

Fig. 2. Topología de FTTRS.

2.2. DFT4FTT

Dynamic Fault Tolerance for the Flexible Time-Triggered (DFT4FTT) es una infraestructura completa que facilita la implementación de DESs con requisitos de tiempo real, fiabilidad y adaptabilidad. Entendemos por infraestructura completa un conjunto de componentes hardware y software (la

arquitectura) junto con los mecanismos de tolerancia a fallos necesarios a nivel de nodo y de red.

A nivel de nodo DFT4FTT proporciona una alta fiabilidad mediante la replicación activa de los componentes computacionales que se ejecutan, las tareas. En concreto, cada tarea critica se ejecuta en paralelo en varios nodos del sistema. A nivel de red DFT4FTT utiliza FTTRS que, como hemos comentado anteriormente, proporciona los mecanismos de tiempo real y tolerancia fallos necesarios en las comunicaciones.

Una de las características más importantes de DFT4FTT es su adaptabilidad. Es decir, su capacidad para reconfigurarse durante la operación del sistema para hacer frente a cambios en los requisitos operacionales del sistema, el entorno o en el propio sistema derivados de averías. Una reconfiguración consiste en modificar la asignación de tareas a los nodos, así como sus comunicaciones asociadas. Esto permite: arrancar nuevas tareas que puedan ser necesarias, dar de baja tareas que ya no son necesarias o realojar tareas de un nodo a otro, por ejemplo, si el primer nodo sufre una avería.

A nivel de arquitectura, cómo se puede ver en la figura 3, DFT4FTT hereda la topología de FTTRS. Sin embargo, dado que ahora también no solo se atacan las comunicaciones si no también las tareas la nomenclatura cambia. Por un lado, los maestros pasan a denominarse Node Managers (NMs). Éstos se encargan de gestionar tanto la ejecución de las tareas como la transmisión de sus mensajes asociados. Además, también son los encargados de llevar a cabo las reconfiguraciones. Es decir, de monitorizar el entorno y el propio sistema y, en caso de que haya un cambio significativo, encontrar y aplicar una nueva distribución de tareas que permita al sistema seguir operando correctamente. Por otro lado, los esclavos pasan a llamarse Computational Nodes (CNs) ya que en DFT4FTT éstos se encargan de ejecutar las tareas indicadas por los NMs.

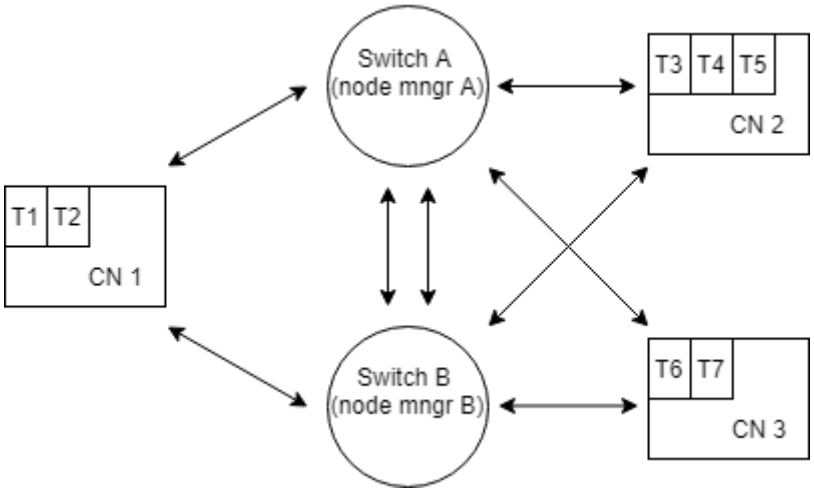

Fig. 3. Topología de DFT4FTT.

Como se ha comentado anteriormente, para conseguir una comunicación fiable FTTRS, aparte de la replicación espacial, utiliza replicación temporal. Más concretamente, para tolerar los fallos transitorios que afectan al canal, los mensajes críticos se retransmiten de manera proactiva. Esto es especialmente importante para el TM, ya que es un mensaje indispensable para el correcto funcionamiento del sistema. Sin embargo, tal y como se ha comentado en la Sec. 2.1, el TM no solo transporta datos, sino que también marca el inicio del EC. Por lo tanto, si nos limitásemos a transmitir varias copias del TM los CNs, no se sincronizarían con los NMs de manera correcta. Como se comentará en la Sec. 2.2.1, Es necesario replicar el TM de forma astuta e implementar en los esclavos un mecanismo de sincronización que tenga en cuenta la recepción de múltiples copias del TM.

Para conseguir una transmisión fiable del TM, DFT4FTT hereda de FTTRS el Slave Elementary Cycle Synchronization Mechanism (SECSM) o mecanismo de sincronización del EC de los esclavos. Este trabajo se centra en la implementación, integración dentro del prototipo de DFT4FTT en desarrollo, así como en la verificación y evaluación de dicho mecanismo.

2.2.1 Mecanismo de sincronización del EC en el esclavo (SECSM)

En esta sección se explica con detalle por qué es necesario un mecanismo de tolerancia a fallos específico para la transmisión del TM y en qué consiste. Es importante mencionar que, dado que se trata de un mecanismo diseñado en el contexto de FTTRS, se utilizarán los términos *maestro* y *esclavo*. Sin embargo, todo lo que aquí se explica es extrapolable a los NMs así como a los CNs. Además, para simplificar las explicaciones, y sin que esto afecte a la efectividad del mecanismo desde el punto de vista de la sincronización, asumiremos un solo maestro.

Como se ha introducido anteriormente, el canal de comunicaciones puede sufrir fallos transitorios. Si un fallo transitorio afecta a un TM, los esclavos serían incapaces de reaccionar durante el EC asociado a dicho TM. Es decir, el sistema permanecería inactivo durante un corto espacio de tiempo. Sin embargo, esta inactividad temporal podría provocar problemas de consistencias en el estado de los esclavos, lo que podría desembocar en una avería en la totalidad del sistema. Esto ocurre porque el TM es un punto único de avería.

Una manera de solventar este problema es forzar al maestro a que retransmita varias copias del TM de manera proactiva. De esta forma, si ocurre un fallo transitorio de una duración no muy larga, alguna de las réplicas podrá llegar a los esclavos. Sin embargo, esto presenta un nuevo

problema. Ahora los esclavos recibirán varias copias del TM. A diferencia de los mensajes de datos, esto es un problema ya que el TM marca el inicio del EC y, por lo tanto, el momento en el que los esclavos reciben el TM es muy relevante. Así pues, este mecanismo se puede considerar como 2 mecanismos que trabajan en conjunto, el envío proactivo del TM por parte del maestro y la recepción y sincronización de las réplicas gracias a las réplicas del TM por parte de los esclavos.

En lo que respecta a la transmisión proactiva del TM, el maestro envía *k* réplicas del TM a todos los esclavos durante la ventana del TM. El valor adecuado de *k* depende de la ratio de fallo del canal de cada sistema, de forma que cada esclavo recibirá al menos uno y un máximo de *k* TM. Además, en DFT4FTT, el valor de *k* puede cambiar si las condiciones del entorno, el estado del sistema o sus requisitos cambian. La decisión de cambiar el valor de *k*, y de otros parámetros del sistema, depende de mecanismos de reconfiguración más avanzados que se implementan a más alto nivel y que no forman parte de este trabajo. Sin embargo, como se explicará más adelante la implementación se llevará a cabo de tal manera que el valor de *k* podrá cambiar mientras el sistema está operando sin perjuicio de la implementación de este mecanismo.

El nivel de replicación del TM es uno de los múltiples aspectos que son reconfigurables en DFT4FTT. Esto permite utilizar los recursos de red de manera más eficiente y eficaz. En el caso en el que el contexto operacional del sistema no sea muy adverso, el valor de *k* puede disminuir, lo que deja libre parte del ancho de banda. Por el contrario, si el contexto operacional es adverso, el valor de *k* puede aumentar para así incrementar la capacidad de tolerar fallos.

El periodo de transmisión de cada uno de estos TM es un tiempo Tau, un tiempo constante y predefinido. Cada uno de estos TM contiene toda la información necesaria para que el esclavo pueda calcular el momento de inicio de la ventana síncrona.

En la figura 4 se muestra una ventana temporal de la recepción de TM desde el punto de vista del esclavo del TM (m_i) en un EC (c) replicado k veces para 2 nodos S1 y S2 donde $1 \leq i \leq k$. Así pues, a_{s2} (3, c) representaría el momento de llegada al esclavo 2 (a_{s2}), de la tercera réplica del TM (3) del EC c (c).

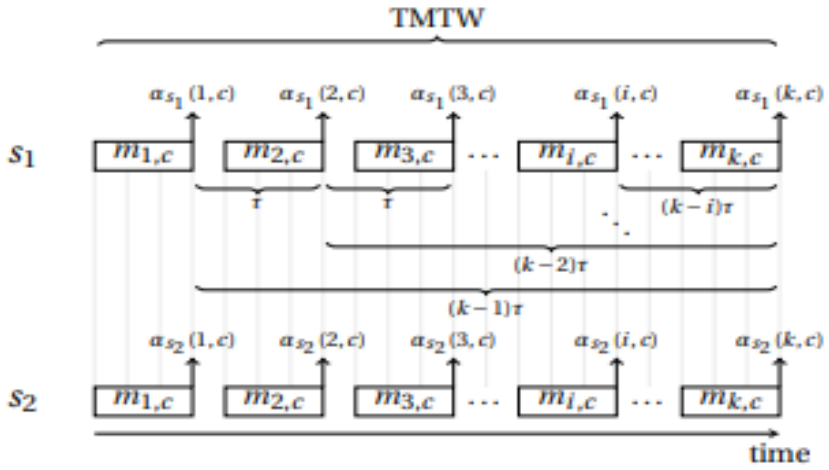

Fig. 4. Ventana del TM, desde el punto de vista de los esclavos. Tal y como aparece en [3].

Para que este mecanismo funcione correctamente es necesario encapsular en el TM información adicional. Esta información será luego usada por los esclavos para llevar a cabo la sincronización. En la figura 5 se muestran en azul los campos de datos ya presentes en el TM y en blanco los nuevos campos, donde:

- *k* es el número de réplicas del TM.
- *TM seq. num.* es el número de réplica del TM y su valor va de 1 a *k*. A partir de ahora se utilizará la letra *i* para referirnos a ella.
- τ es tau, es decir, el tiempo entre el inicio de un TM y el inicio de la siguiente réplica del mismo TM.

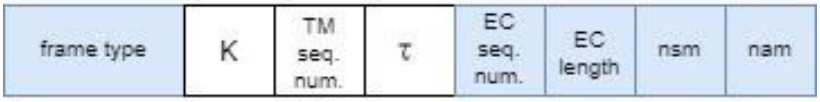

Fig. 5. Trama de datos de un EC.

Por su parte, los esclavos deben gestionar la recepción de las múltiples réplicas del TM a fin de determinar el inicio de la ventana síncrona de forma correcta. Es decir que los esclavos tienen que ser capaces de determinar el inicio de la ventana asíncrona independientemente del TM que reciban, ya que, por dificultades en el entorno, algunos TM se pueden perder de forma que cada esclavo puede recibir un subconjunto diferente de TM, Como se verá en más detalle en la Sec. 5, el primer TM que reciba un esclavo podría ser el primero en enviarse, pero otro esclavo podría recibir como primer TM el tercero en enviarse debido a un fallo temporal en el canal.

Cada vez que un esclavo recibe un TM, éste registra el instante temporal en el que dicho TM ha sido capturado (a_s en la figura 4). Además, el esclavo también extrae *k*, *i* y tau. Toda esta información es necesaria para que el esclavo pueda calcular el inicio de la ventana síncrona. Una vez calculado, se

duerme omitiendo el resto de las réplicas, y se despierta al inicio de la ventana síncrona preparado para transmitir, en caso de que haya mensajes periódicos pendientes de ser transmitidos.

Tal y como se muestra en la figura 4, para calcular el instante de activación de la ventana síncrona, el esclavo determina cuántas replicas faltan por recibir, es decir, resta *i* de *k*. Si esto se multiplica por tau, se puede determinar el tiempo que falta para que acabe la ventana del TM y, por lo tanto, el tiempo que falta para el inicio de la ventana síncrona. Además, si a eso sumamos el instante de tiempo en el que el mensaje fue capturado obtenemos el instante de activación de la ventana síncrona en términos absolutos: $a_s + (k - i) * tau.$

3. Diseño

En este apartado se hará una descripción de la arquitectura lógica de las partes más relevantes de DFT4FTT para este trabajo. Se identificarán qué partes de la arquitectura intervienen en los mecanismos implementados en este TFG, así como los cambios que serán necesarios para conseguir implementar dichos mecanismos. Es importante remarcar que dicha arquitectura ya existía y, por lo tanto, no es una contribución propia.

Es importante remarcar que, tal y como se ha explicado en la Sec. 2, el SECSM fue concebido en el contexto de FTTRS. Es por ello que aquí también se utilizarán los términos *maestro* y *esclavo*, pero éstos son intercambiables por NM y CN, respectivamente, en el contexto de DFT4FTT. Además, de manera similar a lo explicado en la Sec. 2, el hecho de simplificar la configuración del sistema de tal manera que solo exista un maestro no afecta a la explicación ni funcionamiento de este mecanismo. Esto se debe a que la redundancia de los maestros es invisible para los esclavos. Por lo tanto, en esta sección y en las siguientes consideraremos un sistema con la configuración mostrada en la figura 6. En concreto todo el desarrollo se llevará a cabo sobre un prototipo que hemos denominado *single master prototype* y que explicamos con más detalle a continuación.

3.1. Configuración del prototipo con un solo maestro

En la figura 6 se muestra la configuración de sistema con un solo maestro. Esta figura es la que utilizaremos para explicar el diseño de la transmisión proactiva del maestro y la sincronización a partir del TM en el esclavo.

A la izquierda se muestra el maestro y a la derecha el esclavo, en este dibujo se muestran los elementos más importantes que interactúan durante el mecanismo de sincronización. Estos elementos pueden ejecutar o almacenar datos, y las flechas entre ellos representan una comunicación entre ambos elementos donde el envío de la información va en sentido de la flecha. Los elementos dibujados como un cilindro representan bases de datos o elementos que solo guardan información y proveen de esta a los elementos que se la solicitan. Las flechas fuera de los recuadros representan el enlace de comunicación por donde se envían las réplicas del TM entre el maestro y el esclavo, así como los mensajes de datos de los esclavos cuando llega la ventana síncrona.

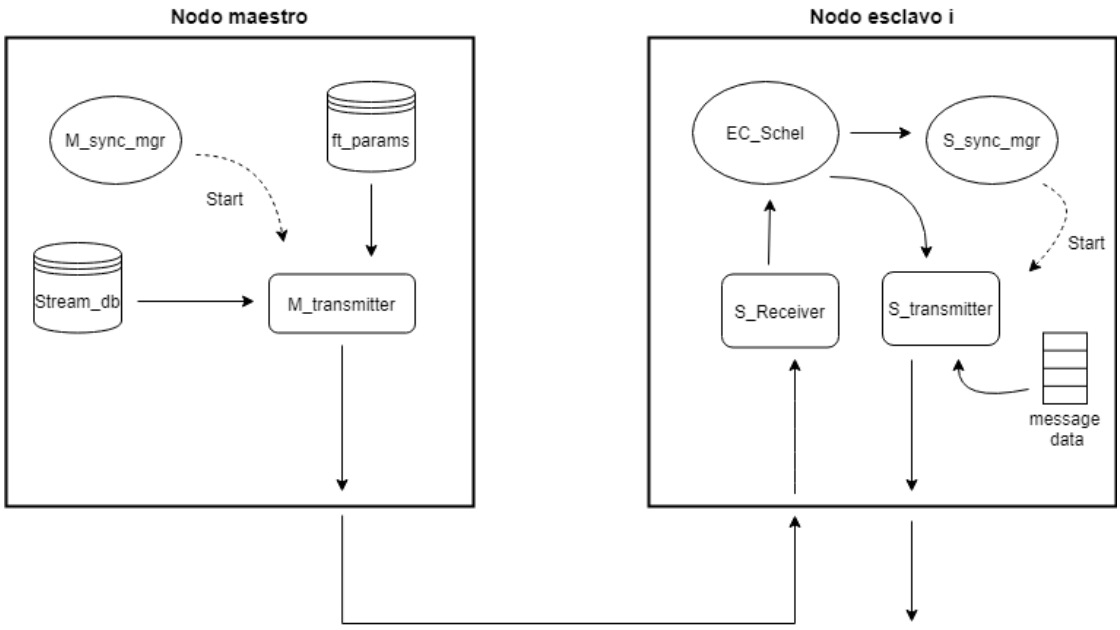

Fig. 6. Single master prototype.

3.2. Retransmisión proactiva del TM en el switch/master

El `M_sync_mgr`, en el maestro, es un componente que se encarga de disparar la ejecución de otros componentes cuando son necesarios. En concreto, este componente despierta al `M_transmitter` al principio de cada EC. Nótese que la flecha discontinua del dibujo implica una señal de este tipo y, a diferencia de las flechas continuas, no hay un flujo de información.

El `M_transmiter` se encarga, primero, de construir el TM. Sin embargo, para ello necesita saber cuál es el EC actual y la planificación para dicho EC, es decir, la lista de mensajes que los esclavos deben transmitir durante el EC actual. Por lo tanto, este componente consulta la `Stream_db`, que es un almacén donde se guarda la lista de mensajes periódicos que los esclavos se intercambian. En el contexto de DFT4FTT lo llamamos *stream* ya que esto suele ser un conjunto de mensajes que se han de enviar.

Una vez el `M_transmiter` ha construido el TM, se encarga de transmitirlo *k* veces. Este valor se obtiene de `ft_params`, un elemento que almacena los parámetros operacionales referentes a la tolerancia a fallos. Tal y como se ha comentado en la Sec. 2.2.1, en DFT4FTT, el valor de *k* puede ir cambiando. La decisión de cambiar este valor, y otros parámetros del sistema, la toma un mecanismo de reconfiguración más alto nivel que no forma parte de este trabajo. En el caso de que fuese necesario modificar el valor de *k*, este mecanismo simplemente accedería a `ft_params` y, en el siguiente EC, el `M_transmiter` utilizaría el nuevo valor. Por lo tanto, el cambio dinámico del número de réplicas del TM no altera el correcto funcionamiento del `M_transmiter`.

3.3. Recepción/sincronización respecto al TM en el esclavo

Cada uno de los esclavos recibe los TMs gracias al `S_Receiver`. Este componente primero, extrae del TM la planificación del EC actual. Esta información es enviada al `EC_Schel`, el cual se encarga de almacenar esta información hasta que es necesaria, es decir, al principio de la ventana síncrona. Además de esto, el `S_Receiver` también registra el momento de recepción de cada TM y determina el instante de activación de la venta síncrona, tal y como se describe en la Sec. 2.2.1. Esta información es enviada al `S_sync_mgr`. De manera similar al `M_sync_mgr`, el `S_sync_mgr` se encarga de despertar a otros componentes en el instante de tiempo que son necesarios, pero esta vez en el esclavo.

Finalmente, cuando el `S_sync_mgr` esclavo calcula que la ventana empezará, despierta al `S_trasmitter`, el cual consulta el `EC_schel` para saber qué mensajes tiene que enviar. Con esta información extrae dichos mensajes del buffer `message data`.

4. Implementación

En esta sección abordaremos los detalles de implementación de este trabajo. El objetivo es conseguir un prototipo sencillo de DFT4FTT que incluya el SECSM. Primero se listarán y explicarán las herramientas y tecnologías que han sido necesarias durante el desarrollo. Segundo se mostrará cómo arrancábamos todos los componentes del sistema en el entorno virtual que utilizamos durante de la fase de desarrollo. Finalmente se mencionarán los ficheros del proyecto que son relevantes, así como las modificaciones llevadas a cabo en cada uno de ellos.

4.1. Herramientas y tecnologías

Como se explicará en detalle más adelante, la configuración del sistema durante la fase de desarrollo consistió en un entorno totalmente virtual ejecutado en una máquina virtual. En concreto se utilizó el programa Oracle VM VirtualBox para crear una máquina virtual Linux Ubuntu 16.04. Esta máquina virtual sirvió, por un lado, para la programación y compilación del código. El código inicial de DFT4FTT estaba desarrollado en lenguaje C. Es por ello que las contribuciones al código en de este trabajo también fueran en este lenguaje de programación.

Por otro lado, esta máquina virtual también ha servido para ejecutar un sistema DFT4FTT sencillo donde verificar la implementación durante la fase de desarrollo. Más concretamente, una vez compilado el código se arrancan tres procesos, uno que se corresponde al maestro, y dos a los esclavos. Además, para comunicar estos procesos se usaron interfaces de red virtuales mediante el software VDEv2: Virtual Distributed Ethernet [4].

Para poder monitorizar el tráfico de red entre el maestro y los esclavos, y así poder verificar el correcto funcionamiento del software durante la fase de desarrollo, utilizamos Wireshark. Este programa permite capturar y analizar los mensajes enviados en una red.

En las pruebas finales, una vez se empezamos a trabajar con un sistema no virtualizado, cada nodo fue construido usando hardware específico para dispositivos empotrados. Específicamente con barebones Jetway JBC373F38-525-B cada uno con un procesador Intel Atom y cuatro interfaces de interfaz de red Ethernet estándar (de las que solo usamos 2).

Como se explicará en el Sec. 6, parte de la verificación del sistema consistió en hacer mediciones del nivel de sincronización. En este sentido, para registrar las marcas temporales de los eventos de sincronización más relevantes utilizamos hardware adicional. Para registrar los eventos relevantes de los esclavos usamos un Arduino Mega 2560 y para registrar los eventos relevantes relacionados con la transmisión de mensajes del maestro utilizamos un netANALYZER. El primero es una placa que incluye un

microcontrolador que podemos programar. En concreto implementamos un programa para registrar, mediante General Purpose Input Output Pins (GPIOs), el momento en cada esclavo consideraba que la ventana síncrona había empezado. El netANALYZER es un dispositivo físico dedicado muy preciso (10 ns de error) que permite capturar y mostrar el tráfico de red que se envía y recibe a través de 2 interfaces de red diferentes. Además permite exportar los datos en formato Wireshark. Este dispositivo se usó registrar los tiempos de las transmisiones de las réplicas de los TM.

Finalmente, para poder procesar el gran volumen de datos extraído de los experimentos, y para mostrar los resultados en forma de gráficas, se usaron scripts programados en el lenguaje Python. Python es un lenguaje interpretado, que no requiere compilación y es multiplataforma. Se trata de un lenguaje de alto nivel que hace fácil trabajar con estructuras de datos complejas como clases o tablas de *hash*.

4.2. Prototipo virtualizado

Como se ha mencionado en el apartado anterior, para este prototipo virtual se utilizó una máquina virtual con Linux. Normalmente, cuando se desarrolla para un DES se programa en una maquina y luego el código se inserta en un sistema hardware para llevar a cabo las pruebas. En este caso se virtualizará el sistema sobre la misma máquina utilizada para lidiar más fácilmente con las primeras fases de desarrollo y verificación. A continuación, se describe cómo se arranca el sistema en este prototipo virtual.

Primero se arranca el switch virtual con tres interfaces de red, una para cada nodo que virtualizaremos. Para simular un maestro empotrado en un switch, arrancaremos el proceso de un maestro y lo conectaremos al switch virtual a través una de sus interfaces, haciendo creer al maestro que está conectado a una red real. Finalmente, conectaremos a las dos interfaces restantes del switch los dos procesos de nodos esclavos.

Todos los comandos se ejecutan como superusuario para tener los permisos necesarios, primero lanzamos el switch virtual con tres interfaces llamadas mst, slv1 y slv2, el -d es para ejecutarlo en segundo plano.

```
sudo vde_switch -t mst -t slv1 -t slv 2 –d
```

Luego creamos los nodos esclavos número 1 y 2 y los conectamos a una interfaz diferente cada uno.

```
sudo ./node -n 1 -i slv1
```

```
sudo ./node -n 2 -i slv2
```

Y finalmente ya podemos crear un master diciéndole el tamaño que tendrá cada EC, la interfaz a la que ha de conectarse y el -S le indica que ha de

funcionar en modo “single master prototype” para que sepa que él será el único maestro de la red.

```
sudo ./node_mgr -e 500 -i mst -S
```

4.3. Principales modificaciones del código

El punto de partida es el código ya implementado del proyecto DFT4FTT, las primeras modificaciones fueron dotar al maestro del mecanismo de transmisión proactiva del TM, una vez estas funcionaran correctamente, implementar en los nodos esclavos el mecanismo de recepción/sincronización.

Las modificaciones, siguiendo el orden utilizado en la sección 3, se comentarán primero las pertenecientes a los ficheros usados por el maestro y después a los del esclavo.

Por parte del nodo maestro son:

En `M_Transmitter.c` se implementa el mecanismo de transmisión proactiva durante cada EC. Extrae el valor de *k* de `ft_params` y se envía el TM en bucle K veces.

Se crea el fichero `ft_params.c` y `ft_params.h` donde se define el valor de Tau en microsegundos (`RTM_TAU_US`), un método para adquirir el valor actual de K (`FT_PARAMS_get_k(void)`) y otro para establecer un nuevo valor de K (F`T_PARAMS_set_k(uint8_t k)`).

En `ftt_msg.h` se añade en el encapsulado del TM la cabecera de DFT4FTT que contiene en cada TM el valor de K, el número de repetición de TM y el valor de tau que como se ha mencionado en secciones anteriores son los datos necesarios para que el esclavo calcule el inicio de la ventana síncrona.

En `ftt_params.h` se renombra la variable donde se almacena el valor tau para especificar las unidades de tiempo.

Por parte de los nodos esclavos son:

En `S_Receiver.c` se ha implementado que guarde la marca temporal de cuando llega el TM, a partir de los datos de la cabecera DFT4FTT calcula el tiempo que falta con la formula explicada en la Sec. 2.2.1 y suma ese tiempo a la marca temporal para saber el inicio de la ventana síncrona.

En `cn_sync_mgr.c` y en `cn_sync_mgr.h` se añade la marca temporal para que despierte al `S_transmitter`

Adjunto a este documento se incluye todo el código modificado durante este trabajo.

5. Experimentación

En esta sección se describen las diferentes pruebas que se han llevado a cabo para verificar el correcto funcionamiento e integración del SECSM. Estas pruebas realizadas pueden separarse en 2 según su carácter y naturaleza. Las primeras son de carácter discreto, pueden funcionar o no, y de naturaleza virtual, mientras que las segundas consisten en validar el grado de sincronización que se puede conseguir en un entorno físico real.

Primero se harán una serie de pruebas virtualizadas ya que esto facilitara la puesta en marcha de las pruebas, así como la evaluación de los resultados. Aunque los resultados de estas pruebas no serán tan precisos, en esta primera parte lo que nos interesa es poder manejar y analizar los datos con más facilidad y tener el sistema virtualizado permite fácilmente añadir nuevos nodos al sistema para hacer pruebas incrementales.

Una vez acabadas estas pruebas virtualizadas se hará un experimento de inyección de errores en el canal, primero en un sistema completo virtualizado, para comprobar que la inyección de fallos y la respuesta del sistema a estos fallos es la deseada, y después en hardware físico para realizar los últimos test con la mayor precisión posible ya que de esas pruebas es de donde se evaluará que tan adecuado es este mecanismo.

Es importante remarcar que la implementación heredada de DFT4FTT se centra principalmente en la fiabilidad del sistema, por eso la implementación heredada no está pensada para funcionar en tiempo real. Sin embargo, este trabajo se centra en la sincronización. Es por ello que, como veremos en esta sección, algunos valores obtenidos no son tan buenos como en prototipo específicamente desarrollado para trabajar en un entorno con restricciones de tiempo real.

5.1. Pruebas durante el desarrollo en plataforma virtual

Las pruebas que se van a realizar en este primer apartado son una serie de pruebas incrementales, con el objetivo de probar de forma individual los mecanismos internos del mecanismo de sincronización dinámico tolerante a fallos de DFT4FTT.

Empezando por la virtualización de un nodo maestro para realizar la primera prueba que consistirá en comprobar si la transmisión proactiva del master funciona correctamente de forma dinámica, para ello se virtualizará un nodo maestro y con el Wireshark se observará la red para comprobar que se envían K copias del TM en cada EC comprobando que funciona tanto para k = 2, k = 3 o k = 4, más allá de 4, valores más grandes de k aportan una mejora significativa a la fiabilidad como se muestra en **[2]**.

Una vez comprobado que la transmisión funciona correctamente, se virtualizarán un switch virtual junto a un nodo maestro y dos esclavos

interconectados para formar una red con topología single master prototype como se muestra en la figura 7.

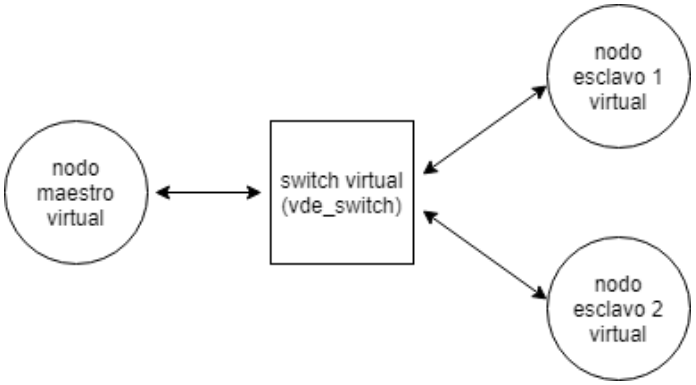

Fig. 7. Entorno de pruebas virtual.

Para la segunda prueba se comprobará que cada esclavo reciba e interpreta correctamente cada uno de los TM. Y a partir de ellos, sean capaces de determinar correctamente en inicio de la ventana síncrona a partir de la formula explicada en la Sec. 2.2.1, de nuevo se comprobará con ayuda del Wireshark que la transmisión ocurre al inicio de la ventana síncrona y se usaran valores de k = 2, 3 y 4.

5.2. Pruebas para evaluar el conjunto en plataforma virtual

Sobre la topología mencionada en la prueba anterior se incluirán unas modificaciones en el código del S_Receiver para gestionar una inyección de fallos controlada en la recepción de los TM haciendo que en cada EC uno o varios TM no sean recibidos por el esclavo.

En esta prueba se quiere comprobar que los esclavos se sincronizan en todos los escenarios posibles donde haya perdida de algún TM, pero asegurando que al menos 1 de las K réplicas del TM sea recibida por cada esclavo. Para conseguir el número de escenarios posibles dependerá del número de réplicas del TM, para estas pruebas se usarán valores de K = 2, 3 y 4. Cada uno de estos mensajes puede o no llegar correctamente, de forma que, para el caso de estudio, cada mensaje tiene dos posibles valores.

Las combinaciones de k elementos que pueden tener 2 valores cada uno, es de 2^k combinaciones diferentes, pero entre todas estas combinaciones está el caso en que fallen los k mensajes, de modo que hay que restar 1 al total de combinaciones. Para cada nodo entonces nos sale un total de (2^k) - 1 combinaciones, si tenemos en cuenta que esto ocurre para cada nodo y puede ser que en cada nodo se dé una configuración de mensajes diferentes, el total de escenarios a estudiar será de [(2^k)-1] ^ (número de nodos esclavos). Para poner un ejemplo de esto, utilizando la topología de la prueba anterior donde teníamos 2 nodos esclavos y con un valor de k = 4 el total de escenarios a estudiar será de [(2^4)-1] ^ (2) = 225. En la figura 8 se muestra

uno de los 225 casos donde las cajitas encima de los buses de comunicación muestran las réplicas de los TM.

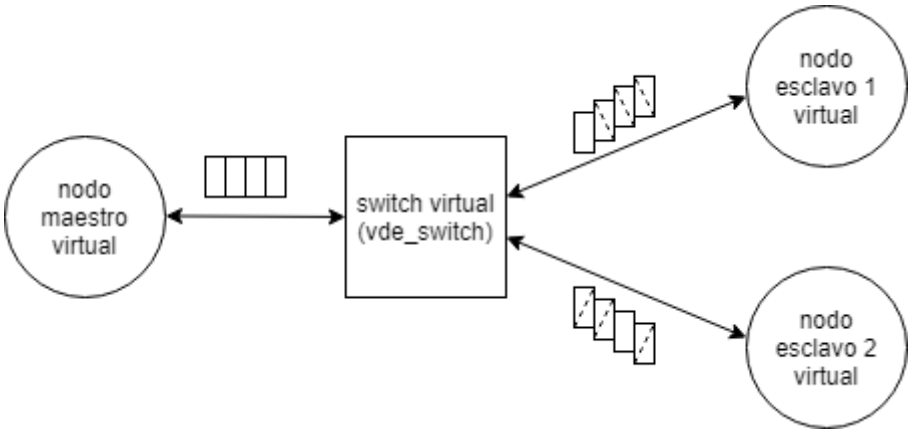

Fig. 8. Escenario 00101000.

Las cajitas rayadas representan TM que no han llegado correctamente al nodo esclavo, en el caso del esclavo 1 solo recibe correctamente el ultimo TM, mientras que el esclavo 2 solo recibe correctamente el segundo TM.

Para conseguir eso, usaremos unas máscaras de (número de nodos esclavos) * k bits, los 0 representarán que el mensaje no llega correctamente y los 1 que sí, estas mascaras se irán modificando de forma ordenada para contemplar todos los casos, en el ejemplo dado, los 4 bits más significativos corresponderían al esclavo 2 y los 4 bits menos significativos al esclavo 1, así pues, la máscara del ejemplo seria: 0010 1000.

Esta prueba se ha realizado con k = 2 y sus 9 escenarios, k = 3 con sus 49 escenarios y k = 4 con los 225 escenarios, cada uno de ellos con una longitud de EC de 10 milisegundos. Y para cada escenario el tamaño de la Tau anteriormente mencionada en el apartado 3.2.1 es de 400 microsegundos, el tamaño de la ventana de sincronización para k = 2 debería ser de 800 microsegundos, para k = 3 de 1,2 milisegundos y para k = 4 de 1,6 milisegundos.

5.3. Pruebas para evaluar el conjunto en plataforma real

Dado que este nuevo mecanismo de tolerancia a fallos puede afectar a la sincronización, con la siguiente prueba se quiere comprobar que manteniendo dicha tolerancia a fallos la sincronización siga siendo adecuada. Poniendo especial atención en mantener la sincronización entre los esclavos por parte de la recepción de mensajes, y que el tamaño de Tau y de un EC no se vean afectados por

En esta prueba, el código del sistema anterior se ha ejecutado en una plataforma física real para evaluar el mecanismo de sincronización. Al ser hardware real y ser un prototipo más cercano a un sistema real, los resultados serán más exactos y el tiempo de respuesta será mejor que en un entorno simulado.

Para llevar a cabo esta evaluación se tendrán que medir con precisión los tiempos de inicio del EC y del inicio en cada esclavo de la ventana síncrona. El sistema que se quiere montar, como se muestra en la figura 9, contiene un maestro y dos esclavos interconectados entre ellos que pasan por un netANALIZER que solo observara la comunicación. En cada uno de los nodos se ha conectado la placa Arduino con el fin de medir la sincronización entre los esclavos.

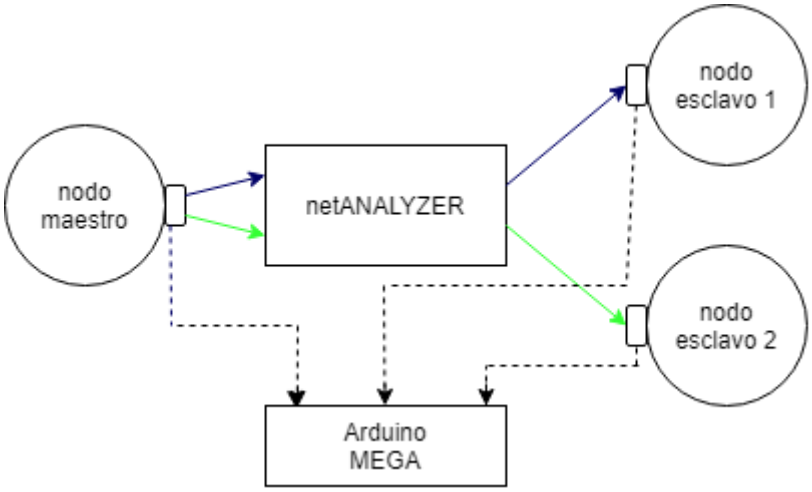

Fig. 9. Sistema de pruebas físico.

Para montar este sistema, primero se abrirán los barebones para enganchar en uno de los GPIO un cable que poder conectar a la placa Arduino, estos cables están representados en la imagen por las líneas discontinuas.

El uso de este Arduino es importante a la hora de medir los tiempos, ya que hay que asegurar que todas las marcas temporales estén cogidas respecto al mismo reloj, ya que lo que nos interesa es el diferencial de tiempo entre las señales, y no sus valores absolutos. Además, el único trabajo adicional que ha de realizar cada nodo para llevar a cabo la medición es modificar un bit que es una acción rápida y constante.

Este Arduino contiene un código para medir esos diferenciales entre las marcas de tiempo. Al inicio de cada EC, el maestro pone su GPIO a 1. De esta manera el Arduino registra el momento de inicio. Cuando un esclavo se despierta al inicio de la ventana síncrona también pone a 1 su GPIO y el Arduino guarda la marca de tiempo asociada a este evento.

Una vez se tienen todas las marcas de tiempo se descargan del Arduino y del netANALYZER. Los datos serán guardados en un fichero txt y de ahí serán consumidos por un código en Python para poder visualizar y evaluar estos datos.

A lo largo de este experimento se han adquirido muchos datos, y vamos a comentar los más relevantes. Para medir la sincronización entre los esclavos se realizaron 1000 experimentos de cada uno de los 225. En cada una de

estas muestras se obtuvo con el Arduino el momento en que cada esclavo se despertaba al inicio de la ventana síncrona, calculamos el valor absoluto entre la diferencia de las marcas temporales de cada uno para ver que desviación temporal había entre los 2 nodos en cada EC.

De las 225000 muestras 224778 tuvieron una diferencia entre el inicio de la ventana síncrona de un nodo y de otro menor a 100 microsegundos que se corresponde al 99,901% de los datos, que son los que se muestran en la figura 10. También se contabilizo en cada una de estas muestras cuantas veces se despertaba primero cada uno, pero los datos eran muy similares en todas las pruebas y no mostraban una inclinación clara a que uno de ellos sea más propenso a despertarse primero.

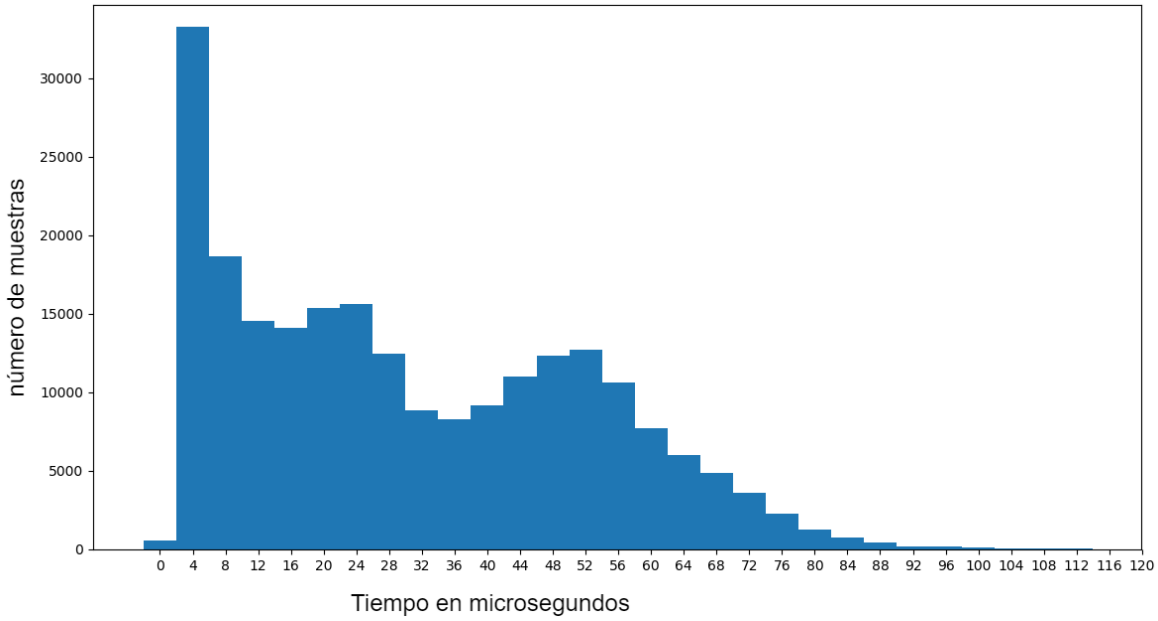

Fig. 10. Offset de sincronización.

Viendo estos resultados, que haya varios valores atípicos, de los cuales los más grandes alcanzan valores de casi 2 y 5 milisegundos, se debe a que el sistema operativo utilizado en este experimento no es de tiempo real.

Otro punto importante es comprobar en la transmisión que los valores de tau, así como la duración de un EC se ha mantenido, aprovechando que el netANALYZER media cada uno de los TM, hemos podido calcular el valor del EC midiendo la distancia que hay entre los primero TM de cada uno de los EC. Y para calcular los valores de Tau hemos mirado la diferencia entre el momento de envía

```
Variabilidad de la duración del EC
Valor máximo: 0.010159510000001148
Valor mínimo: 0.009855270000002747
Valor medio: 0.009999984370291747

Variabilidad de la TAU
Valor máximo: 0.0011259899999913614
Valor mínimo: 0.0003064800000061041
Valor medio: 0.0004210323642166941
```

Fig. 11. Resultados estadísticos.

Así como los valores del EC se mantienen muy estable alrededor de los 10 milisegundos que queríamos, en el caso de Tau se muestra un valor máximo bastante elevado, de las 225000 pruebas realizadas, cada una de ellas tenía 3 valores de Tau, nos da un total de 675000 pruebas y solo 8 de ellas tenían un valor de Tau superior a los 550 microsegundos esto supone que el 99,999% de los datos son consistentes, y estos valores atípicos, así como esta baja precisión se debe a estar utilizando un sistema operativo que no es de tiempo real.

Este último experimento también se ha llevado a cabo con k=2, k=3 y k=4, pero las imágenes mostradas en las figuras anteriores han sido todas de k=4 ya que, al tener un mayor volumen de datos, era un conjunto más representativo de la realidad, pero no ha habido ninguna discrepancia entre los datos para diversos valores de k.

6. Conclusiones

En esta sección primero presentaremos un resumen del trabajo descrito en este documento, así como las conclusiones a las que se han llegado a lo largo del mismo. Después hablaremos sobre el trabajo a futuro que se podría llevar a cabo y las consideraciones sobre el proceso de aprendizaje del estudiante. Finalmente, se mencionará de qué manera el trabajo presentado en este documento ha servido para escribir un artículo de congreso.

6.1 Resumen

Este documento presenta la implementación de un mecanismo fiable de transmisión de TM en el proyecto de DFT4FTT, una infraestructura completa que facilita la implementación de DESs con requisitos de tiempo real, fiabilidad y adaptabilidad. Así como una explicación de antecedentes y tecnologías del ámbito, para entender el proyecto realizado.

Después de dicha implementación se harán una serie de experimentos diseñados para poder medir y evaluar el grado de fiabilidad con el que este mecanismo es capaz de operar, por ejemplo, medir los tiempos de sincronización o los tiempos de transmisión de cada mensaje.

En conclusión, este mecanismo ha cumplido con los resultados esperados, una buena sincronización, y tolerancia a fallos. El hecho de que el sistema operativo no fuese de tiempo real ha perjudicado en la medida de lo esperado el resultado de algunos experimentos.

6.2 Publicaciones

Parte del trabajo descrito en este documento fue indispensable para la realización de un artículo de congreso con revisión por pares:

- A. Ballesteros, M. A. Barranco, S. Arguimbau, M. Costa, J. Proenza. *Temporal Replication of Messages for Adaptive Systems using a Holistic Approach*. En Proceedings of the IEEE 24th International Conference on Emerging Technologies and Factory Automation (ETFA 2019), Zaragoza, 2019. (https://ieeexplore.ieee.org/document/8869470)

Este artículo es un *work-in-progress* en el que se describe el diseño y parcial implementación del mecanismo de replicación dinámica de mensajes en DFT4FTT. El objetivo es el de incrementar la tolerancia frente a fallos transitorios en el canal de comunicaciones, utilizando siempre los mínimos recursos. El artículo no solo cubre la replicación de los TMs, sino también de los mensajes de datos y mensajes de control para la reconfiguración. En este sentido, tanto la implementación como la experimentación llevada a cabo en este trabajo fueron necesarias para poder demostrar la viabilidad de la propuesta presentada en dicho artículo.

Bibliografía

[1] P. Pedreiras, L. Almeida, “The Flexible Time-Triggered (FTT) Paradigm: an Approach to QoS Management in Distributed Real-Time Systems”.

[2] D. Gessner, J. Proenza and M. Barranco, “A Proposal for Managing the Redundancy Provided by the Flexible Time-Triggered Replicated Star for Ethernet”.

[3] I. Alvarez, J. Proenza and M. Barranco, “Implementation and Verification of the Slave Elementary Cycle Synchronization Mechanism of the Flexible Time-Triggered Replicated Star for Ethernet”.

[4] “VDEv2: Virtual Distributed Ethernet”, 2021. https://github.com/virtualsquare/vde-2.