Universidad Zaragoza

Trabajo Fin de Grado

# Emulación de redes sensibles al tiempo (TSN)


Autor

Alex Gracia Rodríguez

Directores

José Luis Briz Velasco

*Univ. Zaragoza*

Héctor Blanco Alcaine

*Intel Corporation | Intel Deutschland GmbH*



ESCUELA DE INGENIERÍA Y ARQUITECTURA
2024

# AGRADECIMIENTOS

En primer lugar, quiero comenzar dando especialmente las gracias a mis tutores de TFG José Luis Briz y Héctor Blanco por confiar en mí para realizar este proyecto, y por todo el apoyo durante la realización del mismo. Su participación ha sido clave durante todo el desarrollo del proyecto. Sin sus consejos y su experiencia el resultado obtenido no habría sido posible.

En segundo lugar quiero agradecer al subgrupo de investigación de TSN del gaZ por su contribución de manera desinteresada a este TFG. Sus conocimientos y experiencia han sido indispensables para el éxito de este proyecto.

También quiero agradecer al Insituto de Investigación en Ingeniería de Aragón (I3A) por permitirme realizar este TFG mediante una beca de iniciación a la investigación. Sobretodo al, gaZ por permitirme utilizar su material como plataforma de experimentación.

Además quiero agradecer a todos mis compañeros de carrera, sobretodo a mis compañeros de Ingeniería de Computadores, mis compañeros de prácticas y a mi círculo más cercano, sin ellos este TFG no habría sido posible.

Por último, agradecer a mi familia por todo su apoyo. En especial a mis padres por la educación que me han brindado y por acompañarme y apoyarme durante todos estos años.

Este trabajo ha sido financiado por MCIN/AEI/10.13039/501100011033 (PID2022-136454NB-C22), Gobierno de Aragon (grupo T58_23R) e Insituto de Investigación en Ingeniería de Aragón (I3A, Conv. de Ayudas a Prácticas con TFG 2023)

# RESUMEN

Este Trabajo de Fin de Grado (TFG) se centra en el desarrollo de una plataforma para la emulación de redes Time-Sensitive Networking (TSN) sobre Mininet y en la validación de la plataforma como mecanismo para el despliegue de planificaciones TSN resultado de planificadores como[1]. Estas planificaciones garantizan cotas deterministas de latencia para los flujos de la red en la capa de enlace. El trabajo desarrollado ha permitido identificar aspectos clave en la implementación de redes TSN, y también limitaciones de los componentes TSN en la plataforma de emulación. Como parte del desarrollo de la plataforma se ha desarrollado una metodología para la caracterización de la plataforma, que permite caracterizar los distintos componentes con un impacto mínimo. Esta metodología se ha utilizado a continuación para analizar el impacto de la plataforma hardware subyacente sobre los resultados de cálculo de latencias, analizando diferentes configuraciones de kernel y hardware sobre sistemas con capacidades distintas. Este trabajo ha contribuido a la creación de una plataforma novedosa de emulación para el despliegue de casos de uso TSN cubriendo aspectos no considerados o no detallados en trabajos previos. La metodología, experimentación y conclusiones de este TFG se van a presentar próximamente en la confrencia Time Sensitive Networking and Applications (TSN&A) 2024,principal foro internacional anual de la industria y academia sobre TSN, a donde se envió y en donde fue aceptada la comunicación correspondiente [2].

# Abstract

This project focuses on the development of a platform for Time-Sensitive Networking (TSN) emulation over Mininet and on the validation of the platform as a mechanism for the deployment of TSN schedules resulting from schedulers such as [1]. These schedulers guarantee deterministic latency bounds for network flows at the link layer. The work developed has identified key issues in the implementation of TSN, as well as limitations of the components in the emulation platform. As part of the platform development, a methodology for timestamping and platform characterization has been developed. It allows characterizing the different components with minimal impact. This methodology was then used to analyze the impact of the underlying hardware on latency calculations by analyzing different kernel and hardware configurations on systems with different capabilities. This work has contributed to the creation of a novel emulation platform for the deployment of TSN use cases covering aspects not considered or not detailed in previous references. The methodology, experimentation and conclusions of this work were submitted, accepted and will be presented at the Time Sensitive Networking and Applications (TSN&A) 2024 Conference, the main annual international forum for industry and academia related to TSN [2].

# Índice

# Capítulo 1

# Introducción

## 1.1. Motivación y contexto

Time-Sensitive Networking (TSN) es un conjunto de tecnologías estándar del IEEE orientadas a garantizar la sincronización y cumplimiento de restricciones temporales en redes Ethernet e inalámbricas, permitiendo la interoperabilidad entre componentes de diferentes fabricantes. TSN es especialmente relevante en la integración de tecnologías operacionales y de la información (OT/IT) en la industria productiva, la distribución eléctrica, y las redes intra vehiculares en automoción e industria aeroespacial. Permite el despliegue de sistemas que requieren latencia ultra baja (Ultra-Low Latency (ULL) networks), la reducción de costes y la mejora de la eficiencia.

TSN proporciona mecanismos de conformación (*traffic shaping*) y planificación del tráfico para garantizar la calidad de servicio en redes en las convergen diferentes tipos de tráfico. Uno de los problemas abiertos es la implantación de un planificación previamente calculada en un caso de uso real, debido a factores difíciles de incluir en el problema teórico de planificación. Así mismo, la tendencia al establecimiento de redes definidas por software (Software Defined Network (SDN)) y en la nube (*cloudification*) abre la vía de test y validación de planificaciones y configuraciones TSN mediante procedimiento de emulación, en ausencia de *testbed* hardware.

La investigación en planificación y conformado de tráfico (*traffic shaping*) en TSN es una de las líneas de investigación del gaZ (Grupo de Arquitectura de Computadores de Zaragoza) de la Universidad de Zaragoza, desarrollada en colaboración con el CINVESTAV-IPN de Guadalajara, México, e Intel Deutschland GnbH. De esta colaboración han surgido hasta el momento dos contribuciones a revista ([3], [1]) y una comunicación a conferencia internacional [2], fruto esta última de este TFG.

## 1.2. Objetivos

### 1.2.1. Objetivos generales

Mediante la realización de este TFG se persigue, por una parte, la ampliación de conceptos relacionados con sistemas TR y redes, mediante la explotación integrada de conocimientos y habilidades propias de la titulación. Por otra, se busca un acercamiento tanto a los problemas y métodos propios de la investigación en TSN, como a la realidad y práctica industrial en el campo.

### 1.2.2. Objetivos específicos

- Estudio de las utilidades TSN disponibles sobre Linux y de las tecnologías utilizadas en High Performance Networking (HPN).

- Análisis de *Mininet* como herramienta de emulación y puesta en marcha de componentes TSN.

- Análisis experimental de métodos de medida de tiempos en TSN sobre Mininet. Propuesta de una metodología de medida.

- Validación de una planificación de un caso de uso TSN sobre Mininet

Este TFG contribuye a las metas 9.2 / 9.2.1; 9.4/9.4.1; 9.5 / 9.5.2 de los Objetivos de Desarrollo Sostenible.

## 1.3. Alcance

La consecución de los objetivos anteriores ha generado las siguientes entregables:

- *Testbed* de emulación TSN sobre Mininet, particularmente orientado a la validación de planificaciones, dotado de filtros, scripts de configuración, e instrumentación de registro de tiempos para cálculo de latencia y *jitter* según la metodología desarrollada en el TFG.

- Esta memoria de TFG con sus Anexos.

- Una comunicación aceptada en el principal foro internacional de TSN [2].

## 1.4. Metodología y entorno de trabajo

El tipo de trabajo ha requerido una aproximación experimental, además de la obligada consulta de fuentes. El estudio del rendimiento de sistemas como XDP, las diferentes `qdisc`, la tecnología TCC de Intel o la misma plataforma Mininet, resulta eminentemente empírico. Para alcanzar los objetivos, se han ensayado alternativas para comprobar las hipótesis que se han ido planteado, como se expone en los capítulos correspondientes. Además de los manuales de las diferentes tecnologías y plataformas involucradas, también se ha realizado una búsqueda bibliográfica y se han localizado y estudiado artículos relacionados (Sec. 2.4).

La principal herramienta de trabajo ha sido Visual Studio Code [4] con las extensiones para Python [5] y C/C++[6]. Python ha sido utilizado junto con la Api de Mininet [7] para el desarrollo de la plataforma de emulación ,mientras que C ha sido el lenguaje elegido para el desarrollo de los clientes y servidores a ejecutar sobre Mininet[7] junto con los distintos método de *profiling*(Cap. 4). Por otro lado se ha utilizado al biblioteca de Python matplotlib[8].

Para el desarrollo del software utilizado en este TFG se ha utilizado una máquina virtual desplegada sobre virt-manager[9] para comprobar el cumplimiento de requisitos, antes de desplegarlo mediante `ssh` sobre las diferentes plataformas de prueba.

Para el desarrollo de los diagramas utilizados en esta memoria se ha utilizado la herramienta `draw.io` [10].

Esta memoria se ha redactado mediante *Overleaf* [11], editor colaborativo de LaTeX.

La metodología y entorno específico de las partes experimentales describen en las Secs. 4.1 y 4.2. Las diferentes plataformas hardware en la que se han realizado los experimentos se pueden observar en la Tab. 4.1. Excepto en las Secs. 4.3.1 y 4.3.3, el resto de resultados experimentales en este TFG se han obtenido bajo la configuración CONF-2 (Sec. 4.1).

### 1.4.1. Consideraciones terminológicas

En esta memoria se traducen al español los términos comunes en Tecnologías de la Información y la Comunicación, como por ejemplo trama (*frame*) o flujo (*flow, stream*). Los términos *talker, listener* tienen connotaciones muy específicas en TSN pero en todo caso los traducimos como *emisor* y *receptor* respectivamente. Preservamos sin embargo en inglés términos específicos que raramente por no decir nunca se traducen el campo, e.g. software, hardware, *bridge*, *end station*, *shaping* o *end-to-end* entre otros. Reservamos e término *núcleo* para un *core* de una CPU, y utilizamos *kernel* para referirnos al núcleo de un sistema operativo, algo por otra parte muy común

en el caso de Linux. Se ha hecho un esfuerzo para mantenerlos en *cursiva*. También se marcan en `typewriter` parámetros, estructuras singulares de datos, metadatos y elementos similares.

En la literatura sobre TSN, especialmente en la comercial, los términos *bridge* y *switch* se utilizan indistintamente. En este trabajo utilizamos el término *bridge*, que es el utilizado en las recomendaciones del estándar IEEE 802.1Q.

Por otra parte, el número de acrónimos relacionados con TSN evoluciona, muta, y espanta a cualquier persona poco familiarizada con esta tecnología. Por ello se han generado enlaces para que en cualquier momento pueda consultarse la definición y regresar al punto de lectura.[1].

## 1.5. Planificación

El desarrollo de este TFG se ha realizado de manera progresiva. En la primera etapa de desarrollo se estudió la documentación correspondiente a TSN. Además de localizar trabajos relacionados con el trabajo a realizar. Una vez estudiado el material encontrado se procedió con el desarrollo de la plataforma de emulación.Una vez desarrollada la plataforma de emulación se procedió a desarrollar una metodología de medición válida para la plataforma. Con la plataforma desarrollada y como culminación del TFG se implemento un caso de uso TSN sobre la plataforma. Además durante toda la realización del TFG se ha participado en las reuniones del grupo de investigación. La Fig.1.1 muestra las tareas y como se han repartido a lo largo del desarrollo del TFG.

## 1.6. Estructura del documento

Esta memoria de TFG se estructura como sigue. El Cap. 2 introduce los conocimientos técnicos necesarios para el seguimiento de este TFG, especialmetnte en lo relativo a TSN. El Cap. 3 identifica los diferentes mecanismos TSN incluidos en el kernel de Linux y discute su integración en Mininet. El Cap. 4 desarrolla una metodología de registro de tiempos para el cálculo de latencia y *jitter*, a partir de diferentes opciones de *profiling*, así como las posibles optimizaciones de la plataforma de emulación en su conjunto. El Cap. 5 define y despliega un Caso de Uso sobre la plataforma de emulación preparada a fin de validar el resultado de una planificación TSN, incidiendo en nuevas cuestiones que surgen al realizar la configuración y puesta en marcha. Finalmente, el Cap. 6 recapitula los resultados experimentales, recoge conclusiones y traza líneas futuras.

[1] E.g. `Alt-<flecha> en las utilidades de Adobe Acrobat©`

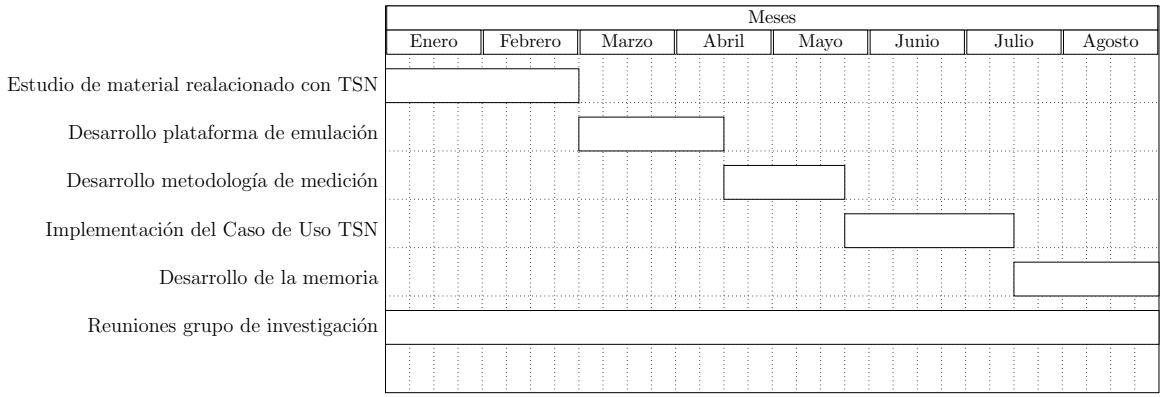



Figura 1.1: Diagrama de Gant

# Capítulo 2

# Fundamentos

En este capítulo se definen los términos y se explican los conceptos necesarios para que esta memoria sea autocontenida en lo posible. Existe una amplia literatura sobre conceptos generales relacionados con redes (e.g. [12],[13]). En el caso de TSN, puede encontrarse una visión general en [14, 15]).

## 2.1. Conceptos generales

### 2.1.1. Redes

**Red de computadores** Conjunto de nodos interconectados por un medio físico que se comunican entre sí.

**Capa de enlace (DLL)** Segunda capa del modelo OSI responsable del intercambio de datos entre el *host* (nodo anfitrión) y el resto de elementos de la red a la que esta conectado. Es habitual encontrarla referida como *Layer 2* en inglés. Su ejemplo más relevante en la es *Ethernet* .

**Latencia (*delay*)** Tiempo que tarda en transmitirse un paquete de datos desde su origen hasta su destino.

***jitter*** Fluctuación de la latencia. El jitter evalúa la diferencia de la latencia de distintas tramas mostrando la desviación con respecto a la latencia.

**Virtual Local Area Network** Mecanismo que permite generar distintas redes lógicas a partir de una misma red física. Las tramas de red se etiquetan con un campo VLAN en el que se indica a que VLAN (red lógica) pertenecen.

### 2.1.2. Linux Network Stack (LNS)

Linux Network Stack (LNS) engloba las soluciones software que permiten la comunicación entre Linux y las tarjetas de red (NIC) mediante la recepción (Rx) y la

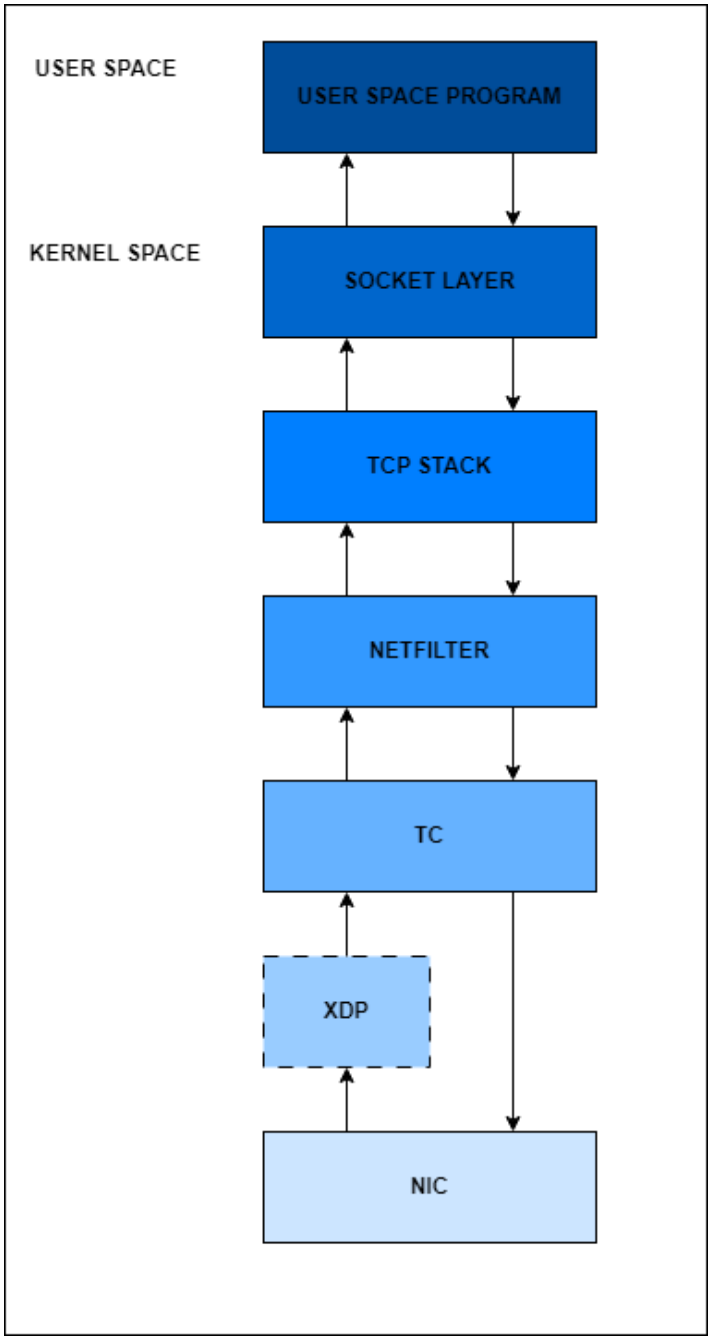



Figura 2.1: Esquema de la estructura de la Linux Network Stack.

transmisión (Tx) de las tramas de red desde las aplicaciones de usuario. La Fig.2.1 muestra las diferentes capas de la LNS.

Las herramientas de la LNS permiten al administrador interactuar con el tráfico de red. Algunas como Linux Traffic Control (`tc`) son clave en este TFG, junto al sistema de colas (`qdisc`) del LNS, especialmente la denominada Time-Aware Priority Shaper (`taprio`) (Sec. 3.1.1).

### 2.1.3. Socket Buffer (SKB)

Todas las colas y búferes de la LNS del kernel utilizan una estructura común, consistente en una lista de elementos `struct sk_buff` (Socket Buffer (`skb`), Fig. 2.2). Reúne información sobre todas las tramas que están siendo procesadas. Cada elemento en `skb` almacena una trama junto a metadatos tales como el instante de recepción, el instante en el que debe de ser transmitida o su prioridad (`skb`).

Los `skb` proporcionan una abstracción de los protocolos subyacentes, y elevan el nivel de abstracción en la interacción de las aplicaciones de red con el sistema operativo. Esto impacta en la latencia y el *jitter* de las aplicaciones utilizando dicho interfaz.

### 2.1.4. *Network Namespaces* y *Virtual Ethernet Pairs*

Los *network namespaces* son una funcionalidad integrada en el kernel de Linux que permite que un proceso tenga su propia LNS aislada del resto de procesos del sistema.

Los *virtual Ethernet pairs* (pares `veth`) son parejas de interfaces de red que se utilizan para interconectar procesos que se encuentran en diferentes *network namespaces*. Para ello se genera una interfaz de red en cada uno de los distintos *namespaces* a conectar.

Por defecto, los pares `veth` solo cuentan con una cola de recepción (*RX queue*) y una de transmisión (*Tx queue*). Esto supone un problema a la hora de integrar tecnologías TSN como `taprio`, que se fundamentan en la existencia de múltiples colas. Este problema se aborda en la Sec. 3.2.2..

### 2.1.5. Berkeley Packet Filter (BPF)

Berkeley Packet Filtering (BPF) es un sistema que permite ejecutar de forma segura funciones definidas desde programas de usuario dentro de las interfaces de red de un kernel UNIX. Su ámbito de aplicación principal son reglas sobre paquetes de red ejecutadas por el cortafuegos del kernel.

La codificación de estas funciones está sometida a unas restricciones cuyo cumplimiento se verifica antes de pasarlas al kernel mediante un compilador JIT como

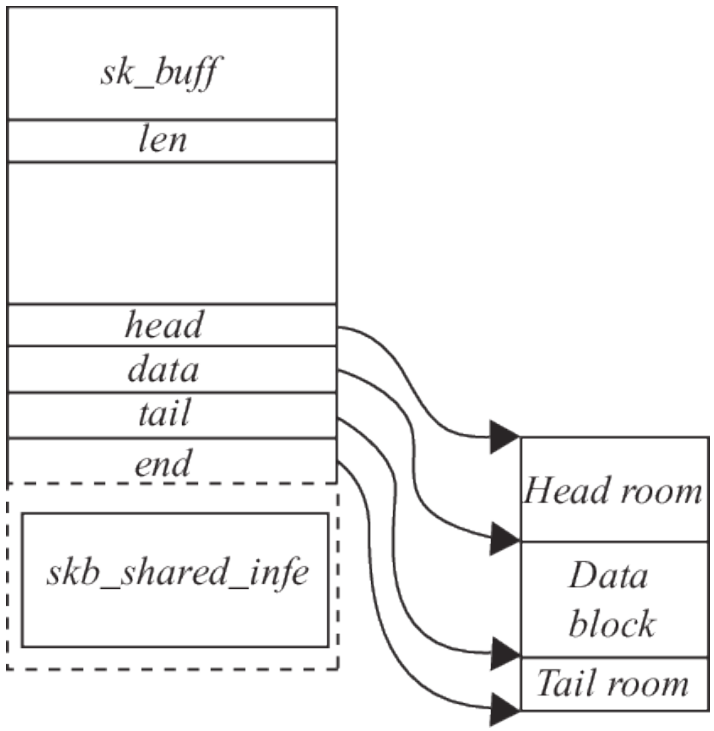



Figura 2.2: Esquema de la estructura `skb` del kernel,Kernel Implementation of Sockets[16] Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Network-buffer-sk-buff_fig1_285355742 [accessed 1 Sept 2024]

programas BPF para su ejecución en un entorno seguro, aislado del resto del sistema. Supone una forma sencilla de extender la funcionalidad del kernel sin necesidad de modificar su código estática o dinámicamente (mediante módulos).

BPF se basa en la arquitectura de lenguaje máquina del mismo nombre. Su versión original de principios de los '90 se transformó notablemente en el eBPF, al que actualmente nos referimos simplemente como BPF. Su función original (la relevante en este TFG) es el filtrado eficiente y seguro las tramas recibidas, pero se utiliza también para *profiling* o incremento de la seguridad en el kernel.

#### Mapas BPF

Un mapa en BPF es una estructura de datos compartida entre los programas BPF y el kernel del sistema operativo. Los programas BPF acceden a los mapas bien para comunicarse entre ellos, bien para hacerlo con procesos que se ejecutan a nivel de usuario.

### 2.1.6. eXpress Data Path (XDP)

eXpress Data Packet (XDP) [17] es un sistema que permite el procesamiento de las tramas de red en el nivel inicial de la LNS, sin pasar por el resto de niveles, incre-

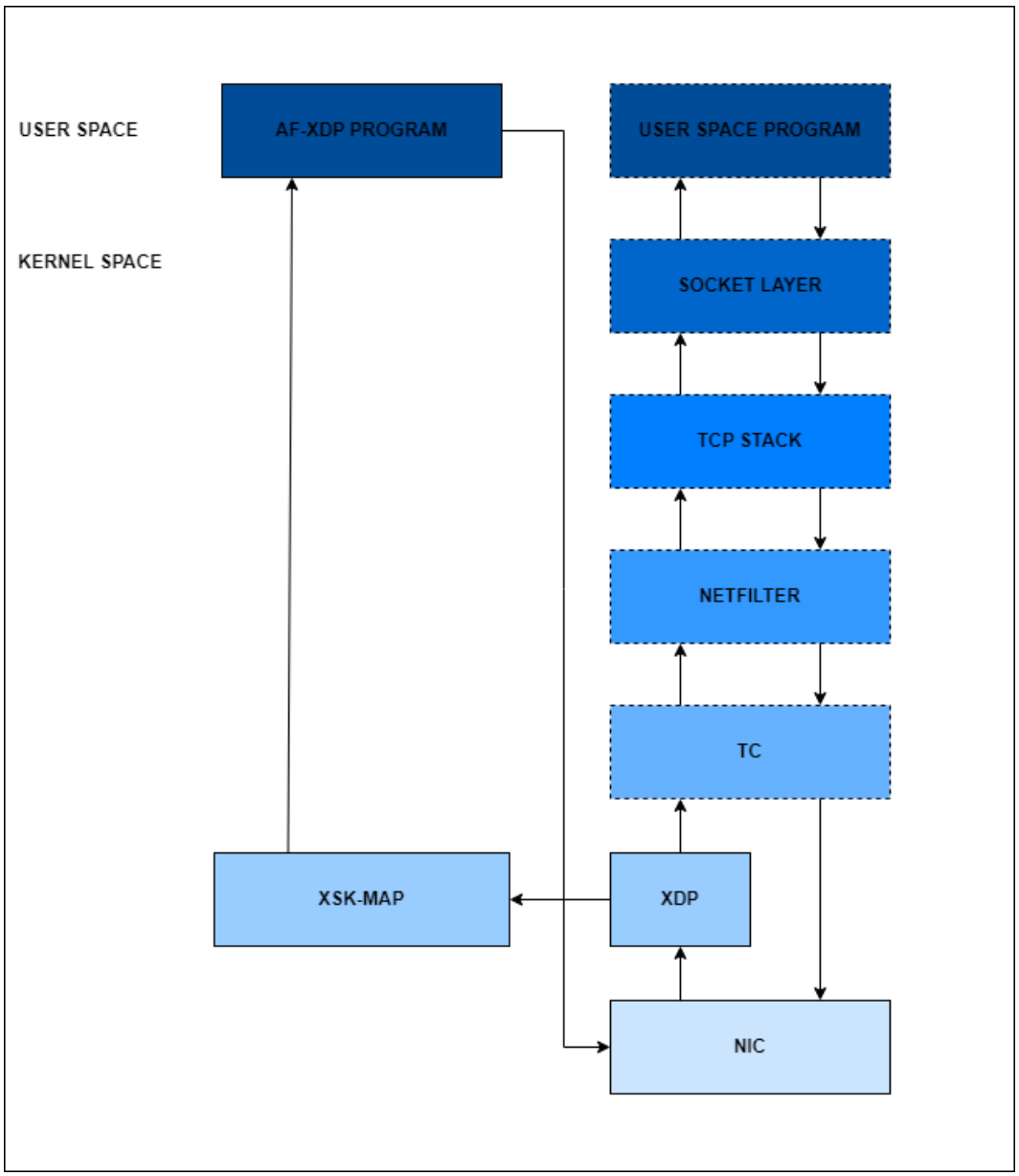



Figura 2.3: Esquema de XDP

mentando drásticamente el rendimiento (Fig.2.3). XDP es de desarrollo relativamente reciente. Surge para superar inconvenientes de métodos previos, fundamentalmente Data Plane Development Kit (DPDK) [18], aún en uso. Igualmente orientado al procesado de tramas antes de que éstas alcancen la LNS, DPDK es tan eficiente como dependiente del hardware subyacente, y mucho más complejo de utilizar de forma segura.

Basado en BPF, XDP permite que cada trama entrante sea procesada por un programa BPF establecido desde un programa de usuario. XDP instrumenta el driver del NIC añadiendo una llamada (*hook*) al programa BPF asociado, justo en el retorno de la rutina de servicio a interrupción, y antes de cualquier operación de asignación de memoria (para evitar su sobrecoste temporal).

Usos habituales de XDP son, por ejemplo, enviar información sobre las tramas recibidas a una aplicación de usuario, filtrar el tráfico de la red[1], o reenviar las tramas a otros interfaces de red o a procesos de usuario para su procesamiento.

### 2.1.7. Expulsión en el kernel

La expulsión de procesos (*preemption*) es un mecanismo mediante el que los sistemas operativos controlan el reparto del tiempo de CPU entre las diferentes tareas sujetas al planificador.

La expulsión (*preemption*) de procesos implica que el sistema operativo se apropia de una CPU en la que se está ejecutando un proceso, expulsando a este último para ceder el uso de dicha CPU a otro proceso. En planificadores orientados a multiprogramación de propósito general, esto sucede por ejemplo cuando el proceso en curso agota su cuota de uso consecutivo de CPU (*time slice*), a fin de conseguir un reparto equitativo o ponderado de las CPUs entre los procesos preparados. En planificadores tiempo real expulsivos, sucede cuando se activa una tarea de mayor prioridad que la que está en curso. La explicación de esta cuestión queda fuera de los objetivos de este TFG y se remite a la sección de documentación de los fuentes del kernel de Linux como referencia preferible [19]. Incluimos aquí una breve explicación, que permita comprender algunas de las cuestiones abordadas en el TFG (e.g. Sec. 3.2.1).

La expulsión presenta un problema. El proceso a expulsar puede por ejemplo estar ejecutando código del kernel en su propio contexto de proceso (i.e. tras una excepción síncrona derivada de una llamada al sistema o un fallo de validez por ejemplo), modificando una estructura crítica accesible por otras excepciones. Si se le expulsa antes de dejar en estado estable la modificación (e.g. una inserción en lista), todo el sistema queda inestable y se producirán errores. La forma de gestionar este problema y la exclusión mutua subyacente depende del propósito de cada sistema operativo y de la arquitectura subyacente. En la actualidad Linux soporta tres modelos de expulsión, al que se suman otros dos si se utiliza el parche `PREEMPT_RT` para tiempo real.

- *No Forced Preemption* - Mientras un proceso está ejecutando código del kernel, solo abandona la CPU si invoca voluntariamente (*inovcación directa*) al planificador (`schedule()`), o bien en el código de regreso de la última excepción anidada, si así lo indica la variable `need_resched` (*lazy invocation*). Es el modelo con menos sobrecarga, y también menos agilidad.

- *Voluntary Kernel Preemption* - Variante de la anterior que añade invocaciones

[1]XDP permite filtrar (descartar) 24 millones de tramas por segundo sobre NIC convencional [17], convirtiéndolo en una herramienta muy útil como defensa de ataques de denegación de servicio

voluntarias al planificador en el código del kernel que se ejecuta en conexto de proceso (no en el que se ejecuta en contexto de interrupción), que tienen efecto sólo si existen procesos preparados de mayor prioridad. Mejora la respuesta en equipos personales usados para trabajo y entretenimiento o comunicación, y posiblemente se elimine en breve.

– *Preemptible Kernel* - Un proceso que ejecuta código de kernel (en contexto de proceso) puede ser expulsado en cualquier momento excepto si está ejectuando el código de una sección crítica.

– *Preemptible Kernel (RT)* - Versión de la anterior en el patch `PREEMPT_RT` en el que las rutinas de kernel que se ejecutan en contexto de interrupción (rutinas de servicio a interrupción, *softirqs* y *tasklets*) se implementan como *kernel threads*.

– *Fully Preemptible Kernel* (RT) - Todo el código del kernel es expulsable excepto en secciones críticas muy limitadas. Como en el caso anterior, rutinas de servicio asíncronas, *softirqs* y *tasklets*, se gestionan como *kernel threads* independientes). En consecuencia todas las rutinas de excepción y actividades de kernel son planificadas, no entrelazadas. Es decir, por ejemplo la ocurrencia de una interrupción conlleva la invocación del planificador, que dará paso a la correspondiente rutina de servicio o no, según la prioridad que tenga asignada. Los *spinlocks* se substituyen por semáforos `mutex_rt` (un mecanismo bloqueante, por definición).

## 2.2. Time-Sensitive Networking (TSN)

TSN es un conjunto de mecanismos estandarizados por el IEEE 802.1 TSN Working Group. Su objetivo principal es permitir establecer cotas deterministas de latencia y *jitter* en flujos prioritarios (i.e. sometidos a restricciones temporales) sobre redes Ethernet y WiFi convencionales. Describen mecanismos y procedimientos estándar que facilitan la interoperabilidad de componentes de diferentes fabricantes. También buscan proporcionar servicios estándar de calidad de servicio (QoS, gestión y tolerancia a fallos, que las tecnologías propietarias no satisfacen o lo hacen sólo parcialmente.

El IEEE 802.1 TSN Working Group también cubre áreas relacionadas como la sincronización del tiempo en una red Ethernet, o las garantías de ancho de banda con aplicaciones en el transporte de audio y vídeo a través de una red Ethernet.

Los estándares TSN permiten que flujos de diferente criticidad convivan en la misma red, de forma que los flujos no prioritarios no afecten a las latencias de los prioritarios.

### 2.2.1. Nodos TSN

Una red TSN consta de nodos terminales (*end points* o *end stations*) y de elementos de comunicación (*bridges*). Dentro de los *end points* podemos diferenciarlos entre *talkers* o *listeners*:

***Listener* (*receptor*)** Nodo perteneciente a una red TSN que es el destino final de un flujo. Un nodo puede ser tanto listener como talker al mismo tiempo.

***Talker* (*emisor*)** Nodo perteneciente a una red TSN que es el origen de un flujo.

***Bridge*** Nodo encargado de interconectar distintos nodos de una red y controlar el tráfico.

### 2.2.2. Flujo

En TSN un flujo es una secuencia de tramas (*frames*) que comparten características y requisitos, como el mismo nodo de origen y mismas restricciones temporales. A cada flujo se le asocia un código de prioridad codificado en el campo PCP (3 bits) de la cabecera VLAN de su trama Ethernet (Fig. 2.4). Así, mecanismos como el TAS (IEEE 802.1Qbv, Sec. 2.2.4) pueden disponer de hasta ocho colas para gestionar hasta ocho clases diferentes.

A efectos de clarificar la terminología seguida en esta memoria según los estándares y el uso habitual, conviene señalar que IEEE 802.1Q-2014 [20] (Anexo II) asocia ocho prioridades diferentes a ocho clases de tráfico (*traffic class*) según sus requisitos (Tab. 2.1). En la práctica, se distinguen menos clases según el ámbito de uso. Por ejemplo, el Internet Industrial Consortium (iiC) distingue cinco clases (Tab. 2.2) en el contexto IACS.

Sin embargo, se diferencian tres clases generales de tráfico a efectos de asignación de recursos de red y modulación de tráfico sobre tecnología TSN (ver e.g. [14]):

**Clase CDT** *(Control-Data Traffic)* Táfico de control de red o *Excellent Effort* que requiere el menor retardo posible.

**Clase A** Tráfico de aplicaciones críticas, que suele denominarse *tráfico TSN*, *sensible al tiempo (time-aware traffic)*, *sujeto a restricciones temporales (time-constrained* y otras expresiones similares, que incluyen restricciones de audio y vídeo.

**Clase B** Tráfico *Best Effort* (no prioritario) (BE).

| Prioridad | Tipo de tráfico |
|---|---|
| 0 | *Background* |
| 1 | *Best Effort* (BE) |
| 2 | *Excellent effort* |
| 3 | Aplicaciones críticas |
| 4 | Video, < 100 ms latencia y jitter |
| 5 | Voz, < 10 ms latency and jitter |
| 6 | *Internetwork control* |
| 7 | Network control |

Tabla 2.1: Asignación de prioridades a distintos tipos de tráfico [20]

| Tipo de Tráfico | Descripción |
|---|---|
| Control de red | Tráfico de máxima prioridad |
| *Excellent Effort* | Tráfico necesario para<br>la configuración y gestión de la red |
| Tráfico de aplicaciones críticas | Tramas que precisan<br>recepción con retardo limitado |
| Voz (audio) | |
| Video | |

Tabla 2.2: Tipos de tráfico en redes TSN [21]

La asignación a cada flujo de los valores específicos codificados en el campo PCP se realiza en la configuración de la TSN según el caso de uso y las capacidades de los moduladores de tráfico (*traffic shapers* o simplemente *shapers* en lo sucesivo) disponibles en la infraestructura. Parte del esfuerzo de este TFG se ha dirigido a la solución de problemas de gestión del campo PCP relacionados con la configuración de la `taprio` sobre Mininet (Sec. 3.1.1).

### 2.2.3. Sincronización del tiempo

Uno de los aspectos imprescindibles en TSN es la sincronización de los relojes de los nodos de la red, un problema compartido con las redes industriales. Por ello, TSN se apoya el sistema preexistente IEEE 1588 (Precision Time Protocol (PTP)), muy conocido para sincronizar los relojes de los diferentes nodos.

El estándar IEEE 1588 define mecanismos generales de sincronización, aplicables a diferentes tipos de redes, e incluso a distintos niveles dentro de la misma tecnología. Por ejemplo, define mecanismos de *timestamping* en la capa 2 o en el nivel IP. También define una serie de aspectos parametrizables, como el intervalo de sincronización, e incluso permite cierto nivel de personalización de las máquinas de estados asociadas al protocolo.

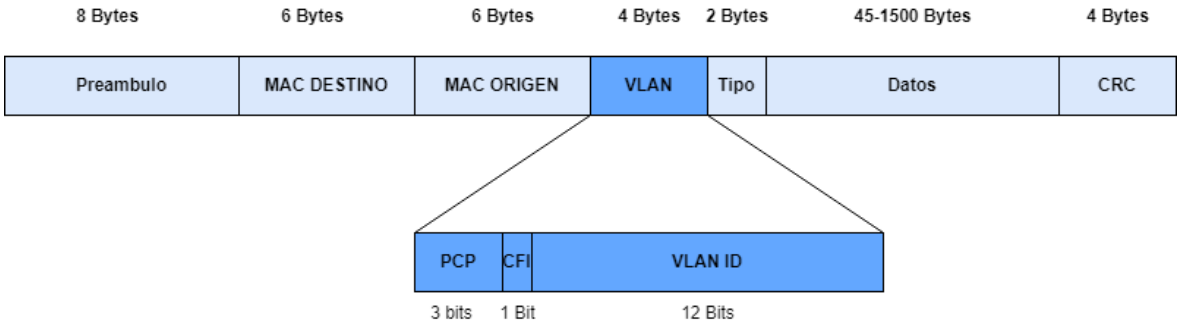



Figura 2.4: Campos VLAN y PCP en una trama Ethernet

Las parametrizaciones del estándar IEEE 1588 reciben el nombre de *perfiles*. Un perfil define valores específicos para los parámetros, y puede limitar las funcionalidades y tipos de red a soportar para que un sistema se pueda considerar conforme al estándar.

PTP define el protocolo para propagar el tiempo proporcionado por un nodo de referencia al resto e nodos de la red.

El protocolo elige un reloj denominado como *grandmaster* (GM) que es utilizado como referencia por el resto de los relojes de la red. El GM envía periódicamente tramas de sincronización al resto de nodos de la red. El protocolo incluye mecanismos para medir y cancelar los retardos de propagación en los que incurre el transito de los mensajes desde el GM hasta los *followers*.

Las implementaciones basadas en software proporcionan desviaciones entre tiempo en el GM y los *followers* por debajo de 1 ms. El soporte hardware permite reducir las desviaciones por debajo de 1 us, alcanzando unos pocos nanosegundos.

TSN define un perfil de IEEE 1588 en el estándar IEEE 802.1AS, también conocido como Generalized Precision Time Protocol (gPTP).

IEEE 802.1AS (alias gPTP) restringe el ámbito del protocolo a la capa de enlace Ethernet (802.3) y Wi-Fi (802.11), lo que permite mejorar la precisión de la sincronización. También especifica diferentes parámetros, como el numero de dominios temporales soportados.

### 2.2.4. Control de flujo

Uno de los aspectos definidos en el estándar IEEE 802.1Q es el control de flujo para redes TSN. Algunas de las extensiones relevantes integradas en dicho estándar son 802.1Qav ó 802.1Qbv, que especifican mecanismos para regular el tráfico y garantizar plazos de entrega de los flujos prioritarios.

### Modulado (*shaping*)

El *control* de tráfico consiste en administrar la cantidad y el tipo de flujo permitido en la red. La *modulación* del tráfico (*traffic shaping*) consiste en limitar la tasa de transmisión de los flujos.

### IEEE 802.1Qav Credit Base Shapper (CBS)

CBS es un sistema basado en los algoritmos *leaky bucket* que limita el ancho de banda que puede utilizar una clase de tráfico. Cada una de las colas de transmisión (*RX queues*) tiene asociado un contador de crédito, que aumenta cuando no se están transmitiendo tramas a través de esa cola y disminuye mientras se transmiten.

Cuando un trama llega a la cola se verifica el contador de créditos. La trama se transmitirá si este contador es superior a un parámetro definido por el administrador o el CNC (ver Sec. 2.2.5). En caso contrario quedará en la cola hasta que el contador supere al parámetro.

### IEEE 802.1Qbv Time Aware Shapper (TAS)

TAS es otro sistema de *shaping* basado en algoritmos *leaky bucket*, más flexible y complejo que CBS. La Fig. 2.5 muestra la estructura genérica de este *shaper* según se describe en el estándar IEEE 802.1Qbv.

El TAS está formado por una estructura de hasta 8 colas diferentes para gestionar diferentes tipos de tráfico con la posibilidad de un *shaper* secundario por cola.

La transmisión de las tramas asignadas a cada cola vienen definidas por unas Gate Control List en las cuales para cada instante de tiempo se definen que colas pueden transmitir. Dichas GCL son el resultado de resolver un problema de planificación ( 2.2.4).

Cuando una trama llega al TAS se le asigna una cola dependiendo de la clase asociada a dicha trama. Una vez la trama llega a la cola se verifica si la entrada de la GCL activa. Si dicha cola puede transmitir la trama es enviada si no , la trama se queda en la cola hasta que se pueda transmitir.

En el caso de la Fig 2.5 en el instante de tiempo T04, podrán ser trasmitidas las tramas que se encuentren encoladas en las colas siete y cinco y en el siguiente ciclo del *scheduler* podrán ser transmitidas las tramas de las colas 3 y 4.

### Planificación

El problema de planificación de flujos con restricciones temporales y de *jitter* es un problema NP-Completo, que se aborda mediante tres tipos de métodos: de opti-

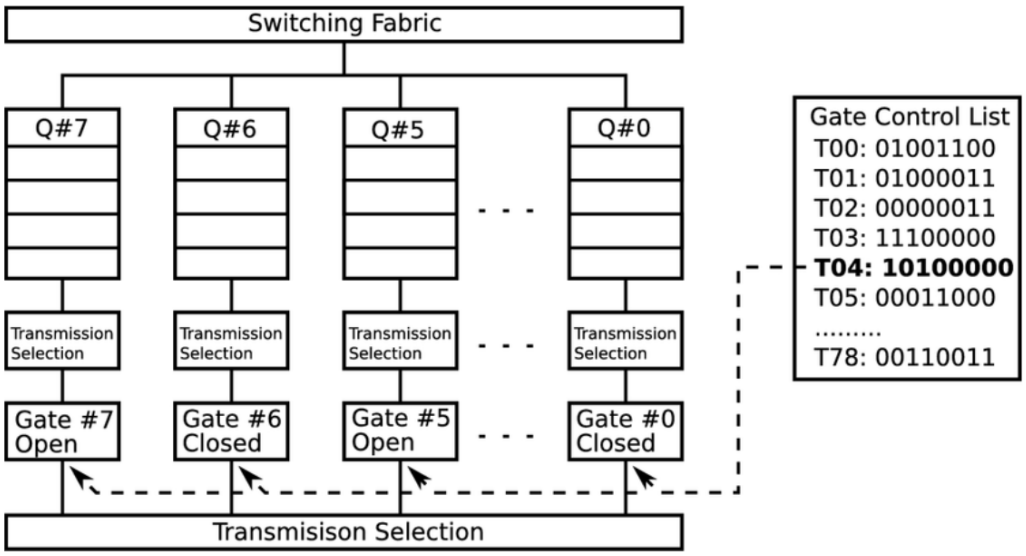



Figura 2.5: Estructura del TAS IEEE 802.1Qbv

mización, heurísticos y mixtos [22][23]. A partir de las planificaciones obtenidas por estos métodos se sintetizan las GCL de los TAS (IEEE 802.1Qbv). Este problema de planificación en TSN y síntesis de GCL es una línea del proyecto de investigación en el que se enmarca este TFG (Sec. 1.1), a la que este TFG contribuye proporcionando un *testbed* emulado (Cap. 3) y validando soluciones de planificación propuestas en el proyecto hasta ahora (Cap. 5).

### 2.2.5. Gestión y configuración de recursos

La gestión recursos es un aspecto importante en TSN. No es central en este TFG pero aparece inevitablemente al realizar configuraciones (Cap. 5) y por ello lo introducimos aquí.

El estándar IEEE 802.1Qat define el protocolo SRP mediante el cual los *talkers* anuncian sus flujos al resto de los nodos de la red. Los *bridges* reciben el anuncio y en caso de poder reservar los recursos necesarios para los flujos anunciados, remiten dicho anuncio al resto de los nodos de la red.

La configuración y gestión de una red TSN puede hacerse de forma centralizada (Centralized Network Configurator (CNC)) o distribuida según el caso de uso. IEEE 802.1Qcc define el protocolo centralizado Centralized Network Configurator (CNC), la opción más utilizada hoy día en sistemas industriales, y el interfaz Centralized User Configuration (CUC) para desarrollar aplicaciones de usuario que interactúen con CNC. CUC/CNC permiten la configuración de la red mediante protocolos de configuración

como NETCONF/RESTCONF o mediante modelos IEEE 802.1Q YANG.

La Fig. 2.6 ejemplifica un sistema de este tipo. El administrador define a través de un Centralized User Configuration (CUC) la topología de red, los flujos a ejecutar juntos con sus restricciones y otros posibles parámetros. El CUC carga estos datos en el CNC. El CNC explora la red (nodos, enlaces, características), realiza la configuración, calcula la planificación y la despliega, verifica, y si todo está en orden, inicializa y pone en marcha el sistema.

Algunos fabricantes de *bridges* TSN ofrecen herramientas e interfaces de configuración y gestión de recursos, pero se trata de un aspecto aún en investigación y desarrollo, con iniciativas como OpenCNC [24].

### 2.2.6. Tolerancia a fallos

Las recomendaciones IEEE 802.1CB e IEEE 802.1Qci de TSN aseguran de que las tramas lleguen a destino en caso de fallo en algún nodo de la red, y permiten la detección de nodos maliciosos. Este aspecto queda fuera de los objetivos de este TFG.

## 2.3. Simulación, emulación y *testbedding*

### 2.3.1. Conceptos generales

Simular supone diseñar un modelo de un sistema físico, en general con el propósito de registrar o controlar todos los posibles estados del modelo al menos durante un cierto intervalo de simulación, sometiéndolo a unas entradas determinadas. Este modelo recoge selectivamente características del sistema físico, según el propósito de la simulación. Permite un seguimiento detallado de todas las variables del modelo deseadas, disparando el coste temporal de la simulación en función del grado de detalle.

La emulación, por el contrario, sólo se ocupa de imitar el comportamiento del sistema físico, mediante virtualización por ejemplo, sin menoscabo de instrumentar el sistema con técnicas similares a las utilizadas en un sistema real (e.g. hardware o software *profiling*). El sistema emulado puede reemplazar al sistema físico porque su comportamiento (e.g. respuesta a cambios en las entradas) reproduce el del sistema físico.

En nuestro contexto, *testbedding* consiste en crear un prototipo de un sistema físico mediante componentes físicos, comúnmente dotado de un entorno de gestión y monitorización que facilita la realización de pruebas y la extracción de métricas de su comportamiento.

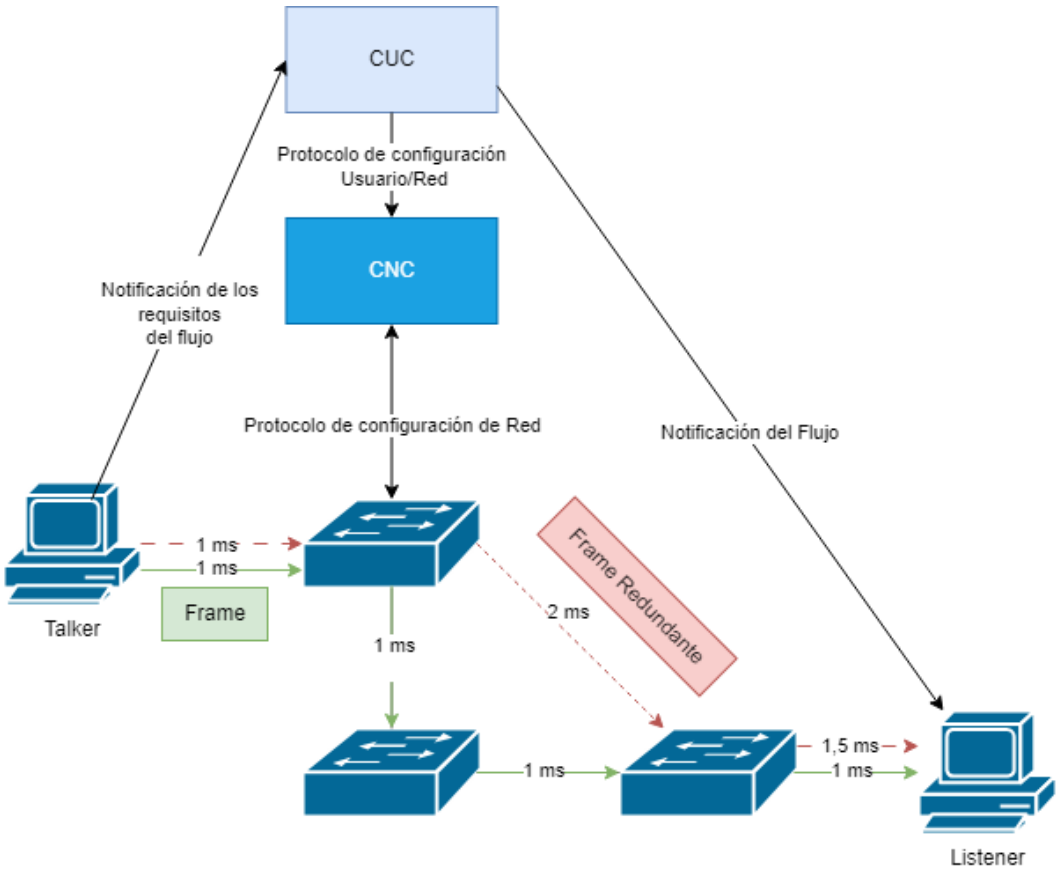



Figura 2.6: Ejemplo de configuración de una red TSN mediante un CNC.

### 2.3.2. Simulación de redes TSN

La simulación de redes consiste en crear modelos discretos de casos de uso (nodos, enlaces, flujos y su planificación según restricciones). Creado el modelo, se generan los eventos correspondientes al caso de uso y se obtienen las métricas del sistema (e.g. latencias y *jitter*). Existen varias herramientas que permiten la simulación de redes TSN. Por ejemplo NeSTiNg[25] es un *framework* que añade al simulador de redes Omnet++[26] funcionalidades TSN tales como los *shapers* TAS y CBS. NeSTiNg/Omnet++ son gratuitos para instituciones académicas, y se distribuyen bajo una licencia de código abierto. RTaW-Pegase[27] es otro simulador, esta vez de pago, que soporta la mayoría de las funcionalidades TSN.

El principal inconveniente de estos simuladores es su rendimiento, quedando limitados en la práctica a casos de uso muy sencillos.

### 2.3.3. Mininet

Actualmente, el principal emulador de redes de código abierto TSN es Mininet.

Mininet permite definir y ejecutar un conjunto de nodos (*end points / hosts*, *bridges*) y enlaces sobre un único sistema Linux. Los *hosts* de Mininet se comportan como los

físicos. Mininet permite ejecutar sobre ellos aplicaciones de usuario que pueden enviar tramas a través de interfaces virtuales como el `veth` de Linux, transmitiéndolas a través de los *switches* también emulados.

Es posible diseñar una red en Mininet que emule una red física, o bien diseñar una red física según la diseñada en Mininet, de forma que en ambas plataformas se ejecuten las mismas aplicaciones con igual código binario.

En la Sec. 3.2.1 se abordarán detalles de implementación y configuración relevantes para la metodología de medición de latencias.

### 2.3.4. Testbedding

La manera más precisa de probar configuraciones TSN es la realización de las pruebas sobre una plataforma física (*testbedding*). Un modelo físico (*testbed*) típico integra *bridges*, *hosts* con tarjetas de red (NIC) que soportan estándares TSN, como IEEE802.1AS para asegurar una sincronización temporal correcta.

Al comienzo del desarrollo de este TFG se consideró desplegar los mecanismos TSN utilizados sobre Mininet en hardware real. La principal dificultad han sido, por una parte, los tiempos de entrega de los componentes necesarios[2] y, por otra, la imposibilidad de acceder a plataformas de *testbedding* existentes en los plazos propios de un TFG.

## 2.4. Trabajos relacionados

Existe un limitado número de trabajos relacionados con los problemas que presenta la emulación de TSN en Mininet [28][29]. Ambos identifican —y no siempre resuelven— problemas de integración de componentes TSN sobre los *bridges* generados por Mininet. En la Sec. 3.2 llevamos a cabo nuestra propia identificación y aporte de soluciones, sobre la base de esos trabajos previos.

Por otro lado, existe un estudio sobre el efecto del implementar TAS en software, comparándolo con su implementación en un *testbed* hardware [30]. Los autores desarrollan una metodología de medida, pero no detallan su implementación.

En [31] se recurre a una red TSN emulada sobre Mininet para estudiar las posibilidades de una aproximación SDN a la hora de implementar tolerancia a fallos en una red TSN. El articulo se centra más en utilizar las capacidades de SDN de Mininet que en la implementación de los mecanismos TSN existentes en Linux. También desarrollan una metodología de *profiling*, cuya aproximación diferente a la de [30] pero obtiene

[2] A fecha de escritura de este TFG el tiempo de espera para un equipamiento mínimo con capacidades TSN se sitúa en un año y medio

resultados similares.

Ha sido útil consultar [32], un trabajo que recoge bien las distintas utilidades TSN existentes en el ecosistema Linux. Los autores despliegan dichas utilidades sobre un *testbed* físico, y no sobre un sistema emulado como en este TFG.

# Capítulo 3

# TSN en Linux y Mininet

La emulación de funcionalidades TSN en Mininet utiliza por una parte subsistemas TSN específicos para Linux como Time-Aware Priority Shaper (`taprio`) o el protocolo gPTP, y componentes de Linux para virtualización tales como Virtualized Ethernet (`veth`) por otra. Estos elementos son relativamente recientes y siguen en desarrollo activo, por lo que presentan problemas de estabilidad y compatibilidad entre versiones. En este capítulo se describen en primer lugar los elementos utilizados en el TFG, y a continuación los problemas encontrados y las soluciones adoptadas durante su integración, configuración y empleo.

## 3.1. Elementos de soporte de TSN y virtualización

### 3.1.1. Linux Traffic Control

Este subsistema se ocupa de clasificar, arbitrar y planificar las tramas que constituyen el tráfico de red mediante disciplinas de colas (`qdisc`) y filtros. Las `qdisc` encolan las tramas entrantes y salientes del interfaz de red.

Existen dos tipos de `qdisc`:

***Classful qdiscs*** Encolan y después desencolan las tramas en colas pertenecientes a *clases hijas*.

***Classless qdiscs*** Colas que no tienen ninguna clase hija.

Los filtros permiten ejecutar acciones directamente sobre las tramas de red. Los usuarios pueden establecer filtros tanto en la entrada como en la salida, para modificar tanto la trama como la meta-información de la misma. Por ejemplo, es posible crear un filtro que modifique la clase a la que pertenece la trama es decir, que cambie el valor del campo PCP de la cabecera VLAN de la trama.

A continuación se describen algunas de las `qdisc` que permiten implementar funcionalidades TSN.

### Taprio Qdisc

La disciplina `taprio` implementa una versión simplificada del TAS descrito en el estándar IEEE 802.1Qbv. Se acopla a los puertos de salida (*egress ports*) de las interfaces de red. Permite definir la clase de tráfico (*traffic class*, en adelante simplemente clase) asociada a cada trama a partir de la prioridad interna de la misma, especificada en su campo `skb→priority`. Su comportamiento es análogo al definido en el estándar 1Qbv. Precisa por tanto asociar clases a valores del atributo PCP dentro del campo VLAN de la cabecera de la trama. Esta diferencia representa actualmente un impedimento a la hora de integrar `taprio` con Mininet. La Sec. 3.2 describe las opciones y la solución adoptada.

Además de asignar una clase a cada prioridad a cada prioridad, `taprio` requiere:

1. Asignar una cola para las tramas de cada clase.

2. Indicar el instante de inicio de la planificación y el reloj de referencia a utilizar.

3. Inicializar la GCL conforme a la planificación, indicando el tiempo que debe de estar activa cada una de sus entradas.

### CBS Qdisc

La CBS `qdisc` implementa el algoritmo de modulado de tráfico homónimo introducido en la Sec. 2.2.4. En una red TSN es común normalmente ajustar los parámetros del CBS según los requerimientos de ancho de banda de cada clase. Para ello, la CBS `qdisc` se usa en combinación con la `qdisc mqprio`, que permite definir a qué clase de tráfico pertenece una trama dependiendo de la prioridad interna de esta última. Definida la clase a la que pertenece, la trama se envía a la `qdisc` configurada con los parámetros propios de dicha clase.

### ETF Qdisc

`ETF qdisc` proporciona la funcionalidad *Launch Time Control* presente en controladores de red como Intel(R) Ethernet Controller I210© o Intel(R) Ethernet Controller I226©. *Launch Time Control* permite especificar el momento preciso en el que una trama es transmitida, de acuerdo a un reloj de referencia en el interfaz de red.

Las aplicaciones en espacio de usuario suministran este *Launch Time* al kernel como una *timestamp*, utilizando el mecanismo de informacion auxiliar del interfaz de sockets.

El kernel encola y mantiene la trama en la `ETF` hasta que el reloj del sistema alcanza el *timestamp* indicado en la trama para su envío. Este funcionamiento da lugar

a problemas cuando se utiliza en emulación, como se expondrá la Sec. 5.1.2 al abordar la configuración y despliegue de un Caso de Uso.

Esta `qdisc` incluye además un modo estricto, en el que la trama se descarta si se ha superado el *Launch Time*.

#### Netem Qdisc

La Network Emulation (`netem`) `qdisc` ha sido creada con el objetivo de ayudar con el desarrollo de nuevos protocolos de red. No implementa ningún mecanismo TSN, si no que permite emular propiedades características de redes reales tales como la corrupción o duplicación de paquetes, o el tiempo de transmisión.

`netem` puede ser utilizada en combinación con otros componentes TSN como la `taprio` `qdisc` como veremos en la Sec. 5.1.4.

#### Clsact Qdisc

Esta `qdisc` está concebida como clasificador. Permite modificar `skb→priority` en función de filtros de usuario que pueden acceder al valor PCP de las tramas. También permite modificar PCP, lo que permite en un *bridge* alterar la clase a la que pertenece una trama.

### 3.1.2. Recursos de sincronización del tiempo en Linux

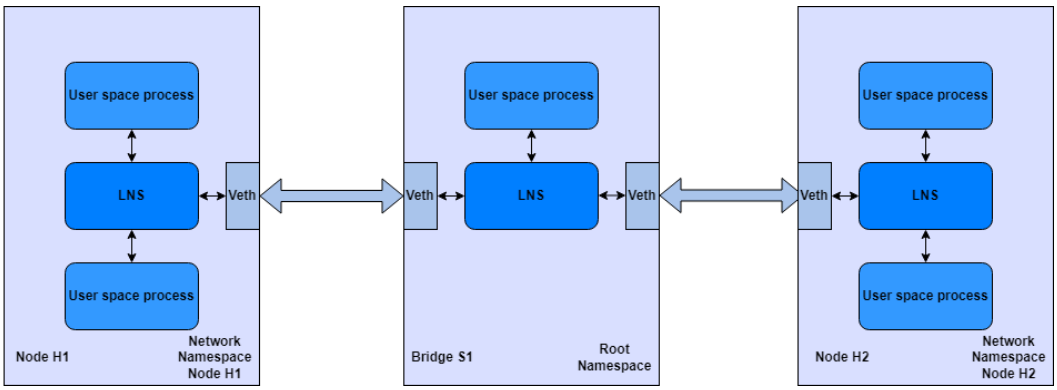
`LinuxPTP` es una *suite* de servicios y utilidades para sincronización temporal basada en IEEE 1588, disponible para entornos Linux. Los más relevantes son `ptp4l` y `phc2sys`.

`ptp4l` proporciona un servicio que implementa diferentes perfiles de PTP, uno de ellos IEEE 802.1AS. Esto permite la sincronización precisa del reloj hardware alojado en los interfaces de red con el GM.

`phc2sys` complementa la funcionalidad de `ptp4l` con un servicio capaz de sincronizar el reloj incorporado en el interfaz de red con los relojes del sistema operativo (e.g. `CLOCK_REALTIME`).

## 3.2. Configuración de Mininet para TSN

Como parte del desarrollo de este TFG se ha implementado una plataforma que emula una red TSN en Mininet, preparada para probar planificaciones sintetizadas para el TAS IEEE 802.1Qbv.




(a) Procesos y *namespaces* en Mininet

```
├─sshd(2133)─┬─sshd(210692)───sshd(210879)───bash(210880)───sudo(211922)───pstree(211923)
│            └─sshd(210887)───sshd(210958)───bash(210959)───sudo(211631)───python3(211632)─┬─bash(211637)───controller(211710)
│                                                                                          ├─H1(211644)───client(211918)  # Proceso cliente que se ejecuta sobre el nodo H1
│                                                                                          ├─H2(211646)
│                                                                                          ├─H3(211648)───server(211915) # Proceso servidor que se ejecuta sobre el nodo H3
│                                                                                          ├─H4(211650)
│                                                                                          ├─S1(211655)───logger_single(211916) # Proceso de registro de medición que se ejecuta sobre S1
│                                                                                          └─S2(211658)───logger_single(211917) # Proceso de registro de medición que se ejecuta sobre S2
```

(b) Procesos de Mininet sobre Linux para una topología con dos *bridges* (S1, S2) interconectados, con dos *hosts* conectados a cada *bridge*

Figura 3.1: Procesos desplegados por Mininet sobre Linux

### 3.2.1. Configuración de la plataforma subyacente

#### Procesos Mininet

Mininet utiliza virtualización basada en procesos para la creación de los nodos. Cada nodo es un proceso Linux diferente. Los procesos de nodos *host* (*end points*) se ejecutan cada uno en un `network namespace` propio. Todos los procesos correspondientes a *bridges* se ejecutan en el `root namespace`. Para interconectar *hosts* y *bridges*, Mininet utiliza pares `veth` (Sec. 2.1.4) que se comportan como un cable Ethernet físico (Fig. 3.1b).

Los procesos de usuario (e.g. emisores y receptores) que se ejecutan sobre los *hosts* se emulan como procesos Linux hijos del proceso *host* sobre el que se ejecutan, y comparten el mismo LNS. La Fig. 3.1ilustra la correspondencia entre componentes emulados, procesos y *namespaces* (a), y el desliegue de procesos en el sistema Linux (b).

La configuración de asignación de estos procesos a núcleos disponibles influye en el cálculo de latencias. Esto se analiza experimentalmente en la Sec. 4.3.3.

#### Modelo de expulsión

El kernel del sistema Linux anfitrión se ha configurado con dos modelos de expulsión diferente, por un lado se ha configurado un *preemptible kernel* sin el parche `PREEMPT_RT`

y otro kernel con el parche `PREEMPT_RT` en modo *fully preemptible kernel (RT)* (ver Sec. 2.1.7), siguiendo lo que parece ser práctica habitual en la industria.

Los programas XDP, que se ejecutan sobre BPF, deberían de estar sometidos al modelo de planificación con el que se ha configurado el kernel. Sin embargo en versiones antiguas de BPF y dependiendo de la versión, se inhibe la expulsión para asegurar que no hay migración y preservar el contenido de mapas temporalmente almacenados como variables `percpu`[33]. Creemos que los posibles efectos laterales son irrelevantes, porque en el curso de las experimentaciones en este TFG hemos observado que los programas XDP tienen latencias muy pequeñas. En todo caso conviene consignarlo, porque podría ser un punto a considerar si en implementaciones reales aparecen efectos laterales inesperados a la hora de realizar mediciones temporales, o se observan desviaciones en el cumplimiento de plazos tiempo real.

### 3.2.2. Emulación del IEEE802.1Qbv TAS

La implementación de un Time Aware Shaper en IEEE 802.1Qbv se fundamenta en la existencia de varias colas para un determinado puerto de red. La emulación se realiza mediante la `taprio qdisc` (Sec. 3.1.1).

La emulación de un *bridge* con Mininet requiere por tanto de múltiples colas para cada uno de sus puertos, que en el caso de las *TX queues* deberán estar sujetas a la planificación de un TAS.

Por lo tanto, el primer paso consiste en configurar `taprio` en los *bridges* de Mininet. Esto representa un primer problema, debido a que Mininet crea por interfaces con una sola cola. Los pares `veth` (Sec. 2.1.4 que Mininet utiliza para implementar los enlaces entre nodos son estructuras del kernel. En consecuencia, ampliar el número de colas de las interfaces supone modificar el kernel. Para ello, en el TFG se ha localizado y utilizado un parche[34] que amplía a ocho las colas de las interfaces de red creadas mediante pares `veth`.

### 3.2.3. Soluciones al problema de etiquetado de clases de tráfico TSN en Mininet

#### Adición del etiquetado VLAN

Los *bridges* reciben tramas Ethernet y utilizan campos como el campo PCP de la cabecera VLAN (Fig. 2.4) para asignar el trafico a una *traffic class*. Las *traffic classes* son despues utilizadas por los algoritmos de conformado y planificacion como IEEE 802.1Qbv.

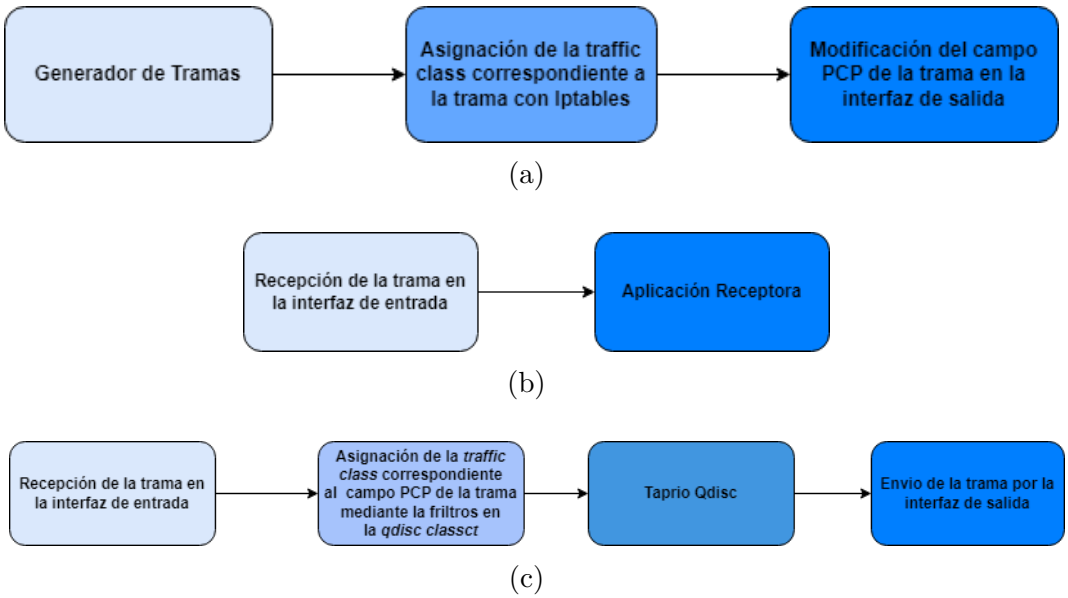



Figura 3.2: Gestión de la identificación de la clase de tráfico de las tramas en *emisores* (a), *receptores* (b) y bridges (c)

Sin embargo, las redes y *bridges* emulados con Mininet carecen tanto de dicha funcionalidad como del propio soporte de Virtual LANs.

Para posibilitar el empleo de `taprio`, se ha ampliado el *framework* Mininet adaptando una nueva clase `Host` con etiquetado VLAN en sus interfaces. De este modo, posibilitamos la emulación del tránsito de tramas entre elementos de red, permitiendo el tratamiento de *traffic classes* descrito anteriormente.

### Configuración de emisores y receptores

Los *emisores* envían las tramas a un puerto del nodo receptor mediante el interfaz de *sockets*. Podrían etiquetar las tramas directamente, con un identificador (`SO_PRIORITY`) según la clase de tráfico a la que corresponda el flujo que emiten. Sin embargo, esto determinaría estáticamente la cola del TAS a la que se dirige el flujo. Por este motivo, es mucho mas útil y flexible que los emisores no asignen estáticamente la prioridad, asociando después cada puerto a una clase TSN (prioridad según subcampo PCP) en tiempo de configuración, con posibilidad de hacer cambios dinámicos.

La solución que hemos adoptado, adaptada al caso de emulación sobre Mininet, se basa en asociar a cada puerto su prioridad mediante un filtro aplicado con la herramienta `iptables` de Linux. Estos filtros almacenan dicha prioridad en el campo `skb→priority` (Sec. 2.1.3) de las tramas que se envían a través de ese puerto. A su vez, las interfaces VLAN implementan un mecanismo que traslada este valor al campo PCP. Las Figs. 3.2 (a) y (b) esquematizan este procedimiento.

#### Configuración de los *bridges*

Ya hemos señalado que Mininet ignora el campo VLAN. En consecuencia, los *bridges* creados en Mininet carecen de interfaz VLAN. Es decir, sus interfaces no acceden ni interpretan dicho campo, reenviando las tramas según su dirección de destino. En consecuencia, no existe el mecanismo al que nos hemos referido en los *hosts* en donde se ejecutan emisores y receptores, realice la copia o conversión `skb→priority` ↔ PCP. Es necesario realizar la conversión manualmente.

En nuestro caso, interesa poder encolar cada trama cuando llega a `taprio`, en la cola que le corresponda según su clase, pero `taprio` no permite realizar esta asignación según el valor PCP sino según el campo `skb→priority` (ver Sec. 3.1.1). Por lo tanto es necesario modificar el campo `skb→priority` de cada trama antes de que llegue a `taprio`. La opción más viable consiste en crear un filtro en la entrada de las tramas al kernel que inicialice `skb→priority` en función del PCP de la trama.

Para crear el filtro se consideró `iptables` en primer lugar, pero tras examinar a fondo la herramienta se determinó que no permite acceder al campo PCP de la trama. Se consideró también `ebtables`, que permitiría marcar la trama en función del PCP para luego modificar `skb→priority` mediante otro filtro en `iptables`, pero no fue posible utilizar esta herramienta por problemas de compatibilidad con el kernel, modificado para adecuarlo a otras necesidades de este trabajo.

Se consideró diseñar un filtro específico y añadirlo al kernel, pero se consideró preferible buscar herramientas ya existentes que faciliten la compatibilidad y la migración.

Por ello se configuró un filtro XDP para modificar un campo de la cabecera IP de la trama según el valor de PCP, y otro filtro posterior en `iptables` que modificar `skb→priority` en consecuencia. El mecanismo como tal funciona, pero se desestimó finalmente por ser excesivamente intrusivo y potencialmente lento, al requerir la intervención varias capas de red.

Finalmente se estudió y recurrió a la `clsact qdisc` por las características que hemos descrito en la Sec. 3.1.1. Mediante `tc`, se crea y configura `clsact` como etapa previa a la `qdisc taprio` (Fig. 3.2 (c)).

Con esta mejora se amplía de manera considerable la capacidad de Mininet de emular una red con *bridges* implementando IEEE 802.1Qbv.

### 3.2.4. Sincronización del tiempo en Mininet

En una red TSN física es necesario contar con el protocolo gPTP para mantener sincronizados los relojes de los nodos (Sec. 2.2.3).

En el caso de Mininet, todos los nodos se ejecutan (se emulan) como procesos

sobre un mismo kernel, compartiendo el mismo reloj del sistema. Sin embargo, debe considerarse la influencia de dos factores que pueden distorsionar las medidas.

El primero es el reloj del sistema operativo utilizado para muestrear los eventos. Por ejemplo, en el caso de `CLOCK_REALTIME`, el sistema operativo no garantiza la ausencia de ajustes, incluso si ello supone la pérdida de la monotonicidad. Al utilizar la escala UTC, también puede estar sometido a la introducción de *leap seconds* para compensar la ralentización de la rotación de la Tierra. `CLOCK_MONOTONIC` proporciona más garantías y es la elección habitual en sistemas TR, pero sigue sujeto a potenciales ajustes de la frecuencia. Para eludir estos problemas, en este TFG se garantiza la ausencia de interferencias tales como cambios manuales por parte del administrador, o la sincronización del tiempo del sistema operativo utilizando NTP o PTP, mediante la desactivación de estos mecanismos. .

El segundo factor a considerar es el hardware subyacente. Desde el punto de vista del hardware, los diferentes procesos involucrados pueden ejecutarse en diferentes núcleos (*cores*), y en un caso general, incluso en diferentes *sockets*. Por lo tanto, la arquitectura hardware influye de manera determinante en el alineamiento de las mediciones de procesos que no se ejecutan en el mismo núcleo. En el caso del hardware utilizado con soporte Intel Time Coordinated Computing (TCC), la implementación del Time Stamp Counter (TSC) en el MPSoC garantiza que las medidas temporales registradas en diferentes núcleos no presenten divergencias.

En el Cap. 4 se analizan experimentalmente los efectos de utilizar en Mininet como emulador TSN unos u otros relojes en el registro de tiempos mediante diferentes métodos.

# Capítulo 4

# Metodología de cálculo de latencias sobre Mininet

Un objetivo importante de este TFG es establecer una metodología fiable de medida de tiempos en TSN sobre Mininet (Sec. 1.2). Para afrontar dicho objetivo ha sido necesario estudiar a fondo toda la LNS, los recursos e interfaces disponibles en Mininet, y los posibles sistemas de registro de tiempos (*timestamping*) de paso tramas utilizados en los diferentes niveles.

## 4.1. Entorno experimental

Hemos instalado Mininet y realizado la experimentación sobre tres plataformas hardware con distintas capacidades, sobre las que se han aplicado diferentes optimizaciones para TR (Tab. 4.1). La distribución Linux es en todo los casos Ubuntu 20.04 TLS, kernel 5.2.21. Las modificaciones y configuraciones necesarias para emular redes TSN con Mininet son las expuestas en la Sec. 3.2. Cada nodo de la red en Mininet corresponde a un proceso ejecutado sobre el sistema operativo anfitrión (Sec. 3.2.1), por lo que hemos minimizado el posible impacto del resto de los procesos en el registro de tiempos aislando cada proceso en un núcleo diferente de la CPU.

La Fig. 4.1 ilustra la topología de red TSN emulada en este TFG a efectos experimentales. La red se compone de dos *bridges* interconectados (S1 y S2), con cuatro *end points* (H1-H4). H1 y H4 están conectados al *bridge* S1; H3 y H4 están conectados al *bridge* S2. Para poder medir latencias se ha instalado en los *bridges* una `qdisc taprio` con una configuración especial que mantiene todas las colas abiertas para que las tramas no sean bloqueadas por la `qdisc`.

| Configuración | CPU | Optimización TR | Modelo de expulsión | Resultados |
|---|---|---|---|---|
| CONF-1 | Intel® Xeon® Gold 5120 CPU @ 2.20GHz 56 núcleos (core *Skylake*) | No | *preemptible kernel* | Sec. 4.3.1 |
| CONF-2 | Intel® Xeon® Gold 5120 CPU @ 2.20GHz 56 núcleos (core *Skylake*) | Software | PREEMPT_RT *full preemption* | Sec. 4.3.2 |
| CONF-3 | 11th Gen Intel® Core™ i7-1185GRE CPU @ 2.80GHz 4 núcleos (core *Tigerlake*) | TCC nativo | PREEMPT_RT *full preemption* | Sec. 4.3.3 Anexo C |

Tabla 4.1: Plataformas hardware usadas en la experimentación. La distribución Linux es en todo los casos Ubuntu 20.04 TLS con kernel 5.2.21 (misma versión, en su caso, del parche `PREEMPT_RT`


Figura 4.1: Infraestructura de red TSN emulada sobre Mininet en el TFG.

## 4.2. Metodología de registro de tiempos

Se ha comenzado caracterizando el comportamiento temporal de los elementos que forman la red. Como primer paso, se han identificado los puntos del sistema que permiten obtener información temporal de las tramas.

Se ha determinado que en una red emulada en Mininet solamente se generan *timestamps* en el momento de recepción de una trama [30]. Estos *timestamps* emulan los generados en una red física por el kernel de un nodo cuando llega una nueva trama a través del NIC. Además de estos *timestamps*, se pueden obtener también el instante de tiempo en los que una trama es enviada desde una proceso de usuario en el nodo emisor, y el instante en el que ha sido recibida por un proceso de usuario en el nodo receptor.

### 4.2.1. Relojes del sistema

Los *timestamps* se generan a partir de los relojes del sistema de cada nodo. En el caso de Mininet, todos los nodos se emulan mediante procesos que se ejecutan en la misma máquina (Sec. 3.1b). A la hora de comparar dos *timestamps*, es necesario asegurarse de que ambos han sido generados a partir del mismo reloj. Tras revisar los diferentes temporizadores se ha considerado que los más adecuados para los objetivos de medición que se persiguen son los siguientes:

1. `CLOCK_REALTIME`: Reloj de referencia en Linux. Marca el instante de tiempo con precisión de nanosegundos a partir del 1 de enero de 1970. Se ajusta para mantenerlo sincronizado con el tiempo UTC, incluyendo los segundos intercalares. Se utiliza comúnmente para obtener la hora y la fecha actuales del sistema.

2. `CLOCK_MONOTONIC`: Contador monótono creciente. Marca el tiempo transcurrido desde algún punto fijo en el pasado, generalmente los nanosegundos transcurridos desde el arranque del sistema. En consecuencia no puede ser modificado directamente, a diferencia de el reloj `CLOCK_REALTIME`, aunque puede estar sujeto a cambios en la frecuencia. Por este motivo es un contador muy utilizado en Linux para temporización de tareas con restricciones TR.

La ISA puede proporcionar mecanismos más precisos para la medición del tiempo. Un ejemplo es el acceso directo al *Time Stamp Counter* (TSC) a través de instrucciones como `rdtsc` y `rdtscp` de la ISA Intel x86, utilizada por algunos kernel de Linux (no todos) como la base de su infraestructura de temporización [1]. Este acceso de bajo nivel proporciona medidas de *ticks*, que posteriormente son ajustadas para obtener picosegundos o nanosegundos. Estos procedimientos sin embargo tienen dos inconvenientes. En primer lugar, son más precisos de lo necesario para lo que se persigue en nuestro contexto. En segundo lugar, supone realizar modificaciones en el kernel, que introducen nuevas sobrecargas según el nivel desde el que se accedan (e.g. desde el espacio de usuario). Por ello, para este TFG se ha considerado más apropiado utilizar los relojes y llamadas al sistema ofrecidas actualmente por Linux.

| Etiqueta | Significado | Espacio |
|---|---|---|
| T1 | Tiempo en el que el proceso cliente que se ejecuta sobre el nodo emisor da la orden al kernel de enviar una trama. | U |
| T2 | Tiempo de recepción de la trama en la NIC (`veth`) de S1 | K |
| T3 | Tiempo de recepción de la trama en la NIC (`veth`) de S2 | K |
| T4 | Tiempo de recepción de la trama en l la NIC (`veth`) de H3 | K |
| T5 | Tiempo en el que el proceso que actúa como proceso servidor ejecutado sobre el receptor recibe una trama a través de un *socket*. | U |

Tabla 4.2: Características de los marcadores temporales usados para cálculo de latencias en los diferentes experimentos. U/K: lectura en espacio de usuario / de kernel.

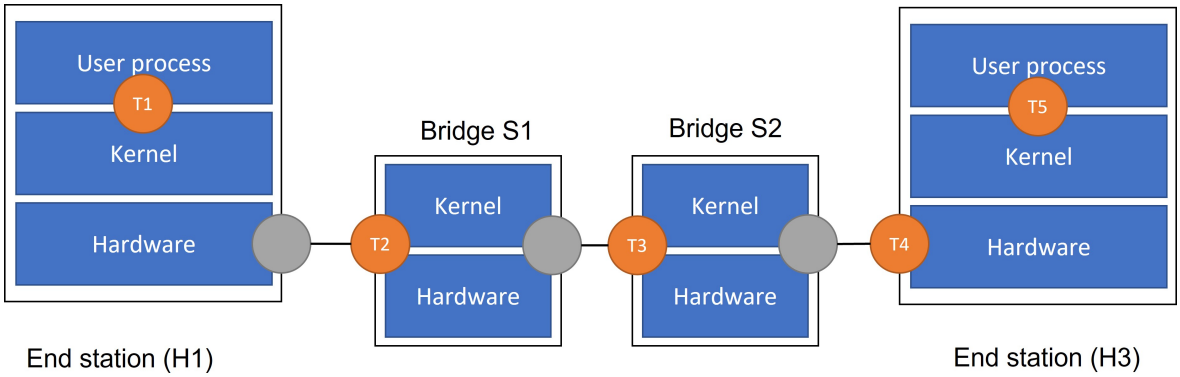



Figura 4.2: Puntos de registro de tiempos (Tab. 4.2)

### 4.2.2. Definición de tiempos registrados y latencias calculadas

La Tab. 4.2 resume los marcadores registrados en la experimentación de este TFG, representados en la Fig. 4.2. Los relojes y método de registro usados se explicitan en cada tipo de experimento. A partir de estos tiempos se han calculado las siguientes latencias:

- **Latencia *end-to-end*** ($e2e$) - Tiempo transcurrido desde que un proceso cliente ejecuta la llamada al sistema correspondiente al envío de una trama hasta que dicha trama es recibida por un proceso que actúa como servidor en un nodo receptor. Se calcula como $e2e = T5 - T1$.

- **Latencia *end-to-end.nic*** ($e2e.nic$) - tiempo desde que las tramas son enviadas hasta que son recibidas en el NIC del nodo destino. $e2e.nic = T4 - T1$.

[1] `rdtsc` ha sido la opción habitual para contar ciclos para medidas de rendimiento y no está afectado por cambios de frecuencia introducidos por mecanismos de ahorro energético basados en DVFS. Sin embargo, en procesadores multicore requiere el uso de barreras de sincronización, ya que se ejecuta fuera de orden, un problema especialmente en multicores. `rdtscp` aparece en CPUs más recientes de Intel y actúa en sí misma como barrera de sincronización pero parece ser menos eficiente que `rdtsc` usada con barreras de sincronización

– **Latencia de envío** ($sendL$) - Tiempo transcurrido desde que un proceso da la orden de transmitir una trama (T1), hasta que dicha trama es enviada por el nodo. En nuestra plataforma podemos considerar propagación y transmisión instantáneas[2], de modo que tiempo efectivo de envío y tiempo de recepción en S1 (T2) pueden considerarse iguales. De este modo $sendL = T2 - T1$.

– **Tiempo de cómputo en *bridges*** (latencia de los *bridges*, $brL$) - Tiempo empleado en el procesado de una trama en el *bridge*, es decir, desde que la trama llega (completa) al puerto de entrada del *bridge* hasta que se inserta en la cola de salida. Se ha calculado como $brL = T3 - T2$ $(= T4 - T3)$.

– **Latencia en la recepción de una trama** ($arrL$) - Tiempo desde que una trama es recibida por el kernel hasta que dicha trama es visible desde un proceso de usuario que se ejecuta sobre el nodo. Se ha calculado como $arrL = T5 - T4$. Esta latencia depende de la implementación de la aplicación de usuario que se ejecuta en el nodo receptor. La hemos caracterizado por completitud, pero no se tiene en cuenta a la hora de realizar una planificación TSN y queda fuera del estándar. Por estos motivos en el Cap.5 utilizamos la latencia $e2e.nic$ como referencia y no la latencia $e2e$.

### 4.2.3. Registro en el espacio del kernel

#### Registro de tiempos en los *Virtual Ethernet Pairs*

Como indicábamos en la Sec. 4.2, el kernel de Linux genera un *timestamp* basado en `CLOCK_REALTIME` en el momento de enviar una trama a través de un par `veth`, almacenándolo en la entrada del `skb` asociada a esa trama. Este *timestamp* representa el tiempo de llegada de una trama a un nodo. Para acceder a estos *timestamps* desde el espacio de usuario es necesario utilizar la interfaz de *sockets* de Linux.

En los nodos receptores es posible registrar el *timestamp* en las propias aplicaciones que actúan como receptoras (*listeners*) de un flujo), mediante un *socket* IP (`AF_INET`). Utilizamos UDP como protocolo de capa de transporte [3] a fin de que cada paquete enviado por el *socket* corresponda a una trama en la red. Aunque TSN se localiza en el nivel de red (DLL), es viable usar UDP (nivel de transporte) debido a que estamos emulando el comportamiento (respuesta) del sistema, y la sobrecarga introducida puede considerarse despreciable.

[2]Esta premisa es habitual en trabajos relacionados con planificación en TSN, e.g. [35], [1]

[3]UDP es un protocolo sin conexión, y carece de procesado de comprobación y corrección de errores. En consecuencia es mucho más ligero que TCP, y es la opción preferida en entornos TR en los que es preferible descartar tramas que esperar a su reenvío

En los *bridges* no hay emisores ni receptores por definición, y es preciso añadir un proceso instrumental para realizar el registro. Para ello recurrimos a un *socket* `AF_PACKET` asociado a cada interfaz [30], que da acceso a las tramas de la capa 2 (nivel del driver), fechadas por el propio kernel (interfaz `veth`) mediante `CLOCK_RT`. Esto limita la medición basada en `AF_PACKET` al uso de dicho contador, sin otra posibilidad.

La Fig. 4.3 (b) muestra el punto de registro de los *timestamps* en los *bridges* mediante `AF_PACKET`, considerando que el tiempo avanza de arriba hacia abajo. El esquema muestra que la validación para comprobar que los *timestamps* pertenecen a la trama que quiere ser analizada queda incluida en el cálculo de $brL$.

#### Registro mediante XDP

XDP (Sec. 2.1.6), a diferencia de `AF_PACKET`, puede registrar *timestamps* utilizando cualquier reloj del sistema. En esta caracterización utilizamos preferentemente `CLOCK_MONOTONIC` por los motivos expuestos en la Sec. 4.2.1. XDP registra el valor mediante la función `bpf_ktime_get_ns()` del sistema BPF (Sec. 2.1.5). Se puede acceder a este *timestamp* desde un proceso de usuario a través de un mapa BPF. Este procedimiento permite realizar un *profiling* de los *bridges* minimizando la sobrecarga. La Fig. 4.3 (a) muestra el esquema de registro. Lo llevamos a cabo creando dos programas XDP. El primero verifica si la trama pertenece a uno de los flujos a analizar, y a continuación obtiene el *timestamp*. El segundo obtiene el *timestamp* justo en la recepción del paquete (el programa XDP se ejecuta justo en el retorno de la rutina de servicio a interrupción, Sec. 2.1.6), y a continuación verifica la trama. A diferencia del registro mediante `AF_PACKET` (Fig. 4.3 (b)), el tiempo dedicado a validación queda excluido del cálculo de $brL$. Es decir, experimentalmente cabe esperar valores menores de tiempo de cómputo en el *bridge* ($brL$) respecto a la medición vía `AF_PACKET`, cuando se caracteriza un *bridge* aislado.

Sin embargo, en el caso de querer analizar el comportamiento temporal de varios *bridges* consecutivos en una única prueba, no es posible ignorar el tiempo de cómputo de la solución XDP. Como se puede deducir de la misma Fig.4.3 (a), la $brL$ del segundo *bridge* (*Siguiente Nodo*) incluirá el tiempo en el que incurre XDP para la validación de la trama.

### 4.2.4. Registro en espacio de usuario

Los procesos de usuario pueden leer cualquier reloj del sistema, pasando el identificador del mismo como parámetro de la llamada al sistema `clock_gettime()`.

Hemos usado esta llamada para registrar el *timestamp* de transferencia de una trama entre el espacio de usuario ($T1$) y el de kernel ($T1'$), sea al enviarla o recibirla, y

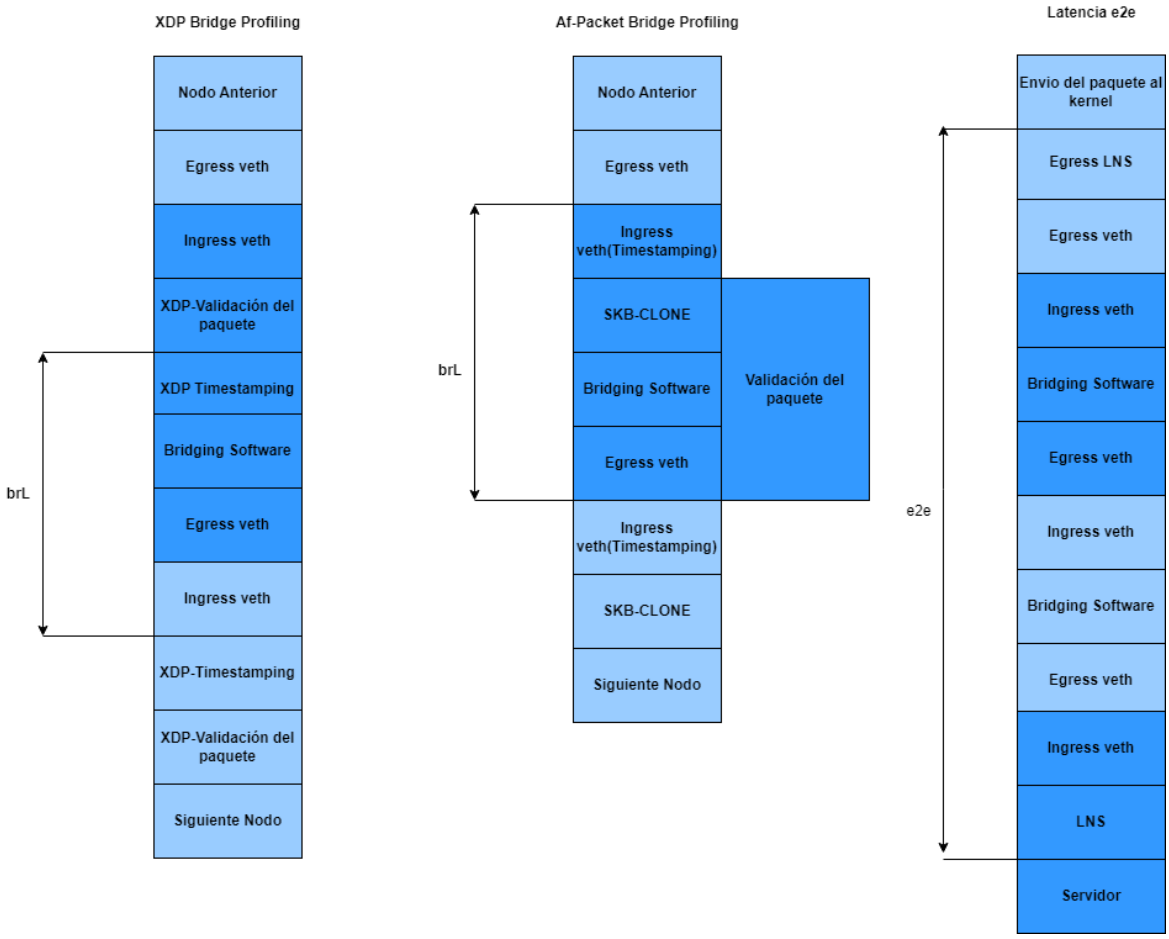



Figura 4.3: Registro de tiempos y caracterización de tiempo de cómputo en bridges ($brL$) mediante: (a) XDP sobre `CLOCK_MONOTONIC` y (b) `AF_PACKET` sobre `CLOCK_RT`. En (c), registro de tiempos y cálculo de $e2e$ desde espacio de usuario (llamada al sistema `clock_gettime(CLOCK_MONOTONIC)`. El tiempo avanza de arriba hacia abajo. Cada sombreado corresponde a un nodo. Por ejemplo *nodo anterior* y *egress veth* corresponden al primer nodo del esquema.

también para el cálculo de la latencia $e2e$. La Fig. 4.3 (c) esquematiza el *span* temporal del $e2e$ calculado con este método, en un caso en el que no se estarían registrando tiempos de paso por *bridges* con ningún método (XDP ó `AF_PACKET`).

### 4.2.5. Análisis experimental de los métodos de registro de tiempos

Vamos a realizar una experimentación preliminar para estimar la hipótesis de que la metodología basada en XDP permite calcular la latencia del *bridge* ($brL$) con un impacto menor que la metodología basada en `AF_PACKET`.

Se han generado 1000 tramas desde el nodo H1 al nodo H3 (Fig. 4.1). Según los resultados de laFig. 4.4, el registro de tiempos mediante XDP tiene efectivamente menos impacto en $brL$ que la solución a través de `AF_PACKET`.

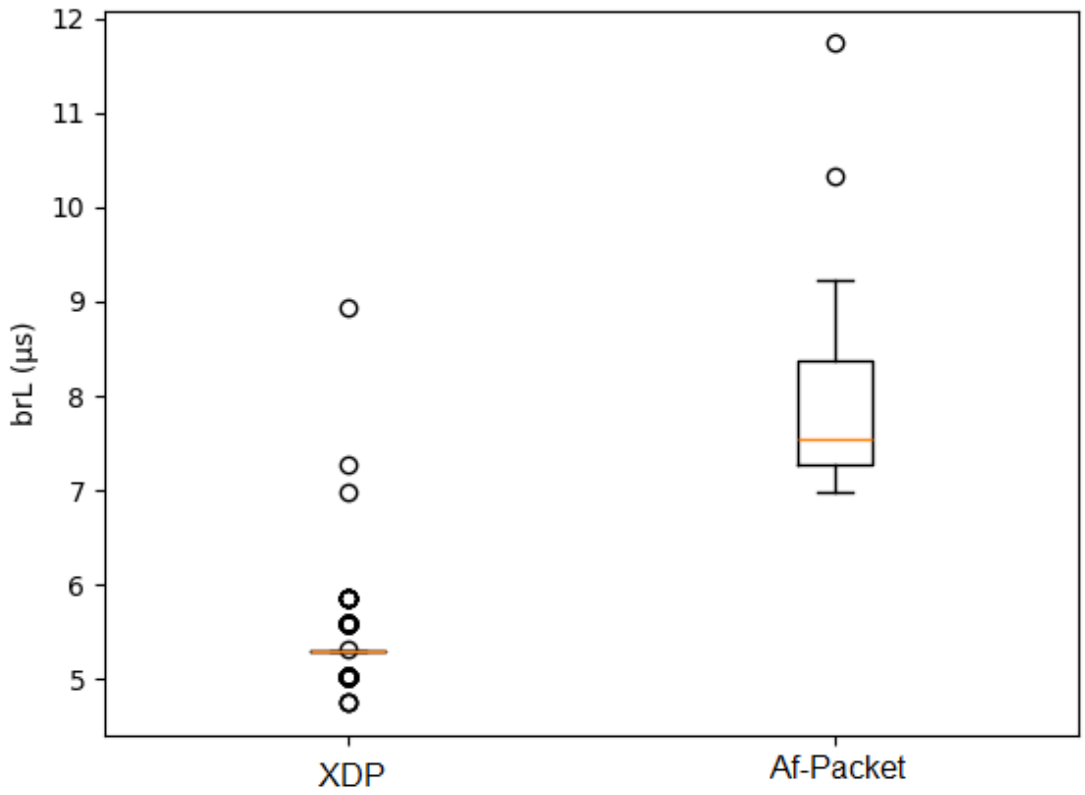



Figura 4.4: Comparación del impacto de registro de tiempos sobre un elemento mediante XDP y `AF_PACKET` en el cálculo de latencia de *bridges* ($brL$, Sec. 4.2.2).

En una segunda comparación, estudiamos también el impacto de ambos métodos en la latencia $e2e.nic$. Recordemos que esta latencia es importante porque es la que generalmente se pretende optimizar en los problemas de planificación TSN. Se han realizado tres pruebas, cuyos resultados se recogen en la Fig. 4.5. En la primera prueba (*Default*) no se han activado mecanismos de *profiling*, de modo que no se registran los *timestamps* T2 Y T3 (Sec. 4.2.2). Refleja unas condiciones similares al del cálculo de $e2e$ en la Fig. 4.3 (c), pero excluyendo $arrL$. En la segunda y tercera pruebas se realiza el registro respectivamente mediante XDP y `AF_PACKET`. El eje vertical indica la latencia $e2e.nic$ en $\mu$s.

XDP resulta en un *jitter* superior, menos valores atípicos y menor $e2e.nic$ respecto a `AF_PACKET`. Con todo, estas diferencias entre ambos métodos son en la práctica despreciables, debido al mayor impacto que tienen otros elementos en el el registro de tiempos de paso de tramas y en el cálculo de $e2e.nic$, como veremos en la Sec. 5.1.2.

A la vista de estos resultados, para realizar el *profiling* de *bridges* en la Sec. 4.3 se ha utilizado el método basado en XDP, debido a su menor impacto en $brL$ respecto a `AF_PACKET` (Fig. 4.4). Sin embargo, para emular el Caso de Uso TSN en el Cap.5 se

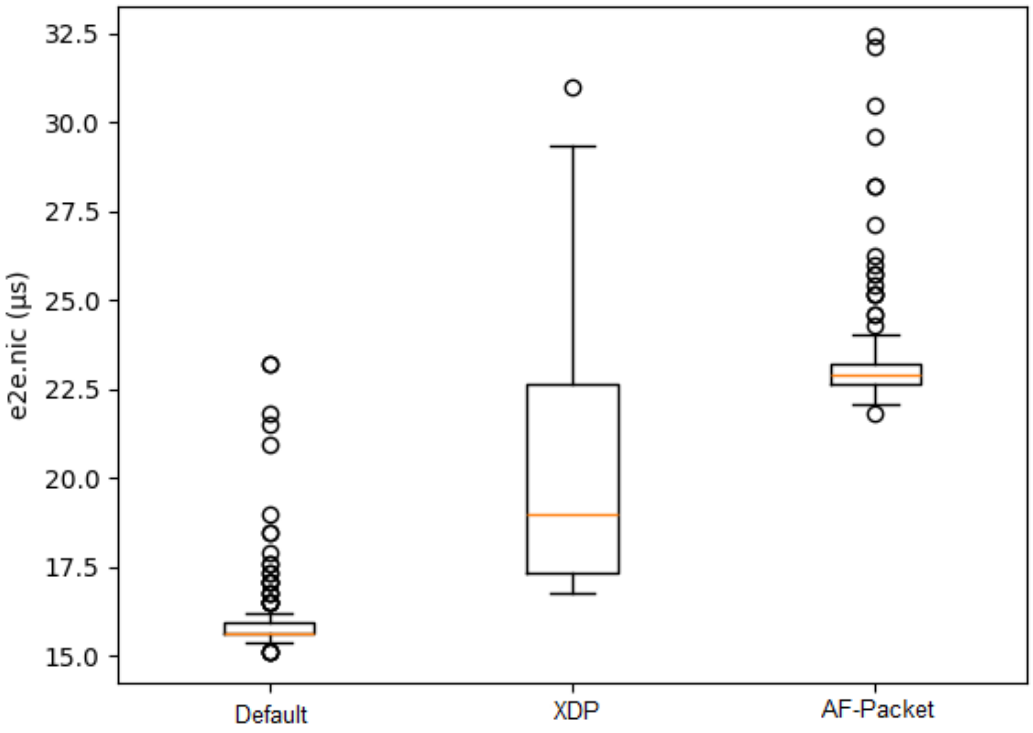



Figura 4.5: Comparación de resultados de cálculo de *e2e.nic* a partir del registro de tiempos en tramas mediante XDP y `AF_PACKET`. El caso *Default* se añade como referencia (ver texto).

ha optado por el método basado en `AF_PACKET`, debido a que en este caso se pretende caracterizar ambos *bridges* en el mismo experimento además de medir *e2e.nic*. Por otra parte, el método basado en `AF_PACKET` es más sencillo de utilizar (no requiere la realización e inyección de programas XDP), y también más flexible a la hora de analizar distintos flujos.

## 4.3. Resultados del cálculo de latencias en las tres plataformas

### 4.3.1. Resultados CONF-1

La Fig. 4.6 muestra (de izquierda a derecha) las latencias $sendL$, $brL$, $arrL$ y $e2e$ calculadas sobre sobre un kernel en configuración CONF-1 (Tab. 4.1). La mayor parte de la latencia punto a punto viene dado por el tiempo que tarda el kernel en enviar una trama desde que es recibida por el receptor hasta que llega a la aplicación de usuario. El tiempo de cómputo de los *bridges* es bastante más reducido en comparación con el

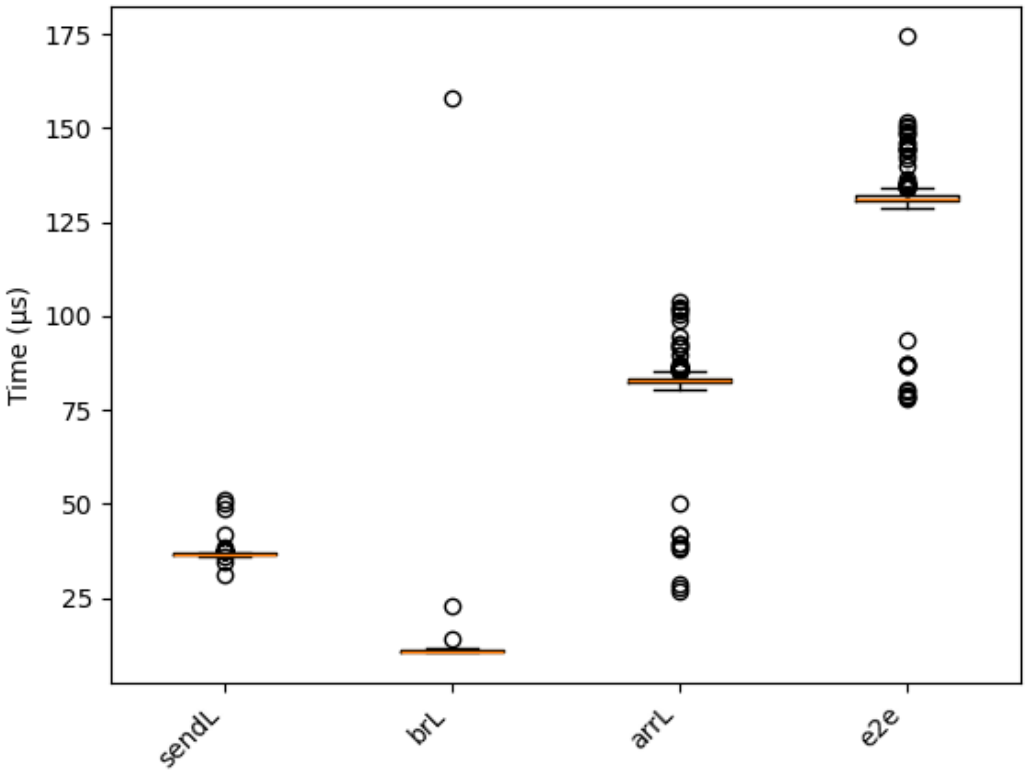



Figura 4.6: Comparación de latencias. Configuración CONF-1 (Tab. 4.1)

retardo causado por los receptores y los emisores. Aún así, el *jitter* es muy elevado, con una latencia que varia entre 60 $\mu$s (mejor caso) y 250 $\mu$s (peor caso).

### 4.3.2. Resultados CONF-2

Para intentar reducir el *jitter* de la metodología de registro se han seguido las sugerencias de Intel para la ejecución de aplicaciones tiempo real sobre procesadores Intel de propósito general. No existen en el caso de la microarquitectura Intel core *Skylake*, por lo que se han seguido las sugerencias para la arquitectura Intel core *Tigerlake*[36]:

- Desactivación de todos los estados de ahorro de energía de la CPU.
- Utilización en el kernel de un reloj específico de la CPU llamado *time stamp counter* como reloj de sistema.
- Desactivación del *watchdog* y del controlador encargado de manejar la frecuencia de las CPUs.
- Configuración del kernel para que, en caso de entrar en estado *idle*, la CPU permanezca en estado activo.

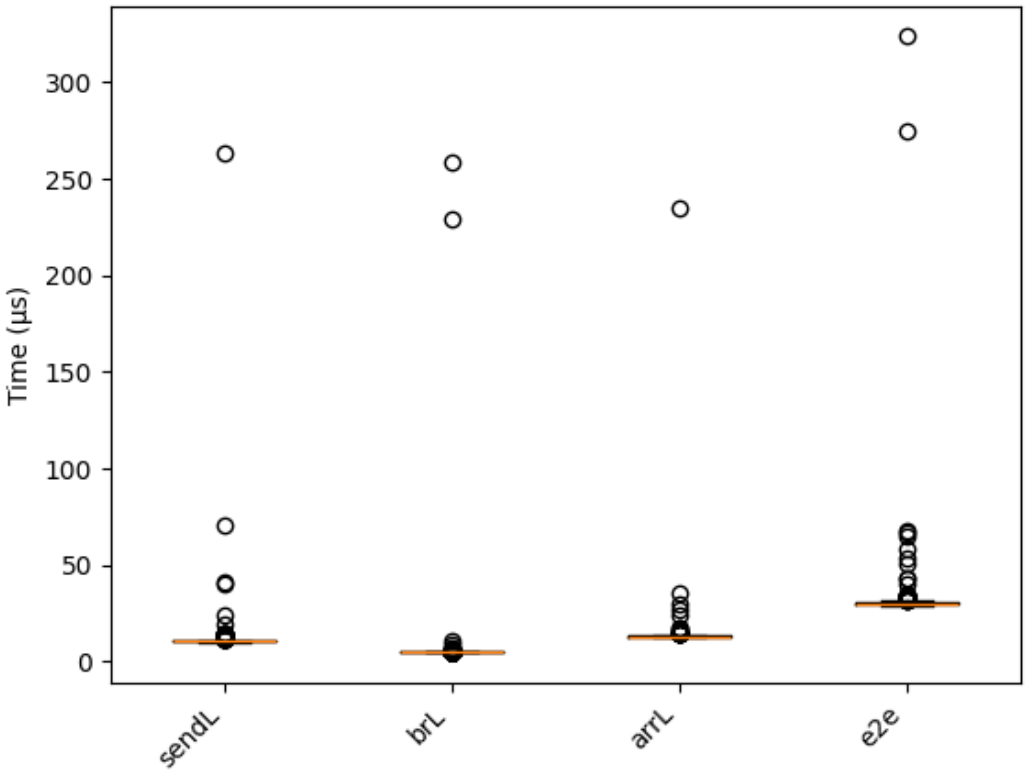



Figura 4.7: Comparación de latencias. Configuración CONF-2 (Tab. 4.1)

Adicionalmente, se han aplicado otras dos optimizaciones relativas a Mininet:

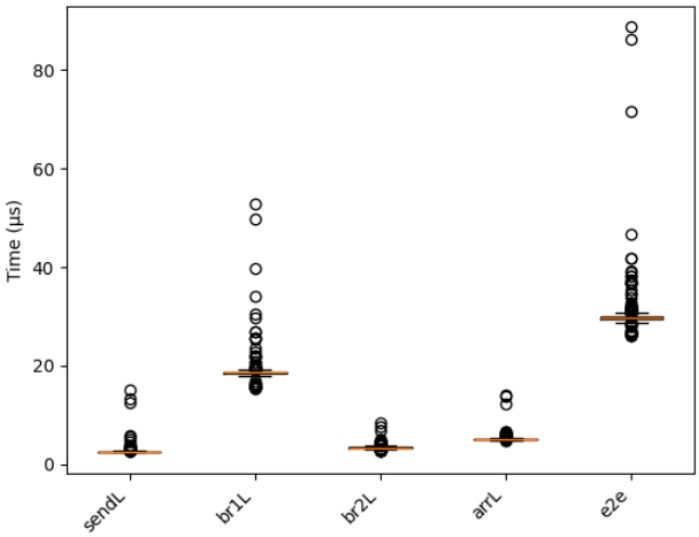
- Aislamiento de los núcleos que se asignarán a los procesos de Mininet del planificador de tareas, de forma que el *scheduler* no pueda ejecutar procesos sobre dichas CPUs.

- Parametrización del planificador del kernel para que ejecute las rutinas de servicio a interrupción sobre núcleos determinados, i.e. CPUs no asignadas a ningún proceso Mininet.

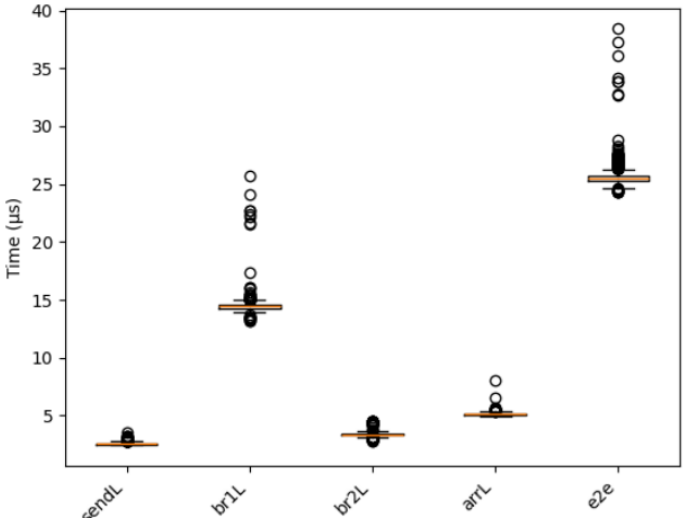
Gracias a estas modificaciones se reducen latencia y *jitter* medio (Fig. 4.7), aunque siguen presentes los valores atípicos que aparecían en la plataforma sin optimización (Fig. 4.6).

### 4.3.3. Resultados CONF-3

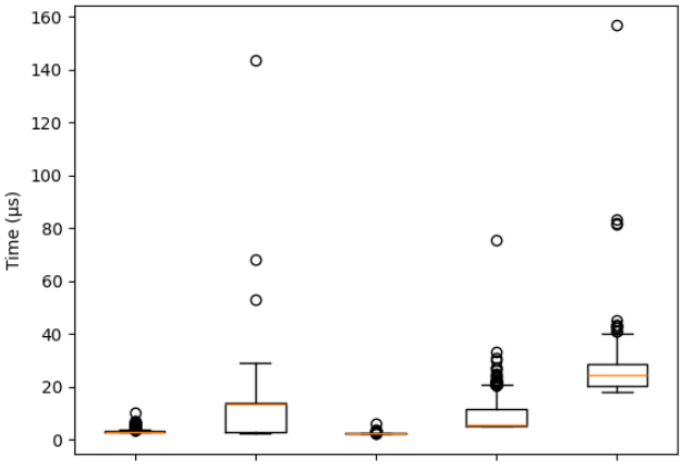
Con el objetivo de eliminar, o al menos reducir, los valores atípicos encontrados en la Sec. 4.3.1, hemos podido experimentar también sobre un PC industrial IEI DRPC-240-TGL Fanless (Anexo C), que incorpora Intel TCC configurable por BIOS (CONF-3,




(a) Resultados CONF-3: asignación permutada 1




(b) Resultados CONF-3: asignación permutada 2




(c) Resultados CONF-3: asignación permutada 3

Figura 4.8: Resultados con tres permutaciones en la asignación de procesos a los cuatro núcleos disponibles en CONF-3

Tab. 4.1), optimizado para procesos TR. Ha habido que renunciar a algunas características del sistema a fin de poder utilizar en lo posible las mismas configuraciones de Mininet, recursos TSN, distribución de Linux, kernel y parche `PREEMPT_RT` preparados en este TFG.

Se realizado el *profiling* con la metodología basada en `AF_PACKET`, ya que es la utilizada en el despliegue de un caso de uso (Cap. 5), en el que el número de procesos desplegados sobre la plataforma es menor que el número de núcleos disponibles, afectando al resultado de la plataforma.

Se han asignado tres de los cuatro núcleos a los procesos de la emulación. El cuarto se utiliza para el resto de los procesos del sistema.

CONF-3 reduce notablemente los atípicos, pero el resultado y las latencias varían notablemente, dependiendo de la asignación de procesos a núcleos que se realice. Por ejemplo, en las Figs. 4.8a y 4.8b los tiempos de cómputo de los *bridge* S1 y S2 son diferentes entre sí. En la Fig. 4.8a la latencia $e2e$ en la Fig. 4.8a es mayor que la de la Fig. 4.8b.

Por otra parte, en la Fig. 4.8c los *bridges* S1 Y S2 tienen el mismo tiempo de cómputo, sin embargo $arrL$ es muy dispar, obteniendo un *jitter* superior al resto de asignaciones. Las distintas permutaciones en la asignación de procesos a núcleos se han realizado de manera aleatoria.

# Capítulo 5

# Emulación de un Caso de Uso

El despliegue y emulación de un Caso de Uso abordado en este capítulo ha resultado imprescindible para comprobar la congruencia y eficacia del sistema de emulación y metodología de medida expuestas en capítulos anteriores, cumpliendo así los objetivos específicos de este TFG (Sec. 1.2).

Este despliegue ha sido útil, además, para revelar algunas cuestiones adicionales que no habían aparecido en el análisis y experimentación metodológicas efectuadas en el Cap. 4, y que aparecen al realizar la configuración y puesta en marcha de un sistema para un Caso de Uso determinado.

En este TFG la configuración de la topología, *bridges*, configuración de la planificación y puesta en marcha del sistema emulado sobre Mininet se realiza mediante *scripts* propios ((Sec. 5.1). Esto nos permite aprender de abajo arriba detalles que afectan a la metodología de medida de tiempos, y tener un mayor control sobre los procedimientos.

Seguidamente, en la Sec. 5.2, definimos el Caso de Uso, desplegado sobre la infraestructura de red emulada en el Cap. 4 para llevar a cabo la experimentación metodológica (Fig. 4.1).

## 5.1. Configuración y despliegue del sistema TSN

### 5.1.1. Instante cero

El despliegue de un Caso de Uso implica definir el momento en el que, configurados todos nodos, topología, y planificación, comienzan a circular los flujos TSN de modo que el envío de tramas se realice en los instantes que les corresponde. Son parte de los aspectos determinados por las recomendaciones IEEE 802.1Q para la gestión de recursos (ver Sec. 2.2.5). Este *instante cero* de puesta en marcha es utilizado tanto por los clientes emisores como por las `taprio qdisc` situadas en los *bridges*.

Hemos creado un *script* en Python que, a través de la API de Mininet, genera en primer lugar la red, define a continuación el *timestamp* que se va a tomar como

*instante cero*, y lanza finalmente todos los procesos del sistema. Estos procesos quedan en estado de espera hasta que se alcanza el tiempo definido como *instante cero*.

### 5.1.2. Real-Time Client

En una plataforma física existen mecanismos que permiten enviar paquetes al medio de transmisión en un instante preciso. Un ejemplo es el `LaunchTime` presente en la NIC Intel(R) Ethernet Controller I210©. Linux ofrece la `ETF qdisc` (Sec. 3.1.1) para explotar dicha funcionalidad hardware, pero la hemos descartado como solución en emulación porque su implementación software añade un *jitter* considerable.

Para conseguir una precisión aceptable en emulación, se ha implementado un cliente que genera tramas UDP dirigidas a un puerto determinado. El envío de tramas se realiza siguiendo un modelo típico de activación periódica de tareas tiempo real, tomando como referencia el *instante cero* (Sec. 5.1.1). A partir de dicho instante, se obtienen los instantes en los que deben de ser transmitidos el resto de los paquetes.

Para poder realizar una planificación correcta de los flujos TSN es necesario identificar la variación entre los instantes en los que las tramas deben enviarse y el instante en el que realmente se envían. El instante de tiempo T1' es el momento en el que el planificador indica que la trama debe de ser enviada por el nodo emisor, mientras que el instante en el que se envía la trama viene dado por T1. En la Fig. 5.1 se observa que ambos instantes (T1 y T1') pueden llegar a diferir entre 60 y 80 $\mu$s para todas las tramas, en una plataforma con optimización TR moderada como CONF-2 (Tab. 4.1). Es preciso considerar este retardo a la hora de caracterizar el sistema para parametrizar el problema de planificación de modo que su solución sea correcta, no sólo analíticamente (lo cual que depende del método de cálculo) sino una vez trasladada la planificación a la plataforma emulada mediante inicialización de las `taprio` (Sec. 5.2.2).

### 5.1.3. Emulación de tiempos de transmisión

El tiempo de transmisión es un parámetro utilizado en planificación de flujos TSN. Se define como el tiempo que tarda en enviarse la trama al medio físico por el que va a ser transmitido. En una una red TSN física, el TAS tiene que mantener abierta la puerta que corresponda no más tarde del instante en el que la trama empieza a ser enviada al medio físico hasta no antes del instante en el que se haya transmitido el último bit de la misma (*ventana de apertura / transmisión*). En nuestro caso no existe un medio físico, ya que trabajamos sobre una red emulada. Por ello hemos ponderado maneras de emular dicho comportamiento utilizando las `qdisc` de linux, optando por utilizar `netem` (Sec. 3.1.1) como clase hija de `taprio`.

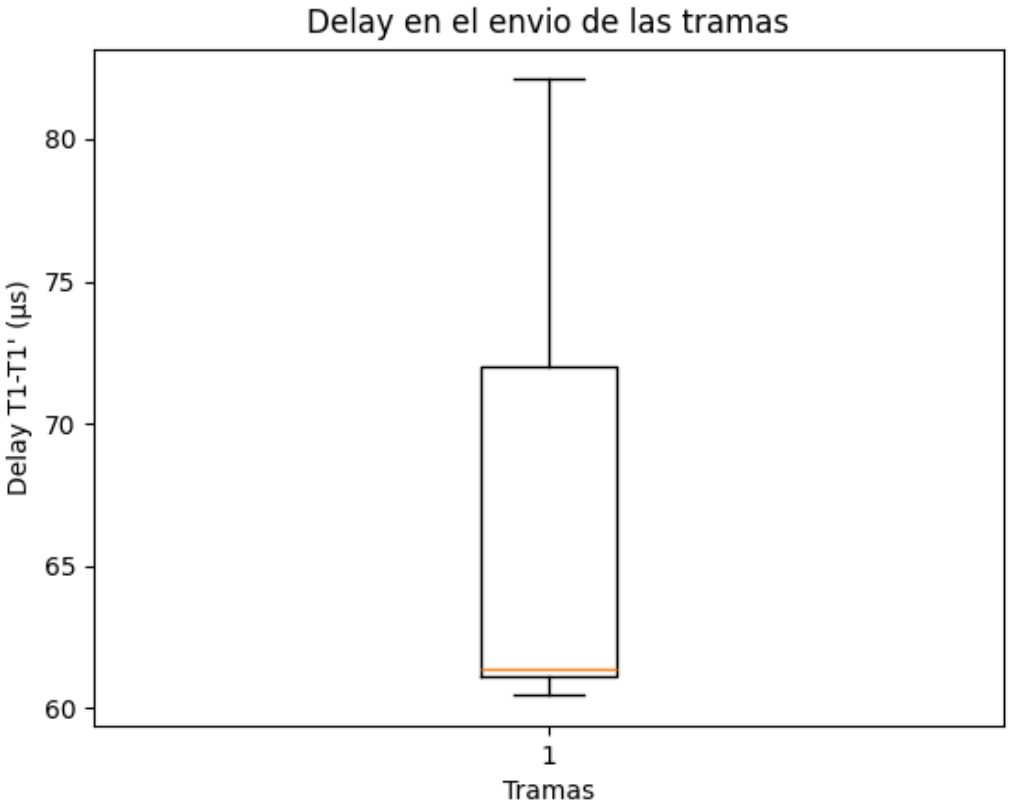



Figura 5.1: Retardo en el envío de tramas (configuración CONF-2 (Tab. 4.1).

La `qdisc taprio` está implementada de forma que la planificación se aplica la salida de la clase hija, en nuestro caso `netem`. Podemos por tanto emular el tiempo de transmisión colocando una `netem` por cola del TAS, i.e. como clase hija de `taprio`, y añadiendo a cada una de esas `netem` dicho tiempo de transmisión. De esta forma, una trama que se encuentre dentro de la `qdisc netem` solo podrá ser transmitida en el caso de que la `qdisc taprio` permita su transmisión (intervalo de transmisión establecido en la GCL según la planificación), que es nuestro objetivo. La Fig. 5.2 ilustra la esta estructura de `qdisc` para emula un TAS.

Esta implementación conlleva el problema de que la primera trama que entra a cada una de las `netem qdisc` que se encuentran como salida de las `taprio` desaparece. Por ejemplo, en el caso del flujo 0, la trama número 1 desaparecerá en el *bridge* S1, y la trama número dos desaparecerá en el *bridge* S2. No se ha encontrado la causa de este problema, por lo que se ha solucionado considerando el primer hiperperiodo como fase transitoria, comenzando el registro de tiempos a partir del segundo.

### 5.1.4. Emulación del tiempo de propagación

En redes de computadores se entiende por *tiempo de propagación* ($T_p$) el tiempo que tarda una señal en llegar del emisor al receptor. Puede calcularse como $T_p = L_l/V_F$, donde $L_l$ y $V_F$ son respectivamente la longitud y el Factor de Velocidad del enlace [1].

La emulación de tiempos de propagación no puede realizarse basándonos en la solución adoptada para los tiempos de transmisión, es decir, añadiendo una nueva `netem` que añada un tiempo. Pese a que la `netem` sea una *classful* `qdisc` (i.e. que puede tener clases hijas), la `taprio` realizaría el *shaping* sobre las tramas con el tiempo de transmisión ya añadido. Es decir, las tramas estarían listas para transmitir fuera de la ventana de apertura de puerta determinada por la planificación.

La solución requiere una modificación del kernel. Como se expuso en la Sec. 3.2.3, se ha decidido en este TFG atenerse a parches, herramientas e interfaces ya existentes. Por otra parte, se trata de un tiempo despreciable en el contexto de los casos de uso industriales que son prioritariamente objetivo del TFG (Sec. 1.2).

## 5.2. Definición del Caso de Uso

### 5.2.1. Flujos

La Tab. 5.1 muestra los flujos desplegados sobre la infraestructura de red de la Fig. 4.1 emulada en Mininet.

### 5.2.2. Planificación TSN de los flujos

La planificación de estos flujos sobre la topología objetivo se ha realizado mediante el método propuesto en [1], basado en un PPLM preacondicionado mediante una heurística. Los parámetros necesarios son los siguientes:

- Flujos : Definición de los distintos flujos TSN a emular sobre la plataforma. Los parámetros a definir para cada flujo son los siguientes:
  - Tiempo de transmisión
  - Nodo Origen
  - Nodo Destino

[1] El Factor de Velocidad ( *Velocity Factor* es una razón relativa a la velocidad de la luz, que depende del medio y tipo de señal. En el vacío por ejemplo es 1, en aire 0.999 y en enlaces de cobre varía según sus características (e.g. 0.5 - 0.8). El concepto de señal a transmitir varía también según el medio. En cobre por ejemplo se refiere al periodo de variación de la tensión, y en fibra óptica a un pulso de luz. Conviene tener en cuenta que dicha variación no se corresponde directamente con un bit. Por ejemplo en Fast Ethernet (modelo 4B5B) se codifican 4 bits cada 5 pulsos de tensión. Pueden ampliarse estos conceptos en textos generales de redes y comunicaciones como [37] y [12].

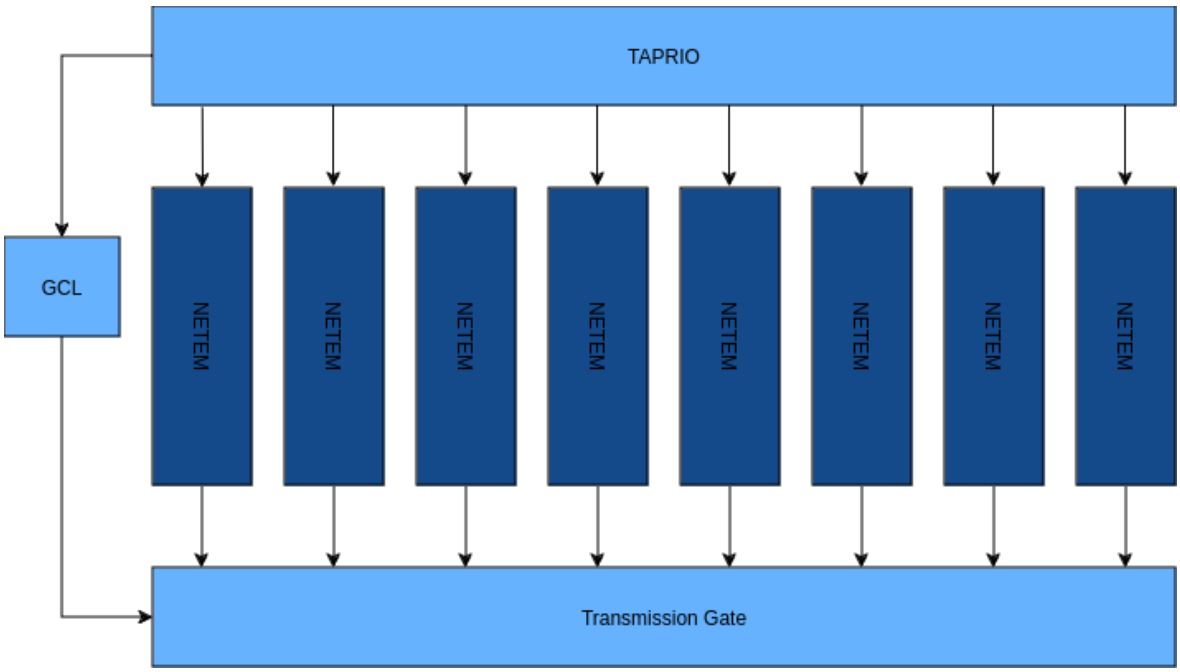



Figura 5.2: Disposición de `taprio` y `netem` para la emulación del tiempo de transmisión

| Flujo | Talker | Listener | Periodo | Deadline |
|---|---|---|---|---|
| 0 | h1 | h3 | 10 ms | 10 ms |
| 1 | h2 | h3 | 20 ms | 20 ms |
| 2 | h4 | h3 | 30 ms | 30 ms |

Tabla 5.1: Flujos del Use Case

- *bridges* a atravesar

– **Tiempo de propagación** (Sec. 5.1.4)

– **Tiempo de cómputo**: Tiempo necesario para procesar una trama en el *bridge*: desde el puerto de entrada hasta la cola de salida.

– **Desfase de reloj**: Máximo desfase que puede tener el reloj de un nodo con respecto al reloj maestro.

Mediante estos parámetros la herramienta formula las restricciones del PPLM que utilizamos [1]), lo resuelve y genera las entradas de las GCL para los TAS de cada *bridge* (Sec. 2.2.4). En nuestro caso, emulamos los TAS mediante la `taprio qdisc`. El *script* de Python ya mencionado en la Sec. 5.1.1 también inicializa las `taprio` a partir de las GCL generadas por el planificador.

### Parametrización del planificador

Se ha decidido considerar un tiempo de transmisión de 100 $\mu$s, basado en consulta de casos. Como tiempo de cómputo, tomamos un $brL$ de 10$\mu$s, a partir de la caracterización

realizada con la metodología expuesta en la Sec. 4.3.

El desfase de reloj es uno de los factores que contribuyen al *jitter* en cualquier plataforma[2], junto a las características de las NIC (`veth` en nuestro caso) y *bridges* entre otros. En una red en Mininet, el desfase de reloj es cero debido a que todos los nodos comparten el mismo reloj. Pero no los otros factores del *jitter* estructural de la plataforma (Sec. 4.3). Por ello, las ventanas de apertura calculadas en la planificación pueden resultar insuficientes.

Por ejemplo, si una trama tiene que ser enviada en el instante 100, un planificador agnóstico respecto al *jitter* generará la entrada GCL de modo que la puerta del TAS se abra en el instante 100 y se cierre en 100 + el tiempo de propagación. Un planificador con *jitter* (desfase de reloj más otros factores) igual a 10 por ejemplo, fijará la apertura de la puerta del TAS en el instante 90 ($= 100 - 10$) y el cierre en el instante 100 + tiempo de propagación + 10. La ventana de apertura será más amplia, y permitirá aceptar tramas con *jitter* de 10 unidades de tiempo.

El planificador que utilizamos [1], en una práctica muy habitual en este tipo de planificadores, no considera el *jitter* estructural como parámetro, sino únicamente el desfase de reloj (cero en Mininet)[3]. Por este motivo, utilizamos el parámetro *desfase de reloj* para considerar el *jitter* no nulo de nuestra plataforma de emulación.

Los valores atípicos observados en la Sec. 4.3 se encuentran alrededor de los 200 $\mu$s por encima de la media aproximadamente (Fig.4.7). Además, observábamos 80 $\mu$s adicionales de retraso en el envío de las tramas ($T1-T1'+$sendL, Fig.5.1) y otros 10 $\mu$s adicionales en caso de que estén activados los mecanismos de *profiling* mencionados en la Sec.4.2.2 para registro de tiempos de paso de tramas por *bridges*. Con estos datos, de manera conservadora, podemos estimar un *jitter* conjunto de 500 $\mu$s. Este es el valor que hemos trasladado como parámetro de desfase de reloj al planificados, a fin de que las aperturas y cierres generadas sean suficientemente amplias para el paso de tramas con *jitter*.

Finalmente, conviene observar que en un sistema físico, además de las tramas de los flujos TSN, circulan tramas necesarias para la configuración de la red, como las del protocolo *spaning tree* de los *bridges* o las de ARP. Estas tramas no existen en el sistema que emulamos.

[2]Podríamos denominarlo *jitter estructural*, para diferenciarlo del *jitter* introducido por otras capas (que no se consideran en TSN) y del *jitter* de planificación, que los planificadores típicamente intentan minimizar pero difícilmente eliminan completamente

[3]En parte o en todo, porque una vez desplegado un caso de uso con su planificación es preciso verificar y a menudo reajustar dicha planificación. La planificación dinámica, o los ajustes dinámicos de una planificación, es un problema abierto y una activa línea de investigación actualmente en TSN.

## 5.3. Resultados experimentales

Para verificar el correcto funcionamiento del sistema, especialmente que las tramas de red se envían en los instantes indicados por la planificación, se han utilizado las herramientas desarrolladas en el Cap. 4.

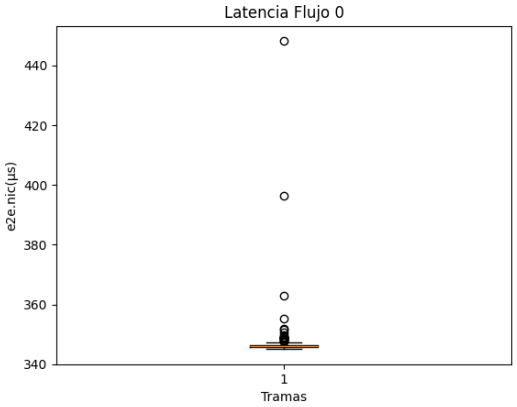
Una vez definida y configurada la planificación en la `taprio`, se han generado tramas en los instantes indicados por el planificador que emulan los flujos con el *Real-Time Client* desarrollado 5.1.2. Se han verificado los *timestamps* generados por las tramas asociadas a los flujos para calcular la latencia punto a punto $e2e.nic$ (Sec. 4.2.2), que es la que se considera en TSN para fijar el cumplimiento de los *deadlines* requeridos por los flujos (Tab. 5.1), y cuya optimización es precisamente objeto de los planificadores TSN. La medida de esta $e2e.nic$ se realiza con el método basado en `AF_PACKET` por las razones que se expusieron en la discusión metodológica (Sec. 4.2.5), y con el registro de tiempos activado en los *bridges* (a fin de poder observar el cumplimiento de tiempos de paso de tramas). Esto último implica que los valores de $e2e.nic$ obtenidos son ligeramente más altos que los que se obtendrían sin la sobrecarga de ese *profiling* en los *bridges* (cabe recordar aquí la Fig. 4.4).

Las gráficas de la Fig. 5.3 permiten comprobar que las latencias $e2e.nic$ de los flujos planificados no superan en ningún caso el *deadline* definido para cada uno de los flujos (Tab. 5.1). Por ejemplo, en el caso de la Fig. 5.3a, no hay ninguna latencia superior a los 10 ms, y por lo tanto todas las tramas de dicho flujo cumplen con las restricciones temporales.Los valores de $e2e.nic$ atípicos que aparecen son los esperables, porque se sitúa dentro del *jitter* estimado en base a la caracterización del mismo en la plataforma de emulación.

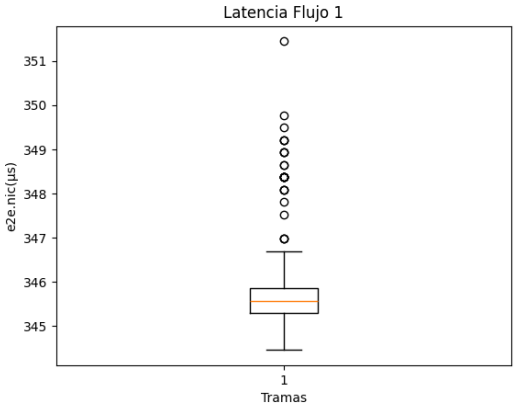
Las Figs. 5.4 y 5.5 visualizan que las tramas atraviesan las ventanas de apertura de los TAS (`taprio`) de los *bridges* conforme a la planificación aplicada. Se muestra para cada *bridge* la gráfica con las ventanas de apertura de sus puertas para cada flujo (segmentos coloreados de las Figs. 5.4a y 5.5a), y la gráfica con los intervalos en los que realmente las tramas atraviesan dichas puertas (segmentos coloreados de las Figs. 5.4b y 5.5b). Visualmente es fácil comprobar que los segmentos de paso de trama son menores (y se sitúan dentro de) los segmentos de apertura de puerta.

## 5.4. Consideraciones finales

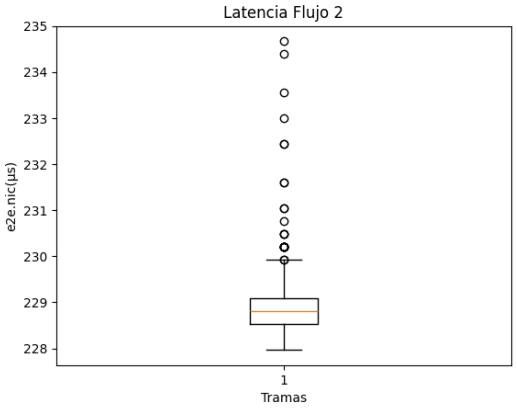
El objetivo de este capítulo ha sido comprobar que el sistema emulado, configurado conforme a lo expuesto en el Cap. 3, y caracterizado conforme a la metodología establecida en el Cap. 4, permite comprobar el funcionamiento de una planificación del




(a) Latencia de las tramas asociadas al Flujo 0

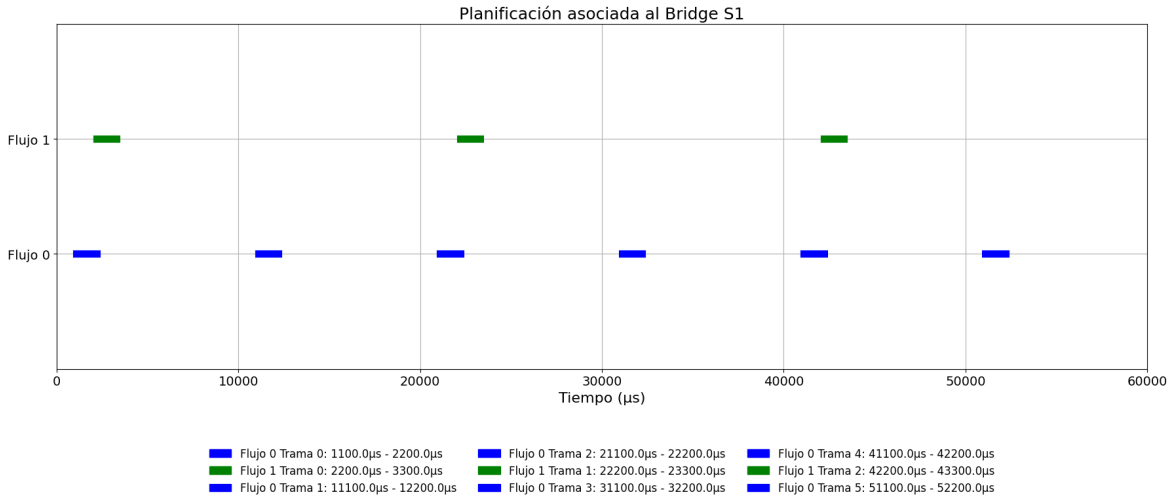



(b) Latencia de las tramas asociadas al Flujo 1

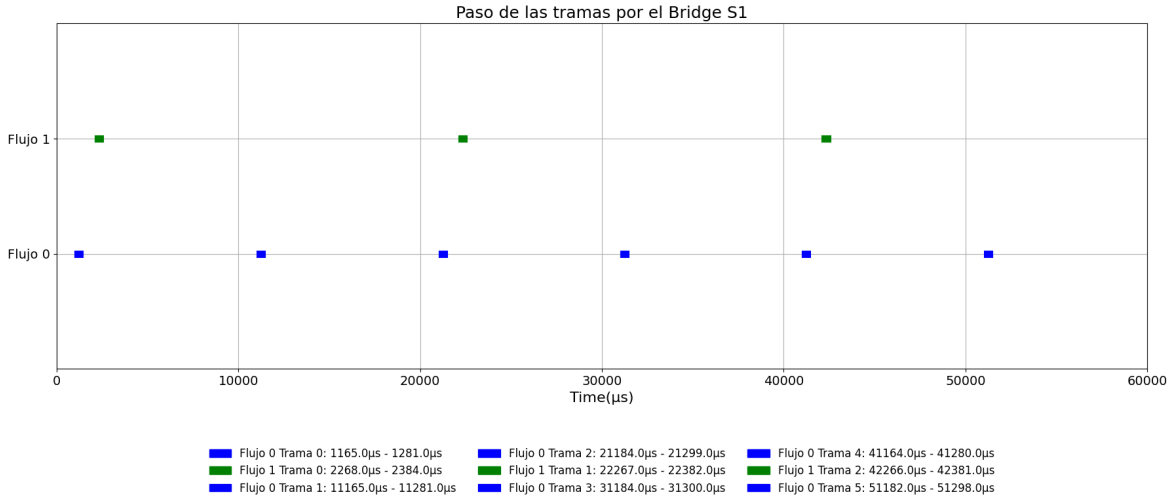



(c) Latencia de las tramas asociadas al Flujo 2

Figura 5.3: Latencia de las tramas asociadas a diferentes flujos

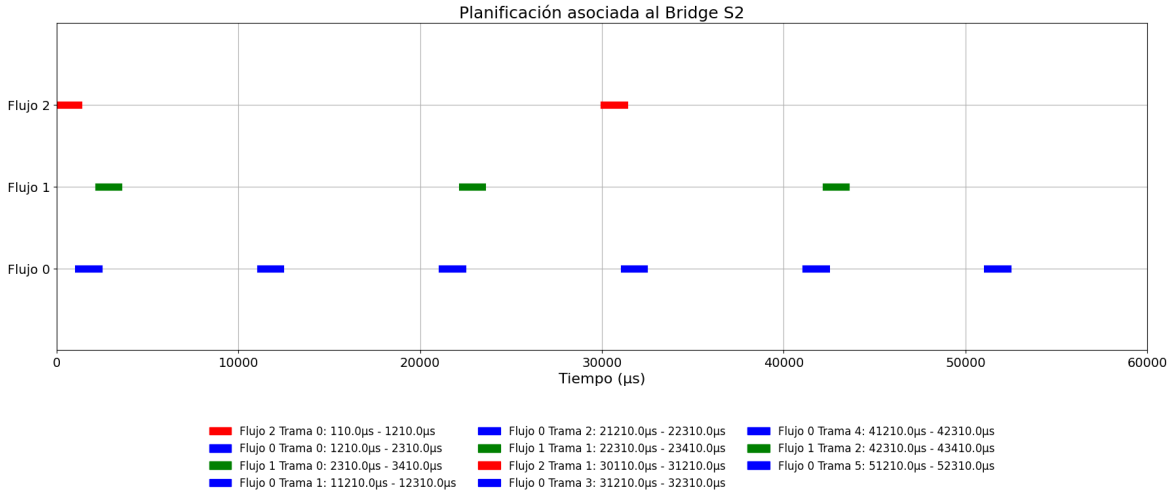



(a) Planificación asociada al *bridge* S1




(b) Paso de las tramas por el *bridge* S1

Figura 5.4: Planificación y Paso de tramas asociadas al *bridge* S1




(a) Planificación asociada al *bridge* S2


(b) Paso de las tramas por el *bridge* S2

Figura 5.5: Planificación y Paso de tramas asociadas al *bridge* S2

tipo de la propuesta en [1].

En la Sec. 2.2.5 se introdujo el *modus operandi* en un despliegue real. Se proporciona un sistema de gestión (*control plane*) la definición de flujos, topología de red (el propio sistema la puede descubrir), tipo de planificación o posibles parámetros determinados de antemano. El sistema configura la red, calcula la planificación y la verifica. Si es correcta, se pone en marcha el sistema (o se añaden nuevos flujos al sistema en marcha). Si no pasa la verificación, se recalibra y se calcula una nueva planificación.

De forma similar a dicho *modus operandi*, en caso de utilizar la plataforma de emulación y metodología expuestas en este TFG para casos de uso, correspondería repetir las emulaciones ajustando por ejemplo el parámetro de *jitter*, para estimar o ajustar la capacidad del sistema. Algo que puede ser útil como paso previo a trasladar el caso de uso a un *testbed* o a un sistema definitivo.

# Capítulo 6

# Conclusiones y líneas abiertas

## 6.1. Discusión de resultados experimentales

### 6.1.1. Métodos de registro de tiempos

El método de registro basado en llamada al sistema es una elección sencilla y suficiente para el cálculo de la latencia de extremo a extremo (*end to end*, $e2e$), especialmente porque en Mininet la sincronización temporal de los contadores está asegurada al tratarse de procesos ejecutados sobre un mismo sistema operativo y hardware. El recurso a un contador u otro (`CLOCK_MONOTONIC` vs. `CLOCK_RT`) no parece de influencia relevante, aunque `CLOCK_MONOTONIC` es la opción lógica en un sistema con restricciones TR.

Se han evaluado dos métodos de registro de tiempo para caracterización de *bridges*, respectivamente basados en XDP y `AF_PACKET`, y se concluye que ambos pueden ser utilizados con este fin. Los resultados experimentales indican que el método implementado con XDP puede proporcionar una cota más ajustada del tiempo de cómputo de *bridges* ($brL$, Fig.4.4) que la solución basada en `AF_PACKET`. En todo caso la diferencia entre ambos métodos es de unos pocos $\mu$s, y en el caso de utilizarse para medir la latencia $e2e.nic$, que es la considerada en planificación TSN, las diferencias pueden considerarse negligibles, siendo `AF_PACKET` más sencillo y flexible en su uso. Por ese motivo, la comprobación de la planificación en el Caso de Uso del Cap. 5 se ha realizado mediante `AF_PACKET`, tanto para registrar tiempos para el cálculo de la $e2e.nic$ como para registrar simultáneamente tiempos de paso de tramas por las puertas de los TAS (`taprio`) en *bridges*.

### 6.1.2. Influencia de la plataforma subyacente

Los resultados obtenidos muestran que el despliegue de la plataforma de emulación sobre una CPU con gran número de núcleos (56 núcleos, CONF-1 y CONF-2) per-

mite latencias relativamente ajustadas, a costa de un considerable aumento del *jitter* (Figs. 4.6 y 4.7).

La utilización de un PC industrial optimizado para tiempo real mediante el sistema Intel TCC (Anexo C) permite disminuir claramente los valores atípicos y la latencia $e2e.nic$, pero los resultados varían según la asignación de procesos Mininet a los cuatro núcleos disponibles (Fig. 4.8). Este tipo de plataformas, con Intel TCC disponible, son conservadoras hoy día en el número de núcleos (4 - 12 hasta donde hemos podido comprobar) y memoria, debido particularmente a compromisos coste - rendimiento. Esto obliga a compartir núcleos para varios procesos Mininet, que no pueden ejecutarse aislados como en CONF-1 y CONF-2. La asignación de más de un proceso de emulación por núcleo y consiguiente falta de aislamiento provoca variaciones en el resultado de los cambios de contexto, y en el comportamiento de los mecanismos especulativos (p.ej. predicción de saltos) y de ocultación de latencia (prebúsqueda, comportamiento de la jerarquía de memoria en general). Ello provoca por ejemplo conflictos diferentes en el acceso a memoria principal en cada caso. Ello influye en el registro de tiempos y, por ende, en las latencias y el *jitter* finalmente calculados. De usarse como plataforma de emulación, supone caracterizar el comportamiento de Mininet para cada Caso de Uso y ensayar permutaciones en la asignación de procesos a núcleos hasta encontrar latencias satisfactorias.

El mejor de los escenarios consiste obviamente en disponer de un multicore que soporte el sistema Intel TCC nativo, y que además cuente con un número de núcleos que minimice el número de procesos por núcleo, a fin de aislar cada uno de los procesos de emulación. Posiblemente no tiene mucho sentido para optimizar la emulación de sistemas TSN mediante Mininet, pero puede tenerlo en el caso de TSN como SDN (*cloudification*).

La conclusión más importante es que una planificación será siempre correcta, con independencia de la plataforma de emulación, siempre que se caractericen correctamente las latencias y se parametrice el planificador coherentemente.

## 6.2. Mininet como plataforma de emulación de sistemas TSN

Mininet ha resultado ser una plataforma satisfactoria para emular casos de uso TSN, pese a que no se ha diseñado con este objetivo. El trabajo desarrollado en este TFG muestra que es posible, con algunas limitaciones, crear una plataforma para comprobar resultados de planificación, uno de los objetivos de la línea de investigación en la que se enmarca el trabajo, siempre que se tengan en cuenta ciertos aspectos.

Una de las principales limitaciones de la plataforma viene dada por la necesidad de disponer de una versión del kernel de Linux compatible con el parche específico que permite integrar la `taprio` en Mininet. Esto impide a fecha de escritura de esta memoria la utilización de mecanismos implementados en versiones más recientes del kernel de Linux. Una de estas funcionalidades es, por ejemplo, la combinación de la `taprio qdisc` con la `ETF qdisc`, que permite reducir el *jitter* de los *bridges*. Otra importante es el uso de *xdp-hints*, que permitiría desarrollar un nuevo mecanismo de registro de tiempos. Estas nuevas versiones también permitirían utilizar herramientas de *testing* de redes TSN como Isochron.

El esfuerzo de comprobación de un planificador sobre un caso de uso (Cap. 5), mediante la configuración Mininet/Linux discutida en el Cap. 3 y la caracterización y metodología expuestas en el Cap. 4 ha evidenciado aspectos interesantes relativos a la configuración y puesta en marcha del sistema en emulación, como la consideración de un *instante cero* y de una estrategia de inicio del sistema. Más importante, si cabe, ha sido la parametrización correcta del algoritmo de planificación utilizado (un sistema propuesto en el grupo de investigación, basado en formular y solucionar un PPLM [1]. En particular, la estimación del *jitter* en base a la caracterización realizada en el Cap. 4.

## 6.3. Líneas abiertas

Enumeramos algunas de las cuestiones abiertas en el curso del TFG, que juzgamos interesantes para trabajos futuros.

En relación a metodología de medida y caracterización, creemos que hay dos aspectos poco desarrollados y que van más allá del problema específico de emulación:

- Estudiar el uso de la instrucciones `rdtsc` y `rdtscp` de Intel, cuyo rendimiento y comportamiento pueden actualmente variar según la plataforma.
- Caracterización de factores específicos que contribuyen al *jitter*.

Aspectos más específicos ligados a Mininet como plataforma de emulación serían los siguientes:

- Diseño de un parche del kernel para emular el tiempo de transmisión
- Actualización del parche de las `veth` para que permita integrar utilidades TSN actuales como la `taprio qdisc` junto a la `ETF qdisc`.
- Desarrollo o adaptación de un *framework* que permita realizar despliegues automáticos sobre la plataforma.

– Comparación de resultados emulación / *testbedding* basados en la metodología planteada.

– Realización de pruebas sobre un plataformas con Intel TCC y un mayor número de núcleos.

# Bibliografía


[1] Alitzel G. Torres-Macías, Juan Segarra, José L. Briz, Antonio Ramírez-Treviño, and Héctor Blanco-Alcaine. Fast ieee802.1qbv gate scheduling through integer linear programming. *IEEE Access*, pages 1–1, 2024.

[2] A. Gracia, J. L. Briz, H Blanco-Alcaine, Juan Segarra, A. Torres, and A. Ramírez-Treviño. Cracking down overheads in tsn emulation over mininet, 2024. Stuttgart (Germany), 1-3 Oct. 2024.

[3] A.G. Torres-Macías, A. Ramirez-Treviño, J.L. Briz, J. Segarra, and H. Blanco-Alcaine. Modeling time-sensitive networking using timed continuous petri nets. *IFAC-PapersOnLine*, 58(1):300–305, 2024. 17th IFAC Workshop on discrete Event Systems WODES 2024.

[4] Visual studio code. `https://code.visualstudio.com/`.

[5] Python extension for visual studio code. `https://marketplace.visualstudio.com/items?itemName=ms-python.python`.

[6] `https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools-extension-pack`.

[7] Mininet. `https://mininet.org/`.

[8] Matplotlib: Visualization with python. `https://matplotlib.org/`.

[9] `https://virt-manager.org/`.

[10] Diagrams net. dagrams.net: Security-first diagramming for teams. `https://www.diagrams.net/`.

[11] Overleaf. Overleaf, the online latex editor. `https://www.overleaf.com`.

[12] L.L. Peterson and B.S. Davie. *Computer Networks: A Systems Approach*. The Morgan Kaufmann Series in Networking. Elsevier Science, 2021.

[13] A.S. Tanenbaum and D.J. Wetherall. *Computer Networks*. Pearson custom library. Pearson, 2013.

[14] Ahmed Nasrallah, Akhilesh S. Thyagaturu, Ziyad Alharbi, Cuixiang Wang, Xing Shao, Martin Reisslein, and Hesham ElBakoury. Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research. *IEEE Communications Surveys & Tutorials*, 21(1):88–145, 2019.

[15] Youhwan Seol, Doyeon Hyeon, Junhong Min, Moonbeom Kim, and Jeongyeup Paek. Timely Survey of Time-Sensitive Networking: Past and Future Directions. *IEEE Access*, 9:142506–142527, 2021.

[16] Sameer Seth and M. Venkatesulu. *Kernel Implementation of Sockets*, pages 101–119. 01 2008.

[17] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. The express data path: fast programmable packet processing in the operating system kernel. In *CoNEXT '18*, page 54–66, New York, NY, USA, 2018. Association for Computing Machinery.

[18] The Linux Foundation. Myth-busting dpdk in 2020, 2020. `https://nextgeninfra.io/wp-content/uploads/2020/07/AvidThink-Linux-Foundation-Myth-busting-DPDK-in-2020-Research-Brief-REV-B.pdf`.

[19] kernel.org. Documentación expulsión del kernel. `https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/kernel/Kconfig.preempt`.

[20] IEEE. IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks, IEEE Standard 802.1Q-2014, 2014.

[21] Industrial Internet Consortium (iiC). Time Sensitive Networks for Flexible Manufacturing Testbed Characterization and Mapping of Converged Traffic Types, 2019.

[22] Thomas Stüber, Lukas Osswald, Steffen Lindner, and Michael Menth. A survey of scheduling algorithms for the time-aware shaper in time-sensitive networking (tsn). *IEEE Access*, 11:61192–61233, 2023.

[23] Hamza Chahed and Andreas Kassler. TSN network scheduling - challenges and approaches. *Network*, 3(4):585–624, 2023.

[24] Hamza Chahed and Andreas J. Kassler. Software-defined time sensitive networks configuration and management. In *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 124–128, 2021.

[25] Nesting. `https://gitlab.com/ipvs/nesting`.

[26] Omnet++. `https://omnetpp.org/`.

[27] Rtaw-pegase. `https://www.realtimeatwork.com/rtaw-pegase/`.

[28] Alexander Oliver Mildner. Evaluation of Online Schedule Synthesis Algorithms for Time-Based Scheduled Time Sensitive Networks. Master's thesis, Department of Informatics, City, State, 2019. MSc Thesis. Supervisor: Prof. Dr.-Ing. Georg Carle. Advisors: Max Helm, Benedikt Jaeger, Dr. Marcel Wagner (Intel), Héctor Blanco Alcaine (Intel).

[29] Maxime Samson, Thomas Vergnaud, Éric Dujardin, Laurent Ciarletta, and Ye-Qiong Song. A model-based approach to automatic generation of tsn network simulations. *2022 IEEE 18th International Conference on Factory Communication Systems (WFCS)*, pages 1–8, 2022.

[30] Marian Ulbricht, Javier Acevedo, Surik Krdoyan, and Frank H. P. Fitzek. Emulation vs. reality: Hardware/software co-design in emulated and real time-sensitive networks. *European Wireless 2021; 26th European Wireless Conference*, pages 1–7, 2021.

[31] Gagan Nandha Kumar, Kostas Katsalis, Panagiotis Papadimitriou, Paul Pop, and Georg Carle. Failure handling for time-sensitive networks using sdn and source routing. In *Proceedings of 2021 IEEE 7th International Conference on Network Softwarization*, pages 226–234, United States, 2021. IEEE. 7¡sup¿th¡/sup¿International Conference on Network Softwarization, NetSoft 2021 ; Conference date: 28-06-2021 Through 02-07-2021.

[32] Ferenc Fejes, Péter Antal, and Márton Kerekes. The tsn building blocks in linux, 2022.

[33] Jakub Sitnicki. Bpf and kernel preemption. `https://lore.kernel.org/bpf/CAMy7=ZWPc279vnKK6L1fssp5h7cb6cqS9_EuMNbfVBg_ixmTrQ@mail.gmail.com/T/`. [Online; accessed 7-July-2024].

[34] Mininettsn paches for integrating tsn in mininet. `https://github.com/ulbricht-inr/MininetTSN`.

[35] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling real-time communication in ieee 802.1qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, page 183–192, New York, NY, USA, 2016. Association for Computing Machinery.

[36] Intel Co. 11 th gen intel® core™ processors real-time tuning guide. `https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://cdrdv2-public.intel.com/640980/Real-Time%2520Tuning%2520Guide%2520-%252011th%2520Gen%2520Intel%2520Core%2520Processors%2520-%25201.3.pdf`.

[37] William H. Tranter Rodger E. Ziemer. *Principles of Communications*. Technology & Engineering. John Wiley & Sons,, 2014.

[38] medium.com. Learn network namespaces and virtual ethernet (veth) devices with graphs. `https://medium.com/@amazingandyyy/introduction-to-network-namespaces-and-virtual-ethernet-veth-devices-304e0c02d084`.

[39] mininet.org. Guia de instalación de mininet. `https://mininet.org/download/`.

[40] Ahmed Nasrallah, Venkatraman Balasubramanian, Akhilesh Thyagaturu, Martin Reisslein, and Hesham ElBakoury. Reconfiguration algorithms for high precision communications in time sensitive networks. In *2019 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, 2019.

[41] Ingo Lütkebohle. BWorld Robot Control Software, 2008. `http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/`.

[42] Acontis Technologies. Building a rt kernel in ubuntu. `https://www.acontis.com/en/building-a-real-time-linux-kernel-in-ubuntu-preemptrt.html`.

# Siglas

`ETF` Earliest TxTime First. 26, 48, 61

`netem` Network Emulation. 27, 48–51, 72

`skb` Socket Buffer. 11, 12, 37, 71

`taprio` Time-Aware Priority Shaper. 11, 17, 25–27, 29–31, 33, 47–51, 53, 59, 61, 72

`tc` Linux Traffic Control. 11

`veth` Virtualized Ethernet. 11, 25, 28, 29, 36–38, 52, 61

**API** Application Programming Interface. 47

**ARP** Address Resolution Protocol. 52

**BE** *Best Effort* (no prioritario). 16, 17

**BPF** Berkeley Packet Filtering. 11–13, 29, 38

**CBS** Credit Base Shapper. 19, 22, 26

**CNC** Centralized Network Configurator. 19–22, 71

**CPU** Central Processing Unit. 42, 43, 59

**CUC** Centralized User Configuration. 20, 21

**DLL** Data Link Layer. 9, 37

**DPDK** Data Plane Development Kit. 13

**eBPF** Extended Berkeley Packet Filter. 12

**GCL** Gate Control List. 19, 20, 26, 49, 51, 52

**GM** Grandmaster. 18, 27

**gPTP** Generalized Precision Time Protocol. 18, 25, 31

**HPN** High Performance Networking. 4

**IACS** Industrial Automation and Control Systems. 16

**iiC** Internet Industrial Consortium. 16

**ISA** Instruction Set Architecture. 35

**IT** Information Technology. 3

**JIT** Just-In-Time. 11

**LNS** Linux Network Stack. 9, 11–13, 28, 33

**MPSoC** Multiprocessor System on a Chip. 32

**NIC** Network Interface Card. 9, 13, 14, 23, 34, 36, 48, 52

**NTP** Network Time Protocol. 32

**OT** Operational Technology. 3

**PCP** Priority Code Point. 16–18, 25–27, 29–31, 71

**PPLM** Problema de Programación Lineal Mixta. 50, 51, 61

**PTP** Precision Time Protocol. 17, 18, 27, 32

**QoS** Quality of Service. 15

**SDN** Software Defined Network. 3, 23, 60

**SRP** Stream Reservation Protocol. 20

**TAS** Time Aware Shapper. 16, 19, 20, 22, 23, 26, 27, 29, 30, 48, 49, 51–53, 59, 71

**TCP** Transmission Control Protocol. 37

**TFG** Trabajo de Fin de Grado. 3–6, 11, 12, 14, 17, 20, 21, 23–25, 27, 29, 32–36, 45, 47, 50, 57, 60, 61, 71, I, III

**TR** Tiempo Real. 4, 32–35, 37, 45, 48, 59

**TSN** Time-Sensitive Networking. 3–7, 9, 11, 15–18, 20–27, 30–34, 37, 40, 45, 47, 48, 50, 52, 53, 59–61, 71, 73, I, III, IV

**UDP** User Datagram Protocol. 37, 48

**ULL** Ultra-Low Latency. 3

**VLAN** Virtual Local Area Network. 9, 16, 18, 25, 26, 29–31, 71

**XDP** eXpress Data Packet. 5, 12–14, 29, 31, 38–41, 59, 71, 72, 114, 115

# Lista de Figuras

# Lista de Tablas

# Anexos

# Anexos A

# Puesta en Marcha de Mininet

Para la instalación de Mininet se han seguido los pasos de [39]. Se ha instalado la versión 2.3.0. Algunos de los enlaces del script de instalación no funcionan correctamente y ha sido necesario actualizarlos.

Una vez instalado Mininet se ha procedido a instalar un kernel 5.2.21-rt15, y aplicar el parche `PREEMPT_RT` y el parche que aumenta el número de colas de los virtual ethernet pairs:

```
mkdir kernel

cd kernel

wget https://mirrors.edge.kernel.org/pub/  \
linux/kernel/v5.x/linux-5.2.21.tar.gz

wget \
https://mirrors.edge.kernel.org/pub/linux/kernel/ \
projects/rt/5.2/patch-5.2.21-rt15.patch.gz

tar -xzf linux-5.2.21.tar.gz

xz -d patch-5.2.21-rt15.patch.gz

git clone https://github.com/ulbricht-inr/MininetTSN

cp MininetTSN-master/diif.patch linux-5.2.21/drivers/net

cd drivers/net
patch -p4 < diff.patch

cd ../..

patch -p1 < ../patch-5.2.21-rt15.patch
```

Además también es necesario utilizar el compilador gcc v8 en caso de utilizar una versión mas moderna el kernel no se compilará con éxito.

Una vez aplicados los parches y modificado el compilador, se sigue la rutina habitual de parcheado y construcción de un kernel de Linux. Puede encontrarse información para hacerlo de forma nativa en Ubuntu por ejemplo en [42].

# Anexos B

# Resultado Planificación

## B.1. Problema ILP

```
    /* **** flow 0 **** */
/* Frame period: 10000 microseconds */
/* Frame transmission time: 100 microseconds */
/* Traversed nodes: [(1001, 1), (1, 2), (2, 1003)] */
/* Start node: (1001, 1) */
/* End nodes: {(2, 1003)} */
/* Required end-to-end time: 10000 microseconds */
/* Weight for WFQ: 1 */
/* **** flow 1 **** */
/* Frame period: 20000 microseconds */
/* Frame transmission time: 100 microseconds */
/* Traversed nodes: [(1002, 1), (1, 2), (2, 1003)] */
/* Start node: (1002, 1) */
/* End nodes: {(2, 1003)} */
/* Required end-to-end time: 20000 microseconds */
/* Weight for WFQ: 1 */
/* **** flow 2 **** */
/* Frame period: 30000 microseconds */
/* Frame transmission time: 100 microseconds */
/* Traversed nodes: [(1004, 2), (2, 1003)] */
/* Start node: (1004, 2) */
/* End nodes: {(2, 1003)} */
/* Required end-to-end time: 30000 microseconds */
/* Weight for WFQ: 1 */
```

```
/* Hyperperiod: 60000 microseconds */
/* Number of hyperperiods to anayze: 2 */

/* 9 windows in node/port (1, 2) with (Flow, Frame) order: [(0, 0), (1, 0), (0, 1),
(0, 2), (1, 1), (0, 3), (0, 4), (1, 2), (0, 5)] */
/* 11 windows in node/port (2, 1003) with (Flow, Frame) order: [(2, 0), (0, 0), (1,
0), (0, 1), (0, 2), (1, 1), (2, 1), (0, 3), (0, 4), (1, 2), (0, 5)] */
/* 6 windows in node/port (1001, 1) with (Flow, Frame) order: [(0, 0), (0, 1), (0,
(0, 3), (0, 4), (0, 5)] */
/* 2 windows in node/port (1004, 2) with (Flow, Frame) order: [(2, 0), (2, 1)] */
/* 3 windows in node/port (1002, 1) with (Flow, Frame) order: [(1, 0), (1, 1), (1,
*/
/* Maximum number of windows per port: 11 */

/* Objective function */

max: timeMargin;

/* Flow offset constraints (for each flow) */

/* 0 <= flow_offset (default unless free variable)*/
/* flow_offset < flow_period */
st0offset < 10000;
st1offset < 20000;
st2offset < 30000;

/* Frame ready constraints from source end-station (for each frame) */

/* frame_ready = flow_period*num_frame + flow_offset */
st0fr0sw1001pt1ready = 0 + st0offset;
st0fr1sw1001pt1ready = 10000 + st0offset;
st0fr2sw1001pt1ready = 20000 + st0offset;
st0fr3sw1001pt1ready = 30000 + st0offset;
st0fr4sw1001pt1ready = 40000 + st0offset;
st0fr5sw1001pt1ready = 50000 + st0offset;
st1fr0sw1002pt1ready = 0 + st1offset;
st1fr1sw1002pt1ready = 20000 + st1offset;
```

```
st1fr2sw1002pt1ready = 40000 + st1offset;
st2fr0sw1004pt2ready = 0 + st2offset;
st2fr1sw1004pt2ready = 30000 + st2offset;

/* Frame start in first switch constraints (for each frame) */

/* frame_ready <= frame_start */
st0fr0sw1001pt1ready <= st0fr0sw1001pt1start;
st0fr1sw1001pt1ready <= st0fr1sw1001pt1start;
st0fr2sw1001pt1ready <= st0fr2sw1001pt1start;
st0fr3sw1001pt1ready <= st0fr3sw1001pt1start;
st0fr4sw1001pt1ready <= st0fr4sw1001pt1start;
st0fr5sw1001pt1ready <= st0fr5sw1001pt1start;
st0fr0sw1pt2ready <= st0fr0sw1pt2start;
st0fr1sw1pt2ready <= st0fr1sw1pt2start;
st0fr2sw1pt2ready <= st0fr2sw1pt2start;
st0fr3sw1pt2ready <= st0fr3sw1pt2start;
st0fr4sw1pt2ready <= st0fr4sw1pt2start;
st0fr5sw1pt2ready <= st0fr5sw1pt2start;
st0fr0sw2pt1003ready <= st0fr0sw2pt1003start;
st0fr1sw2pt1003ready <= st0fr1sw2pt1003start;
st0fr2sw2pt1003ready <= st0fr2sw2pt1003start;
st0fr3sw2pt1003ready <= st0fr3sw2pt1003start;
st0fr4sw2pt1003ready <= st0fr4sw2pt1003start;
st0fr5sw2pt1003ready <= st0fr5sw2pt1003start;
st1fr0sw1002pt1ready <= st1fr0sw1002pt1start;
st1fr1sw1002pt1ready <= st1fr1sw1002pt1start;
st1fr2sw1002pt1ready <= st1fr2sw1002pt1start;
st1fr0sw1pt2ready <= st1fr0sw1pt2start;
st1fr1sw1pt2ready <= st1fr1sw1pt2start;
st1fr2sw1pt2ready <= st1fr2sw1pt2start;
st1fr0sw2pt1003ready <= st1fr0sw2pt1003start;
st1fr1sw2pt1003ready <= st1fr1sw2pt1003start;
st1fr2sw2pt1003ready <= st1fr2sw2pt1003start;
st2fr0sw1004pt2ready <= st2fr0sw1004pt2start;
st2fr1sw1004pt2ready <= st2fr1sw1004pt2start;
st2fr0sw2pt1003ready <= st2fr0sw2pt1003start;
```

```
st2fr1sw2pt1003ready <= st2fr1sw2pt1003start;

/* Frame completion constraints */

/* flowXframeYswitchZcomplete = flowXframeYswitchZstart +  flowXframeYtransmissionT
st0fr0sw1001pt1complete = st0fr0sw1001pt1start + 100;
st0fr0sw1pt2complete = st0fr0sw1pt2start + 100;
st0fr0sw2pt1003complete = st0fr0sw2pt1003start + 100;
st0fr1sw1001pt1complete = st0fr1sw1001pt1start + 100;
st0fr1sw1pt2complete = st0fr1sw1pt2start + 100;
st0fr1sw2pt1003complete = st0fr1sw2pt1003start + 100;
st0fr2sw1001pt1complete = st0fr2sw1001pt1start + 100;
st0fr2sw1pt2complete = st0fr2sw1pt2start + 100;
st0fr2sw2pt1003complete = st0fr2sw2pt1003start + 100;
st0fr3sw1001pt1complete = st0fr3sw1001pt1start + 100;
st0fr3sw1pt2complete = st0fr3sw1pt2start + 100;
st0fr3sw2pt1003complete = st0fr3sw2pt1003start + 100;
st0fr4sw1001pt1complete = st0fr4sw1001pt1start + 100;
st0fr4sw1pt2complete = st0fr4sw1pt2start + 100;
st0fr4sw2pt1003complete = st0fr4sw2pt1003start + 100;
st0fr5sw1001pt1complete = st0fr5sw1001pt1start + 100;
st0fr5sw1pt2complete = st0fr5sw1pt2start + 100;
st0fr5sw2pt1003complete = st0fr5sw2pt1003start + 100;
st1fr0sw1002pt1complete = st1fr0sw1002pt1start + 100;
st1fr0sw1pt2complete = st1fr0sw1pt2start + 100;
st1fr0sw2pt1003complete = st1fr0sw2pt1003start + 100;
st1fr1sw1002pt1complete = st1fr1sw1002pt1start + 100;
st1fr1sw1pt2complete = st1fr1sw1pt2start + 100;
st1fr1sw2pt1003complete = st1fr1sw2pt1003start + 100;
st1fr2sw1002pt1complete = st1fr2sw1002pt1start + 100;
st1fr2sw1pt2complete = st1fr2sw1pt2start + 100;
st1fr2sw2pt1003complete = st1fr2sw2pt1003start + 100;
st2fr0sw1004pt2complete = st2fr0sw1004pt2start + 100;
st2fr0sw2pt1003complete = st2fr0sw2pt1003start + 100;
st2fr1sw1004pt2complete = st2fr1sw1004pt2start + 100;
st2fr1sw2pt1003complete = st2fr1sw2pt1003start + 100;
```

```
/* Path completion constraints */

/* flowXframeYswitchZcomplete + (propagation_time+processing_time) <= flowXframeYsw
st0fr0sw1001pt1complete + 10 <= st0fr0sw1pt2ready;
st0fr1sw1001pt1complete + 10 <= st0fr1sw1pt2ready;
st0fr2sw1001pt1complete + 10 <= st0fr2sw1pt2ready;
st0fr3sw1001pt1complete + 10 <= st0fr3sw1pt2ready;
st0fr4sw1001pt1complete + 10 <= st0fr4sw1pt2ready;
st0fr5sw1001pt1complete + 10 <= st0fr5sw1pt2ready;
st0fr0sw1pt2complete + 10 <= st0fr0sw2pt1003ready;
st0fr1sw1pt2complete + 10 <= st0fr1sw2pt1003ready;
st0fr2sw1pt2complete + 10 <= st0fr2sw2pt1003ready;
st0fr3sw1pt2complete + 10 <= st0fr3sw2pt1003ready;
st0fr4sw1pt2complete + 10 <= st0fr4sw2pt1003ready;
st0fr5sw1pt2complete + 10 <= st0fr5sw2pt1003ready;
st1fr0sw1002pt1complete + 10 <= st1fr0sw1pt2ready;
st1fr1sw1002pt1complete + 10 <= st1fr1sw1pt2ready;
st1fr2sw1002pt1complete + 10 <= st1fr2sw1pt2ready;
st1fr0sw1pt2complete + 10 <= st1fr0sw2pt1003ready;
st1fr1sw1pt2complete + 10 <= st1fr1sw2pt1003ready;
st1fr2sw1pt2complete + 10 <= st1fr2sw2pt1003ready;
st2fr0sw1004pt2complete + 10 <= st2fr0sw2pt1003ready;
st2fr1sw1004pt2complete + 10 <= st2fr1sw2pt1003ready;

/* Frame order constraints */

/* flowXframeYswitchZcomplete <= flowXframeY+1switchZstart */
st0fr0sw1001pt1complete <= st0fr1sw1001pt1start;
st0fr1sw1001pt1complete <= st0fr2sw1001pt1start;
st0fr2sw1001pt1complete <= st0fr3sw1001pt1start;
st0fr3sw1001pt1complete <= st0fr4sw1001pt1start;
st0fr4sw1001pt1complete <= st0fr5sw1001pt1start;
st0fr0sw1pt2complete <= st0fr1sw1pt2start;
st0fr1sw1pt2complete <= st0fr2sw1pt2start;
st0fr2sw1pt2complete <= st0fr3sw1pt2start;
st0fr3sw1pt2complete <= st0fr4sw1pt2start;
st0fr4sw1pt2complete <= st0fr5sw1pt2start;
```

```
st0fr0sw2pt1003complete <= st0fr1sw2pt1003start;
st0fr1sw2pt1003complete <= st0fr2sw2pt1003start;
st0fr2sw2pt1003complete <= st0fr3sw2pt1003start;
st0fr3sw2pt1003complete <= st0fr4sw2pt1003start;
st0fr4sw2pt1003complete <= st0fr5sw2pt1003start;
st1fr0sw1002pt1complete <= st1fr1sw1002pt1start;
st1fr1sw1002pt1complete <= st1fr2sw1002pt1start;
st1fr0sw1pt2complete <= st1fr1sw1pt2start;
st1fr1sw1pt2complete <= st1fr2sw1pt2start;
st1fr0sw2pt1003complete <= st1fr1sw2pt1003start;
st1fr1sw2pt1003complete <= st1fr2sw2pt1003start;
st2fr0sw1004pt2complete <= st2fr1sw1004pt2start;
st2fr0sw2pt1003complete <= st2fr1sw2pt1003start;

/* Gate open/close constraints (ordered gates) */

/* 0 == gateFirstgapOpenOffset */
/* gateFirstgapCloseOffset = gateXopenOffset */
/*   gateXopenOffset <= gateXcloseOffset */
/*   gateXcloseOffset = gateXgapOpenOffset */
/*   gateXgapOpenOffset <= gateXgapCloseOffset */
/*   gateXgapCloseOffset = gateX+1openOffset */
/* gateLastopenOffset <= gateLastcloseOffset */
/* gateLastcloseOffset = gateLastgapOpenOffset */
/* gateLastgapOpenOffset <= gateLastgapCloseOffset */
/* gateLastgapCloseOffset = Hyperperiod */
0 = sw1pt2gt0w0gapOpenOffset;
sw1pt2gt0w0gapOpenOffset <= sw1pt2gt0w0gapCloseOffset;
sw1pt2gt0w0gapCloseOffset = sw1pt2gt0w0openOffset;
sw1pt2gt0w0openOffset <= sw1pt2gt0w0closeOffset;
sw1pt2gt0w0closeOffset = sw1pt2gt1w0gapOpenOffset;
sw1pt2gt1w0gapOpenOffset <= sw1pt2gt1w0gapCloseOffset;
sw1pt2gt1w0gapCloseOffset = sw1pt2gt1w0openOffset;
sw1pt2gt1w0openOffset <= sw1pt2gt1w0closeOffset;
sw1pt2gt1w0closeOffset = sw1pt2gt0w1gapOpenOffset;
sw1pt2gt0w1gapOpenOffset <= sw1pt2gt0w1gapCloseOffset;
sw1pt2gt0w1gapCloseOffset = sw1pt2gt0w1openOffset;
```

```
sw1pt2gt0w1openOffset <= sw1pt2gt0w1closeOffset;
sw1pt2gt0w1closeOffset = sw1pt2gt0w2gapOpenOffset;
sw1pt2gt0w2gapOpenOffset <= sw1pt2gt0w2gapCloseOffset;
sw1pt2gt0w2gapCloseOffset = sw1pt2gt0w2openOffset;
sw1pt2gt0w2openOffset <= sw1pt2gt0w2closeOffset;
sw1pt2gt0w2closeOffset = sw1pt2gt1w1gapOpenOffset;
sw1pt2gt1w1gapOpenOffset <= sw1pt2gt1w1gapCloseOffset;
sw1pt2gt1w1gapCloseOffset = sw1pt2gt1w1openOffset;
sw1pt2gt1w1openOffset <= sw1pt2gt1w1closeOffset;
sw1pt2gt1w1closeOffset = sw1pt2gt0w3gapOpenOffset;
sw1pt2gt0w3gapOpenOffset <= sw1pt2gt0w3gapCloseOffset;
sw1pt2gt0w3gapCloseOffset = sw1pt2gt0w3openOffset;
sw1pt2gt0w3openOffset <= sw1pt2gt0w3closeOffset;
sw1pt2gt0w3closeOffset = sw1pt2gt0w4gapOpenOffset;
sw1pt2gt0w4gapOpenOffset <= sw1pt2gt0w4gapCloseOffset;
sw1pt2gt0w4gapCloseOffset = sw1pt2gt0w4openOffset;
sw1pt2gt0w4openOffset <= sw1pt2gt0w4closeOffset;
sw1pt2gt0w4closeOffset = sw1pt2gt1w2gapOpenOffset;
sw1pt2gt1w2gapOpenOffset <= sw1pt2gt1w2gapCloseOffset;
sw1pt2gt1w2gapCloseOffset = sw1pt2gt1w2openOffset;
sw1pt2gt1w2openOffset <= sw1pt2gt1w2closeOffset;
sw1pt2gt1w2closeOffset = sw1pt2gt0w5gapOpenOffset;
sw1pt2gt0w5gapOpenOffset <= sw1pt2gt0w5gapCloseOffset;
sw1pt2gt0w5gapCloseOffset = sw1pt2gt0w5openOffset;
sw1pt2gt0w5openOffset <= sw1pt2gt0w5closeOffset;
sw1pt2gt0w5closeOffset = sw1pt2gt1w6gapOpenOffset;
sw1pt2gt1w6gapOpenOffset <= sw1pt2gt1w6gapCloseOffset;
sw1pt2gt1w6gapCloseOffset = 60000;

0 = sw2pt1003gt2w0gapOpenOffset;
sw2pt1003gt2w0gapOpenOffset <= sw2pt1003gt2w0gapCloseOffset;
sw2pt1003gt2w0gapCloseOffset = sw2pt1003gt2w0openOffset;
sw2pt1003gt2w0openOffset <= sw2pt1003gt2w0closeOffset;
sw2pt1003gt2w0closeOffset = sw2pt1003gt0w0gapOpenOffset;
sw2pt1003gt0w0gapOpenOffset <= sw2pt1003gt0w0gapCloseOffset;
sw2pt1003gt0w0gapCloseOffset = sw2pt1003gt0w0openOffset;
sw2pt1003gt0w0openOffset <= sw2pt1003gt0w0closeOffset;
```

```
sw2pt1003gt0w0closeOffset = sw2pt1003gt1w0gapOpenOffset;
sw2pt1003gt1w0gapOpenOffset <= sw2pt1003gt1w0gapCloseOffset;
sw2pt1003gt1w0gapCloseOffset = sw2pt1003gt1w0openOffset;
sw2pt1003gt1w0openOffset <= sw2pt1003gt1w0closeOffset;
sw2pt1003gt1w0closeOffset = sw2pt1003gt0w1gapOpenOffset;
sw2pt1003gt0w1gapOpenOffset <= sw2pt1003gt0w1gapCloseOffset;
sw2pt1003gt0w1gapCloseOffset = sw2pt1003gt0w1openOffset;
sw2pt1003gt0w1openOffset <= sw2pt1003gt0w1closeOffset;
sw2pt1003gt0w1closeOffset = sw2pt1003gt0w2gapOpenOffset;
sw2pt1003gt0w2gapOpenOffset <= sw2pt1003gt0w2gapCloseOffset;
sw2pt1003gt0w2gapCloseOffset = sw2pt1003gt0w2openOffset;
sw2pt1003gt0w2openOffset <= sw2pt1003gt0w2closeOffset;
sw2pt1003gt0w2closeOffset = sw2pt1003gt1w1gapOpenOffset;
sw2pt1003gt1w1gapOpenOffset <= sw2pt1003gt1w1gapCloseOffset;
sw2pt1003gt1w1gapCloseOffset = sw2pt1003gt1w1openOffset;
sw2pt1003gt1w1openOffset <= sw2pt1003gt1w1closeOffset;
sw2pt1003gt1w1closeOffset = sw2pt1003gt2w1gapOpenOffset;
sw2pt1003gt2w1gapOpenOffset <= sw2pt1003gt2w1gapCloseOffset;
sw2pt1003gt2w1gapCloseOffset = sw2pt1003gt2w1openOffset;
sw2pt1003gt2w1openOffset <= sw2pt1003gt2w1closeOffset;
sw2pt1003gt2w1closeOffset = sw2pt1003gt0w3gapOpenOffset;
sw2pt1003gt0w3gapOpenOffset <= sw2pt1003gt0w3gapCloseOffset;
sw2pt1003gt0w3gapCloseOffset = sw2pt1003gt0w3openOffset;
sw2pt1003gt0w3openOffset <= sw2pt1003gt0w3closeOffset;
sw2pt1003gt0w3closeOffset = sw2pt1003gt0w4gapOpenOffset;
sw2pt1003gt0w4gapOpenOffset <= sw2pt1003gt0w4gapCloseOffset;
sw2pt1003gt0w4gapCloseOffset = sw2pt1003gt0w4openOffset;
sw2pt1003gt0w4openOffset <= sw2pt1003gt0w4closeOffset;
sw2pt1003gt0w4closeOffset = sw2pt1003gt1w2gapOpenOffset;
sw2pt1003gt1w2gapOpenOffset <= sw2pt1003gt1w2gapCloseOffset;
sw2pt1003gt1w2gapCloseOffset = sw2pt1003gt1w2openOffset;
sw2pt1003gt1w2openOffset <= sw2pt1003gt1w2closeOffset;
sw2pt1003gt1w2closeOffset = sw2pt1003gt0w5gapOpenOffset;
sw2pt1003gt0w5gapOpenOffset <= sw2pt1003gt0w5gapCloseOffset;
sw2pt1003gt0w5gapCloseOffset = sw2pt1003gt0w5openOffset;
sw2pt1003gt0w5openOffset <= sw2pt1003gt0w5closeOffset;
sw2pt1003gt0w5closeOffset = sw2pt1003gt1w6gapOpenOffset;
```

```
sw2pt1003gt1w6gapOpenOffset <= sw2pt1003gt1w6gapCloseOffset;
sw2pt1003gt1w6gapCloseOffset = 60000;

0 = sw1001pt1gt0w0gapOpenOffset;
sw1001pt1gt0w0gapOpenOffset <= sw1001pt1gt0w0gapCloseOffset;
sw1001pt1gt0w0gapCloseOffset = sw1001pt1gt0w0openOffset;
sw1001pt1gt0w0openOffset <= sw1001pt1gt0w0closeOffset;
sw1001pt1gt0w0closeOffset = sw1001pt1gt0w1gapOpenOffset;
sw1001pt1gt0w1gapOpenOffset <= sw1001pt1gt0w1gapCloseOffset;
sw1001pt1gt0w1gapCloseOffset = sw1001pt1gt0w1openOffset;
sw1001pt1gt0w1openOffset <= sw1001pt1gt0w1closeOffset;
sw1001pt1gt0w1closeOffset = sw1001pt1gt0w2gapOpenOffset;
sw1001pt1gt0w2gapOpenOffset <= sw1001pt1gt0w2gapCloseOffset;
sw1001pt1gt0w2gapCloseOffset = sw1001pt1gt0w2openOffset;
sw1001pt1gt0w2openOffset <= sw1001pt1gt0w2closeOffset;
sw1001pt1gt0w2closeOffset = sw1001pt1gt0w3gapOpenOffset;
sw1001pt1gt0w3gapOpenOffset <= sw1001pt1gt0w3gapCloseOffset;
sw1001pt1gt0w3gapCloseOffset = sw1001pt1gt0w3openOffset;
sw1001pt1gt0w3openOffset <= sw1001pt1gt0w3closeOffset;
sw1001pt1gt0w3closeOffset = sw1001pt1gt0w4gapOpenOffset;
sw1001pt1gt0w4gapOpenOffset <= sw1001pt1gt0w4gapCloseOffset;
sw1001pt1gt0w4gapCloseOffset = sw1001pt1gt0w4openOffset;
sw1001pt1gt0w4openOffset <= sw1001pt1gt0w4closeOffset;
sw1001pt1gt0w4closeOffset = sw1001pt1gt0w5gapOpenOffset;
sw1001pt1gt0w5gapOpenOffset <= sw1001pt1gt0w5gapCloseOffset;
sw1001pt1gt0w5gapCloseOffset = sw1001pt1gt0w5openOffset;
sw1001pt1gt0w5openOffset <= sw1001pt1gt0w5closeOffset;
sw1001pt1gt0w5closeOffset = sw1001pt1gt1w6gapOpenOffset;
sw1001pt1gt1w6gapOpenOffset <= sw1001pt1gt1w6gapCloseOffset;
sw1001pt1gt1w6gapCloseOffset = 60000;

0 = sw1004pt2gt2w0gapOpenOffset;
sw1004pt2gt2w0gapOpenOffset <= sw1004pt2gt2w0gapCloseOffset;
sw1004pt2gt2w0gapCloseOffset = sw1004pt2gt2w0openOffset;
sw1004pt2gt2w0openOffset <= sw1004pt2gt2w0closeOffset;
sw1004pt2gt2w0closeOffset = sw1004pt2gt2w1gapOpenOffset;
sw1004pt2gt2w1gapOpenOffset <= sw1004pt2gt2w1gapCloseOffset;
```

```
sw1004pt2gt2w1gapCloseOffset = sw1004pt2gt2w1openOffset;
sw1004pt2gt2w1openOffset <= sw1004pt2gt2w1closeOffset;
sw1004pt2gt2w1closeOffset = sw1004pt2gt3w2gapOpenOffset;
sw1004pt2gt3w2gapOpenOffset <= sw1004pt2gt3w2gapCloseOffset;
sw1004pt2gt3w2gapCloseOffset = 60000;

0 = sw1002pt1gt1w0gapOpenOffset;
sw1002pt1gt1w0gapOpenOffset <= sw1002pt1gt1w0gapCloseOffset;
sw1002pt1gt1w0gapCloseOffset = sw1002pt1gt1w0openOffset;
sw1002pt1gt1w0openOffset <= sw1002pt1gt1w0closeOffset;
sw1002pt1gt1w0closeOffset = sw1002pt1gt1w1gapOpenOffset;
sw1002pt1gt1w1gapOpenOffset <= sw1002pt1gt1w1gapCloseOffset;
sw1002pt1gt1w1gapCloseOffset = sw1002pt1gt1w1openOffset;
sw1002pt1gt1w1openOffset <= sw1002pt1gt1w1closeOffset;
sw1002pt1gt1w1closeOffset = sw1002pt1gt1w2gapOpenOffset;
sw1002pt1gt1w2gapOpenOffset <= sw1002pt1gt1w2gapCloseOffset;
sw1002pt1gt1w2gapCloseOffset = sw1002pt1gt1w2openOffset;
sw1002pt1gt1w2openOffset <= sw1002pt1gt1w2closeOffset;
sw1002pt1gt1w2closeOffset = sw1002pt1gt2w3gapOpenOffset;
sw1002pt1gt2w3gapOpenOffset <= sw1002pt1gt2w3gapCloseOffset;
sw1002pt1gt2w3gapCloseOffset = 60000;



/* Gap calculation */

/* brigeXportYgap = sum(bridgeXportYgaps) */
/* maxgap = sum(bridgeXportYgap) */
sw1pt2gap = + sw1pt2gt0w0gapCloseOffset - sw1pt2gt0w0gapOpenOffset
sw1pt2gt1w0gapCloseOffset - sw1pt2gt1w0gapOpenOffset + sw1pt2gt0w1gapCloseOffset -
sw1pt2gt0w1gapOpenOffset + sw1pt2gt0w2gapCloseOffset - sw1pt2gt0w2gapOpenOffset +
sw1pt2gt1w1gapCloseOffset - sw1pt2gt1w1gapOpenOffset + sw1pt2gt0w3gapCloseOffset -
sw1pt2gt0w3gapOpenOffset + sw1pt2gt0w4gapCloseOffset - sw1pt2gt0w4gapOpenOffset +
sw1pt2gt1w2gapCloseOffset - sw1pt2gt1w2gapOpenOffset + sw1pt2gt0w5gapCloseOffset -
sw1pt2gt0w5gapOpenOffset + sw1pt2gt1w6gapCloseOffset - sw1pt2gt1w6gapOpenOffset;

sw2pt1003gap = + sw2pt1003gt2w0gapCloseOffset - sw2pt1003gt2w0gapOpenOffset +
sw2pt1003gt0w0gapCloseOffset - sw2pt1003gt0w0gapOpenOffset +
```

```
sw2pt1003gt1w0gapCloseOffset - sw2pt1003gt1w0gapOpenOffset +
sw2pt1003gt0w1gapCloseOffset - sw2pt1003gt0w1gapOpenOffset +
sw2pt1003gt0w2gapCloseOffset - sw2pt1003gt0w2gapOpenOffset +
sw2pt1003gt1w1gapCloseOffset - sw2pt1003gt1w1gapOpenOffset +
sw2pt1003gt2w1gapCloseOffset - sw2pt1003gt2w1gapOpenOffset +
sw2pt1003gt0w3gapCloseOffset - sw2pt1003gt0w3gapOpenOffset +
sw2pt1003gt0w4gapCloseOffset - sw2pt1003gt0w4gapOpenOffset +
sw2pt1003gt1w2gapCloseOffset - sw2pt1003gt1w2gapOpenOffset +
sw2pt1003gt0w5gapCloseOffset - sw2pt1003gt0w5gapOpenOffset +
sw2pt1003gt1w6gapCloseOffset - sw2pt1003gt1w6gapOpenOffset;
sw1001pt1gap = + sw1001pt1gt0w0gapCloseOffset - sw1001pt1gt0w0gapOpenOffset +
sw1001pt1gt0w1gapCloseOffset - sw1001pt1gt0w1gapOpenOffset +
sw1001pt1gt0w2gapCloseOffset - sw1001pt1gt0w2gapOpenOffset +
sw1001pt1gt0w3gapCloseOffset - sw1001pt1gt0w3gapOpenOffset +
sw1001pt1gt0w4gapCloseOffset - sw1001pt1gt0w4gapOpenOffset +
sw1001pt1gt0w5gapCloseOffset - sw1001pt1gt0w5gapOpenOffset +
sw1001pt1gt1w6gapCloseOffset - sw1001pt1gt1w6gapOpenOffset;

sw1004pt2gap = + sw1004pt2gt2w0gapCloseOffset - sw1004pt2gt2w0gapOpenOffset +
sw1004pt2gt2w1gapCloseOffset - sw1004pt2gt2w1gapOpenOffset +
sw1004pt2gt3w2gapCloseOffset - sw1004pt2gt3w2gapOpenOffset;
sw1002pt1gap = + sw1002pt1gt1w0gapCloseOffset - sw1002pt1gt1w0gapOpenOffset +
sw1002pt1gt1w1gapCloseOffset - sw1002pt1gt1w1gapOpenOffset +
sw1002pt1gt1w2gapCloseOffset - sw1002pt1gt1w2gapOpenOffset +
sw1002pt1gt2w3gapCloseOffset - sw1002pt1gt2w3gapOpenOffset;
maxgap = + sw1pt2gap + sw2pt1003gap + sw1001pt1gap + sw1004pt2gap + sw1002pt1gap;

/* Frame start offset constraints */

/* flowXframeYswitchZstart = flowXframeYswitchZoffset +  SUM( NUMwindow
flowXframeYswitchZwindowNUMwindow ) */
st0fr0sw1001pt1start = st0fr0sw1001pt1offset + 0 st0fr0sw1001pt1window0hp0 + 60000
st0fr0sw1001pt1window0hp1;
st0fr0sw1pt2start = st0fr0sw1pt2offset + 0 st0fr0sw1pt2window0hp0 + 60000
st0fr0sw1pt2window0hp1;
st0fr0sw2pt1003start = st0fr0sw2pt1003offset + 0 st0fr0sw2pt1003window0hp0 + 60000
st0fr0sw2pt1003window0hp1;
```

```
st0fr1sw1001pt1start = st0fr1sw1001pt1offset + 0 st0fr1sw1001pt1window1hp0 + 60000
st0fr1sw1001pt1window1hp1;
st0fr1sw1pt2start = st0fr1sw1pt2offset + 0 st0fr1sw1pt2window1hp0 + 60000
st0fr1sw1pt2window1hp1;
st0fr1sw2pt1003start = st0fr1sw2pt1003offset + 0 st0fr1sw2pt1003window1hp0 + 60000
st0fr1sw2pt1003window1hp1;
st0fr2sw1001pt1start = st0fr2sw1001pt1offset + 0 st0fr2sw1001pt1window2hp0 + 60000
st0fr2sw1001pt1window2hp1;
st0fr2sw1pt2start = st0fr2sw1pt2offset + 0 st0fr2sw1pt2window2hp0 + 60000
st0fr2sw1pt2window2hp1;
st0fr2sw2pt1003start = st0fr2sw2pt1003offset + 0 st0fr2sw2pt1003window2hp0 + 60000
st0fr2sw2pt1003window2hp1;
st0fr3sw1001pt1start = st0fr3sw1001pt1offset + 0 st0fr3sw1001pt1window3hp0 + 60000
st0fr3sw1001pt1window3hp1;
st0fr3sw1pt2start = st0fr3sw1pt2offset + 0 st0fr3sw1pt2window3hp0 + 60000
st0fr3sw1pt2window3hp1;
st0fr3sw2pt1003start = st0fr3sw2pt1003offset + 0 st0fr3sw2pt1003window3hp0 + 60000
st0fr3sw2pt1003window3hp1;
st0fr4sw1001pt1start = st0fr4sw1001pt1offset + 0 st0fr4sw1001pt1window4hp0 + 60000
st0fr4sw1001pt1window4hp1;
st0fr4sw1pt2start = st0fr4sw1pt2offset + 0 st0fr4sw1pt2window4hp0 + 60000
st0fr4sw1pt2window4hp1;
st0fr4sw2pt1003start = st0fr4sw2pt1003offset + 0 st0fr4sw2pt1003window4hp0 + 60000
st0fr4sw2pt1003window4hp1;
st0fr5sw1001pt1start = st0fr5sw1001pt1offset + 0 st0fr5sw1001pt1window5hp0 + 60000
st0fr5sw1001pt1window5hp1;
st0fr5sw1pt2start = st0fr5sw1pt2offset + 0 st0fr5sw1pt2window5hp0 + 60000
st0fr5sw1pt2window5hp1;
st0fr5sw2pt1003start = st0fr5sw2pt1003offset + 0 st0fr5sw2pt1003window5hp0 + 60000
st0fr5sw2pt1003window5hp1;
st1fr0sw1002pt1start = st1fr0sw1002pt1offset + 0 st1fr0sw1002pt1window0hp0 + 60000
st1fr0sw1002pt1window0hp1;
st1fr0sw1pt2start = st1fr0sw1pt2offset + 0 st1fr0sw1pt2window0hp0 + 60000
st1fr0sw1pt2window0hp1;
st1fr0sw2pt1003start = st1fr0sw2pt1003offset + 0 st1fr0sw2pt1003window0hp0 + 60000
st1fr0sw2pt1003window0hp1;
st1fr1sw1002pt1start = st1fr1sw1002pt1offset + 0 st1fr1sw1002pt1window1hp0 + 60000
```

```
st1fr1sw1002pt1window1hp1;
st1fr1sw1pt2start = st1fr1sw1pt2offset + 0 st1fr1sw1pt2window1hp0 + 60000
st1fr1sw1pt2window1hp1;
st1fr1sw2pt1003start = st1fr1sw2pt1003offset + 0 st1fr1sw2pt1003window1hp0 + 60000
st1fr1sw2pt1003window1hp1;
st1fr2sw1002pt1start = st1fr2sw1002pt1offset + 0 st1fr2sw1002pt1window2hp0 + 60000
st1fr2sw1002pt1window2hp1;
st1fr2sw1pt2start = st1fr2sw1pt2offset + 0 st1fr2sw1pt2window2hp0 + 60000
st1fr2sw1pt2window2hp1;
st1fr2sw2pt1003start = st1fr2sw2pt1003offset + 0 st1fr2sw2pt1003window2hp0 + 60000
st1fr2sw2pt1003window2hp1;
st2fr0sw1004pt2start = st2fr0sw1004pt2offset + 0 st2fr0sw1004pt2window0hp0 + 60000
st2fr0sw1004pt2window0hp1;
st2fr0sw2pt1003start = st2fr0sw2pt1003offset + 0 st2fr0sw2pt1003window0hp0 + 60000
st2fr0sw2pt1003window0hp1;
st2fr1sw1004pt2start = st2fr1sw1004pt2offset + 0 st2fr1sw1004pt2window1hp0 + 60000
st2fr1sw1004pt2window1hp1;
st2fr1sw2pt1003start = st2fr1sw2pt1003offset + 0 st2fr1sw2pt1003window1hp0 + 60000
st2fr1sw2pt1003window1hp1;

/* 1 = flowXframeYswitchZwindow0 + flowXframeYswitchZwindow1 +... */
1 = + st0fr0sw1001pt1window0hp0 + st0fr0sw1001pt1window0hp1;
1 = + st0fr0sw1pt2window0hp0 + st0fr0sw1pt2window0hp1;
1 = + st0fr0sw2pt1003window0hp0 + st0fr0sw2pt1003window0hp1;
1 = + st0fr1sw1001pt1window1hp0 + st0fr1sw1001pt1window1hp1;
1 = + st0fr1sw1pt2window1hp0 + st0fr1sw1pt2window1hp1;
1 = + st0fr1sw2pt1003window1hp0 + st0fr1sw2pt1003window1hp1;
1 = + st0fr2sw1001pt1window2hp0 + st0fr2sw1001pt1window2hp1;
1 = + st0fr2sw1pt2window2hp0 + st0fr2sw1pt2window2hp1;
1 = + st0fr2sw2pt1003window2hp0 + st0fr2sw2pt1003window2hp1;
1 = + st0fr3sw1001pt1window3hp0 + st0fr3sw1001pt1window3hp1;
1 = + st0fr3sw1pt2window3hp0 + st0fr3sw1pt2window3hp1;
1 = + st0fr3sw2pt1003window3hp0 + st0fr3sw2pt1003window3hp1;
1 = + st0fr4sw1001pt1window4hp0 + st0fr4sw1001pt1window4hp1;
1 = + st0fr4sw1pt2window4hp0 + st0fr4sw1pt2window4hp1;
1 = + st0fr4sw2pt1003window4hp0 + st0fr4sw2pt1003window4hp1;
1 = + st0fr5sw1001pt1window5hp0 + st0fr5sw1001pt1window5hp1;
```

```
1 = + st0fr5sw1pt2window5hp0 + st0fr5sw1pt2window5hp1;
1 = + st0fr5sw2pt1003window5hp0 + st0fr5sw2pt1003window5hp1;
1 = + st1fr0sw1002pt1window0hp0 + st1fr0sw1002pt1window0hp1;
1 = + st1fr0sw1pt2window0hp0 + st1fr0sw1pt2window0hp1;
1 = + st1fr0sw2pt1003window0hp0 + st1fr0sw2pt1003window0hp1;
1 = + st1fr1sw1002pt1window1hp0 + st1fr1sw1002pt1window1hp1;
1 = + st1fr1sw1pt2window1hp0 + st1fr1sw1pt2window1hp1;
1 = + st1fr1sw2pt1003window1hp0 + st1fr1sw2pt1003window1hp1;
1 = + st1fr2sw1002pt1window2hp0 + st1fr2sw1002pt1window2hp1;
1 = + st1fr2sw1pt2window2hp0 + st1fr2sw1pt2window2hp1;
1 = + st1fr2sw2pt1003window2hp0 + st1fr2sw2pt1003window2hp1;
1 = + st2fr0sw1004pt2window0hp0 + st2fr0sw1004pt2window0hp1;
1 = + st2fr0sw2pt1003window0hp0 + st2fr0sw2pt1003window0hp1;
1 = + st2fr1sw1004pt2window1hp0 + st2fr1sw1004pt2window1hp1;
1 = + st2fr1sw2pt1003window1hp0 + st2fr1sw2pt1003window1hp1;

/* Gate to frame transmission start constraints */

/* gateXopenOffset = flowXframeYswitchZstartoffset - MaxClockError */
sw1001pt1gt0w0openOffset = st0fr0sw1001pt1offset - 500;
sw1pt2gt0w0openOffset = st0fr0sw1pt2offset - 500;
sw2pt1003gt0w0openOffset = st0fr0sw2pt1003offset - 500;
sw1001pt1gt0w1openOffset = st0fr1sw1001pt1offset - 500;
sw1pt2gt0w1openOffset = st0fr1sw1pt2offset - 500;
sw2pt1003gt0w1openOffset = st0fr1sw2pt1003offset - 500;
sw1001pt1gt0w2openOffset = st0fr2sw1001pt1offset - 500;
sw1pt2gt0w2openOffset = st0fr2sw1pt2offset - 500;
sw2pt1003gt0w2openOffset = st0fr2sw2pt1003offset - 500;
sw1001pt1gt0w3openOffset = st0fr3sw1001pt1offset - 500;
sw1pt2gt0w3openOffset = st0fr3sw1pt2offset - 500;
sw2pt1003gt0w3openOffset = st0fr3sw2pt1003offset - 500;
sw1001pt1gt0w4openOffset = st0fr4sw1001pt1offset - 500;
sw1pt2gt0w4openOffset = st0fr4sw1pt2offset - 500;
sw2pt1003gt0w4openOffset = st0fr4sw2pt1003offset - 500;
sw1001pt1gt0w5openOffset = st0fr5sw1001pt1offset - 500;
sw1pt2gt0w5openOffset = st0fr5sw1pt2offset - 500;
sw2pt1003gt0w5openOffset = st0fr5sw2pt1003offset - 500;
```

```
sw1002pt1gt1w0openOffset = st1fr0sw1002pt1offset - 500;
sw1pt2gt1w0openOffset = st1fr0sw1pt2offset - 500;
sw2pt1003gt1w0openOffset = st1fr0sw2pt1003offset - 500;
sw1002pt1gt1w1openOffset = st1fr1sw1002pt1offset - 500;
sw1pt2gt1w1openOffset = st1fr1sw1pt2offset - 500;
sw2pt1003gt1w1openOffset = st1fr1sw2pt1003offset - 500;
sw1002pt1gt1w2openOffset = st1fr2sw1002pt1offset - 500;
sw1pt2gt1w2openOffset = st1fr2sw1pt2offset - 500;
sw2pt1003gt1w2openOffset = st1fr2sw2pt1003offset - 500;
sw1004pt2gt2w0openOffset = st2fr0sw1004pt2offset - 500;
sw2pt1003gt2w0openOffset = st2fr0sw2pt1003offset - 500;
sw1004pt2gt2w1openOffset = st2fr1sw1004pt2offset - 500;
sw2pt1003gt2w1openOffset = st2fr1sw2pt1003offset - 500;

/* Gate to frame transmission complete constraints */

/* flowXframeYswitchZstartoffset + flowXframeYtransmissionTime
+ MaxClockError = gateXcloseOffset */
st0fr0sw1001pt1offset + 100 + 500 = sw1001pt1gt0w0closeOffset;
st0fr0sw1pt2offset + 100 + 500 = sw1pt2gt0w0closeOffset;
st0fr0sw2pt1003offset + 100 + 500 = sw2pt1003gt0w0closeOffset;
st0fr1sw1001pt1offset + 100 + 500 = sw1001pt1gt0w1closeOffset;
st0fr1sw1pt2offset + 100 + 500 = sw1pt2gt0w1closeOffset;
st0fr1sw2pt1003offset + 100 + 500 = sw2pt1003gt0w1closeOffset;
st0fr2sw1001pt1offset + 100 + 500 = sw1001pt1gt0w2closeOffset;
st0fr2sw1pt2offset + 100 + 500 = sw1pt2gt0w2closeOffset;
st0fr2sw2pt1003offset + 100 + 500 = sw2pt1003gt0w2closeOffset;
st0fr3sw1001pt1offset + 100 + 500 = sw1001pt1gt0w3closeOffset;
st0fr3sw1pt2offset + 100 + 500 = sw1pt2gt0w3closeOffset;
st0fr3sw2pt1003offset + 100 + 500 = sw2pt1003gt0w3closeOffset;
st0fr4sw1001pt1offset + 100 + 500 = sw1001pt1gt0w4closeOffset;
st0fr4sw1pt2offset + 100 + 500 = sw1pt2gt0w4closeOffset;
st0fr4sw2pt1003offset + 100 + 500 = sw2pt1003gt0w4closeOffset;
st0fr5sw1001pt1offset + 100 + 500 = sw1001pt1gt0w5closeOffset;
st0fr5sw1pt2offset + 100 + 500 = sw1pt2gt0w5closeOffset;
st0fr5sw2pt1003offset + 100 + 500 = sw2pt1003gt0w5closeOffset;
st1fr0sw1002pt1offset + 100 + 500 = sw1002pt1gt1w0closeOffset;
```

```
st1fr0sw1pt2offset + 100 + 500 = sw1pt2gt1w0closeOffset;
st1fr0sw2pt1003offset + 100 + 500 = sw2pt1003gt1w0closeOffset;
st1fr1sw1002pt1offset + 100 + 500 = sw1002pt1gt1w1closeOffset;
st1fr1sw1pt2offset + 100 + 500 = sw1pt2gt1w1closeOffset;
st1fr1sw2pt1003offset + 100 + 500 = sw2pt1003gt1w1closeOffset;
st1fr2sw1002pt1offset + 100 + 500 = sw1002pt1gt1w2closeOffset;
st1fr2sw1pt2offset + 100 + 500 = sw1pt2gt1w2closeOffset;
st1fr2sw2pt1003offset + 100 + 500 = sw2pt1003gt1w2closeOffset;
st2fr0sw1004pt2offset + 100 + 500 = sw1004pt2gt2w0closeOffset;
st2fr0sw2pt1003offset + 100 + 500 = sw2pt1003gt2w0closeOffset;
st2fr1sw1004pt2offset + 100 + 500 = sw1004pt2gt2w1closeOffset;
st2fr1sw2pt1003offset + 100 + 500 = sw2pt1003gt2w1closeOffset;

/* Time margin constraints */

/* time_margin = flowXtime_margin + flowYtime_margin... */
timeMargin = + st0timeMargin + st1timeMargin + st2timeMargin;

/* flowXtime_margin <= flowXframeYtime_margin */
st0timeMargin <= st0fr0timeMargin;
st0timeMargin <= st0fr1timeMargin;
st0timeMargin <= st0fr2timeMargin;
st0timeMargin <= st0fr3timeMargin;
st0timeMargin <= st0fr4timeMargin;
st0timeMargin <= st0fr5timeMargin;
st1timeMargin <= st1fr0timeMargin;
st1timeMargin <= st1fr1timeMargin;
st1timeMargin <= st1fr2timeMargin;
st2timeMargin <= st2fr0timeMargin;
st2timeMargin <= st2fr1timeMargin;

/* flowXframeYtime_margin <= e2e_requirement - flowXframeYdelay */
st0fr0timeMargin = 10000 - st0fr0delay;
st0fr1timeMargin = 10000 - st0fr1delay;
st0fr2timeMargin = 10000 - st0fr2delay;
st0fr3timeMargin = 10000 - st0fr3delay;
st0fr4timeMargin = 10000 - st0fr4delay;
```

```
st0fr5timeMargin = 10000 - st0fr5delay;
st1fr0timeMargin = 20000 - st1fr0delay;
st1fr1timeMargin = 20000 - st1fr1delay;
st1fr2timeMargin = 20000 - st1fr2delay;
st2fr0timeMargin = 30000 - st2fr0delay;
st2fr1timeMargin = 30000 - st2fr1delay;

/* Frame delay constraints */

/* frame_delay = frame_complete + (propagation_time+processing_time)
- frame_offset */
st0fr0sw2pt1003complete + 10 - st0fr0sw1001pt1ready <= st0fr0delay;
st0fr1sw2pt1003complete + 10 - st0fr1sw1001pt1ready <= st0fr1delay;
st0fr2sw2pt1003complete + 10 - st0fr2sw1001pt1ready <= st0fr2delay;
st0fr3sw2pt1003complete + 10 - st0fr3sw1001pt1ready <= st0fr3delay;
st0fr4sw2pt1003complete + 10 - st0fr4sw1001pt1ready <= st0fr4delay;
st0fr5sw2pt1003complete + 10 - st0fr5sw1001pt1ready <= st0fr5delay;
st1fr0sw2pt1003complete + 10 - st1fr0sw1002pt1ready <= st1fr0delay;
st1fr1sw2pt1003complete + 10 - st1fr1sw1002pt1ready <= st1fr1delay;
st1fr2sw2pt1003complete + 10 - st1fr2sw1002pt1ready <= st1fr2delay;
st2fr0sw2pt1003complete + 10 - st2fr0sw1004pt2ready <= st2fr0delay;
st2fr1sw2pt1003complete + 10 - st2fr1sw1004pt2ready <= st2fr1delay;

/* Jitter constraints */

/* flowXaverage_delay = sum(flowXframeYdelay) / max_frames */
st0avgDelay = + 0.166667 st0fr0delay + 0.166667 st0fr1delay + 0.166667 st0fr2delay
+ 0.166667 st0fr3delay + 0.166667 st0fr4delay + 0.166667 st0fr5delay;
st1avgDelay = + 0.333333 st1fr0delay + 0.333333 st1fr1delay + 0.333333 st1fr2delay;
st2avgDelay = + 0.500000 st2fr0delay + 0.500000 st2fr1delay;

/* flowXaverage_delay - flowXframeYdelay <=  flowXframeYjitter */
/* flowXframeYdelay - flowXaverage_delay <=  flowXframeYjitter */
st0avgDelay - st0fr0delay <= st0fr0jitter;
st0fr0delay - st0avgDelay <= st0fr0jitter;
st0avgDelay - st0fr1delay <= st0fr1jitter;
st0fr1delay - st0avgDelay <= st0fr1jitter;
```

```
st0avgDelay - st0fr2delay <= st0fr2jitter;
st0fr2delay - st0avgDelay <= st0fr2jitter;
st0avgDelay - st0fr3delay <= st0fr3jitter;
st0fr3delay - st0avgDelay <= st0fr3jitter;
st0avgDelay - st0fr4delay <= st0fr4jitter;
st0fr4delay - st0avgDelay <= st0fr4jitter;
st0avgDelay - st0fr5delay <= st0fr5jitter;
st0fr5delay - st0avgDelay <= st0fr5jitter;
st1avgDelay - st1fr0delay <= st1fr0jitter;
st1fr0delay - st1avgDelay <= st1fr0jitter;
st1avgDelay - st1fr1delay <= st1fr1jitter;
st1fr1delay - st1avgDelay <= st1fr1jitter;
st1avgDelay - st1fr2delay <= st1fr2jitter;
st1fr2delay - st1avgDelay <= st1fr2jitter;
st2avgDelay - st2fr0delay <= st2fr0jitter;
st2fr0delay - st2avgDelay <= st2fr0jitter;
st2avgDelay - st2fr1delay <= st2fr1jitter;
st2fr1delay - st2avgDelay <= st2fr1jitter;

/* flowXframeYjitter <=  flowXjitter */
st0fr0jitter <= st0jitter;
st0fr1jitter <= st0jitter;
st0fr2jitter <= st0jitter;
st0fr3jitter <= st0jitter;
st0fr4jitter <= st0jitter;
st0fr5jitter <= st0jitter;
st1fr0jitter <= st1jitter;
st1fr1jitter <= st1jitter;
st1fr2jitter <= st1jitter;
st2fr0jitter <= st2jitter;
st2fr1jitter <= st2jitter;

/* flowXjitter <=  maxjitter */
st0jitter <= maxjitter;
st1jitter <= maxjitter;
st2jitter <= maxjitter;
```

```
/* Binary variables (hyperperiod where each frame is transmitted)*/

bin st0fr0sw1001pt1window0hp0, st0fr0sw1001pt1window0hp1, st0fr0sw1pt2window0hp0,
st0fr0sw1pt2window0hp1, st0fr0sw2pt1003window0hp0, st0fr0sw2pt1003window0hp1,
st0fr1sw1001pt1window1hp0, st0fr1sw1001pt1window1hp1, st0fr1sw1pt2window1hp0,
st0fr1sw1pt2window1hp1, st0fr1sw2pt1003window1hp0, st0fr1sw2pt1003window1hp1,
st0fr2sw1001pt1window2hp0, st0fr2sw1001pt1window2hp1, st0fr2sw1pt2window2hp0,
st0fr2sw1pt2window2hp1, st0fr2sw2pt1003window2hp0, st0fr2sw2pt1003window2hp1,
st0fr3sw1001pt1window3hp0, st0fr3sw1001pt1window3hp1, st0fr3sw1pt2window3hp0,
st0fr3sw1pt2window3hp1, st0fr3sw2pt1003window3hp0, st0fr3sw2pt1003window3hp1,
st0fr4sw1001pt1window4hp0, st0fr4sw1001pt1window4hp1, st0fr4sw1pt2window4hp0,
st0fr4sw1pt2window4hp1, st0fr4sw2pt1003window4hp0, st0fr4sw2pt1003window4hp1,
st0fr5sw1001pt1window5hp0, st0fr5sw1001pt1window5hp1, st0fr5sw1pt2window5hp0,
st0fr5sw1pt2window5hp1, st0fr5sw2pt1003window5hp0, st0fr5sw2pt1003window5hp1,
st1fr0sw1002pt1window0hp0, st1fr0sw1002pt1window0hp1, st1fr0sw1pt2window0hp0,
st1fr0sw1pt2window0hp1, st1fr0sw2pt1003window0hp0, st1fr0sw2pt1003window0hp1,
st1fr1sw1002pt1window1hp0, st1fr1sw1002pt1window1hp1, st1fr1sw1pt2window1hp0,
st1fr1sw1pt2window1hp1, st1fr1sw2pt1003window1hp0, st1fr1sw2pt1003window1hp1,
st1fr2sw1002pt1window2hp0, st1fr2sw1002pt1window2hp1, st1fr2sw1pt2window2hp0,
st1fr2sw1pt2window2hp1, st1fr2sw2pt1003window2hp0, st1fr2sw2pt1003window2hp1,
st2fr0sw1004pt2window0hp0, st2fr0sw1004pt2window0hp1, st2fr0sw2pt1003window0hp0,
st2fr0sw2pt1003window0hp1, st2fr1sw1004pt2window1hp0, st2fr1sw1004pt2window1hp1,
st2fr1sw2pt1003window1hp0, st2fr1sw2pt1003window1hp1;
```

## B.2. Resultado ILP

```
Actual values of the variables:
Value of objective function: 59120.00000000
Actual values of the variables:
timeMargin                  59120
st0offset                    1490
st1offset                    2590
st2offset                     500
st0fr0sw1001pt1ready         1490
st0fr1sw1001pt1ready        11490
st0fr2sw1001pt1ready        21490
```

```
st0fr3sw1001pt1ready        31490
st0fr4sw1001pt1ready        41490
st0fr5sw1001pt1ready        51490
st1fr0sw1002pt1ready         2590
st1fr1sw1002pt1ready        22590
st1fr2sw1002pt1ready        42590
st2fr0sw1004pt2ready          500
st2fr1sw1004pt2ready        30500
st0fr0sw1001pt1start         1490
st0fr1sw1001pt1start        11490
st0fr2sw1001pt1start        21490
st0fr3sw1001pt1start        31490
st0fr4sw1001pt1start        41490
st0fr5sw1001pt1start        51490
st0fr0sw1pt2ready            1600
st0fr0sw1pt2start            1600
st0fr1sw1pt2ready           11600
st0fr1sw1pt2start           11600
st0fr2sw1pt2ready           21600
st0fr2sw1pt2start           21600
st0fr3sw1pt2ready           31600
st0fr3sw1pt2start           31600
st0fr4sw1pt2ready           41600
st0fr4sw1pt2start           41600
st0fr5sw1pt2ready           51600
st0fr5sw1pt2start           51600
st0fr0sw2pt1003ready         1710
st0fr0sw2pt1003start         1710
st0fr1sw2pt1003ready        11710
st0fr1sw2pt1003start        11710
st0fr2sw2pt1003ready        21710
st0fr2sw2pt1003start        21710
st0fr3sw2pt1003ready        31710
st0fr3sw2pt1003start        31710
st0fr4sw2pt1003ready        41710
st0fr4sw2pt1003start        41710
st0fr5sw2pt1003ready        51710
```

```
st0fr5sw2pt1003start        51710
st1fr0sw1002pt1start         2590
st1fr1sw1002pt1start        22590
st1fr2sw1002pt1start        42590
st1fr0sw1pt2ready            2700
st1fr0sw1pt2start            2700
st1fr1sw1pt2ready           22700
st1fr1sw1pt2start           22700
st1fr2sw1pt2ready           42700
st1fr2sw1pt2start           42700
st1fr0sw2pt1003ready         2810
st1fr0sw2pt1003start         2810
st1fr1sw2pt1003ready        22810
st1fr1sw2pt1003start        22810
st1fr2sw2pt1003ready        42810
st1fr2sw2pt1003start        42810
st2fr0sw1004pt2start          500
st2fr1sw1004pt2start        30500
st2fr0sw2pt1003ready          610
st2fr0sw2pt1003start          610
st2fr1sw2pt1003ready        30610
st2fr1sw2pt1003start        30610
st0fr0sw1001pt1complete        1590
st0fr0sw1pt2complete          1700
st0fr0sw2pt1003complete        1810
st0fr1sw1001pt1complete       11590
st0fr1sw1pt2complete          11700
st0fr1sw2pt1003complete       11810
st0fr2sw1001pt1complete       21590
st0fr2sw1pt2complete          21700
st0fr2sw2pt1003complete       21810
st0fr3sw1001pt1complete       31590
st0fr3sw1pt2complete          31700
st0fr3sw2pt1003complete       31810
st0fr4sw1001pt1complete       41590
st0fr4sw1pt2complete          41700
st0fr4sw2pt1003complete       41810
```

```
st0fr5sw1001pt1complete        51590
st0fr5sw1pt2complete        51700
st0fr5sw2pt1003complete        51810
st1fr0sw1002pt1complete         2690
st1fr0sw1pt2complete         2800
st1fr0sw2pt1003complete         2910
st1fr1sw1002pt1complete        22690
st1fr1sw1pt2complete        22800
st1fr1sw2pt1003complete        22910
st1fr2sw1002pt1complete        42690
st1fr2sw1pt2complete        42800
st1fr2sw2pt1003complete        42910
st2fr0sw1004pt2complete          600
st2fr0sw2pt1003complete          710
st2fr1sw1004pt2complete        30600
st2fr1sw2pt1003complete        30710
sw1pt2gt0w0gapOpenOffset            0
sw1pt2gt0w0gapCloseOffset         1100
sw1pt2gt0w0openOffset         1100
sw1pt2gt0w0closeOffset         2200
sw1pt2gt1w0gapOpenOffset         2200
sw1pt2gt1w0gapCloseOffset         2200
sw1pt2gt1w0openOffset         2200
sw1pt2gt1w0closeOffset         3300
sw1pt2gt0w1gapOpenOffset         3300
sw1pt2gt0w1gapCloseOffset        11100
sw1pt2gt0w1openOffset        11100
sw1pt2gt0w1closeOffset        12200
sw1pt2gt0w2gapOpenOffset        12200
sw1pt2gt0w2gapCloseOffset        21100
sw1pt2gt0w2openOffset        21100
sw1pt2gt0w2closeOffset        22200
sw1pt2gt1w1gapOpenOffset        22200
sw1pt2gt1w1gapCloseOffset        22200
sw1pt2gt1w1openOffset        22200
sw1pt2gt1w1closeOffset        23300
sw1pt2gt0w3gapOpenOffset        23300
```

```
sw1pt2gt0w3gapCloseOffset        31100
sw1pt2gt0w3openOffset         31100
sw1pt2gt0w3closeOffset         32200
sw1pt2gt0w4gapOpenOffset         32200
sw1pt2gt0w4gapCloseOffset         41100
sw1pt2gt0w4openOffset         41100
sw1pt2gt0w4closeOffset         42200
sw1pt2gt1w2gapOpenOffset         42200
sw1pt2gt1w2gapCloseOffset         42200
sw1pt2gt1w2openOffset         42200
sw1pt2gt1w2closeOffset         43300
sw1pt2gt0w5gapOpenOffset         43300
sw1pt2gt0w5gapCloseOffset         51100
sw1pt2gt0w5openOffset         51100
sw1pt2gt0w5closeOffset         52200
sw1pt2gt1w6gapOpenOffset         52200
sw1pt2gt1w6gapCloseOffset         60000
sw2pt1003gt2w0gapOpenOffset            0
sw2pt1003gt2w0gapCloseOffset          110
sw2pt1003gt2w0openOffset          110
sw2pt1003gt2w0closeOffset          1210
sw2pt1003gt0w0gapOpenOffset          1210
sw2pt1003gt0w0gapCloseOffset          1210
sw2pt1003gt0w0openOffset          1210
sw2pt1003gt0w0closeOffset          2310
sw2pt1003gt1w0gapOpenOffset          2310
sw2pt1003gt1w0gapCloseOffset          2310
sw2pt1003gt1w0openOffset          2310
sw2pt1003gt1w0closeOffset          3410
sw2pt1003gt0w1gapOpenOffset          3410
sw2pt1003gt0w1gapCloseOffset         11210
sw2pt1003gt0w1openOffset         11210
sw2pt1003gt0w1closeOffset         12310
sw2pt1003gt0w2gapOpenOffset         12310
sw2pt1003gt0w2gapCloseOffset         21210
sw2pt1003gt0w2openOffset         21210
sw2pt1003gt0w2closeOffset         22310
```

```
sw2pt1003gt1w1gapOpenOffset        22310
sw2pt1003gt1w1gapCloseOffset       22310
sw2pt1003gt1w1openOffset        22310
sw2pt1003gt1w1closeOffset       23410
sw2pt1003gt2w1gapOpenOffset        23410
sw2pt1003gt2w1gapCloseOffset       30110
sw2pt1003gt2w1openOffset        30110
sw2pt1003gt2w1closeOffset       31210
sw2pt1003gt0w3gapOpenOffset        31210
sw2pt1003gt0w3gapCloseOffset       31210
sw2pt1003gt0w3openOffset        31210
sw2pt1003gt0w3closeOffset       32310
sw2pt1003gt0w4gapOpenOffset        32310
sw2pt1003gt0w4gapCloseOffset       41210
sw2pt1003gt0w4openOffset        41210
sw2pt1003gt0w4closeOffset       42310
sw2pt1003gt1w2gapOpenOffset        42310
sw2pt1003gt1w2gapCloseOffset       42310
sw2pt1003gt1w2openOffset        42310
sw2pt1003gt1w2closeOffset       43410
sw2pt1003gt0w5gapOpenOffset        43410
sw2pt1003gt0w5gapCloseOffset       51210
sw2pt1003gt0w5openOffset        51210
sw2pt1003gt0w5closeOffset       52310
sw2pt1003gt1w6gapOpenOffset        52310
sw2pt1003gt1w6gapCloseOffset       60000
sw1001pt1gt0w0gapOpenOffset            0
sw1001pt1gt0w0gapCloseOffset         990
sw1001pt1gt0w0openOffset          990
sw1001pt1gt0w0closeOffset         2090
sw1001pt1gt0w1gapOpenOffset        2090
sw1001pt1gt0w1gapCloseOffset       10990
sw1001pt1gt0w1openOffset        10990
sw1001pt1gt0w1closeOffset       12090
sw1001pt1gt0w2gapOpenOffset        12090
sw1001pt1gt0w2gapCloseOffset       20990
sw1001pt1gt0w2openOffset        20990
```

```
sw1001pt1gt0w2closeOffset        22090
sw1001pt1gt0w3gapOpenOffset        22090
sw1001pt1gt0w3gapCloseOffset        30990
sw1001pt1gt0w3openOffset        30990
sw1001pt1gt0w3closeOffset        32090
sw1001pt1gt0w4gapOpenOffset        32090
sw1001pt1gt0w4gapCloseOffset        40990
sw1001pt1gt0w4openOffset        40990
sw1001pt1gt0w4closeOffset        42090
sw1001pt1gt0w5gapOpenOffset        42090
sw1001pt1gt0w5gapCloseOffset        50990
sw1001pt1gt0w5openOffset        50990
sw1001pt1gt0w5closeOffset        52090
sw1001pt1gt1w6gapOpenOffset        52090
sw1001pt1gt1w6gapCloseOffset        60000
sw1004pt2gt2w0gapOpenOffset            0
sw1004pt2gt2w0gapCloseOffset            0
sw1004pt2gt2w0openOffset            0
sw1004pt2gt2w0closeOffset        1100
sw1004pt2gt2w1gapOpenOffset        1100
sw1004pt2gt2w1gapCloseOffset        30000
sw1004pt2gt2w1openOffset        30000
sw1004pt2gt2w1closeOffset        31100
sw1004pt2gt3w2gapOpenOffset        31100
sw1004pt2gt3w2gapCloseOffset        60000
sw1002pt1gt1w0gapOpenOffset            0
sw1002pt1gt1w0gapCloseOffset        2090
sw1002pt1gt1w0openOffset        2090
sw1002pt1gt1w0closeOffset        3190
sw1002pt1gt1w1gapOpenOffset        3190
sw1002pt1gt1w1gapCloseOffset        22090
sw1002pt1gt1w1openOffset        22090
sw1002pt1gt1w1closeOffset        23190
sw1002pt1gt1w2gapOpenOffset        23190
sw1002pt1gt1w2gapCloseOffset        42090
sw1002pt1gt1w2openOffset        42090
sw1002pt1gt1w2closeOffset        43190
```

```
sw1002pt1gt2w3gapOpenOffset        43190
sw1002pt1gt2w3gapCloseOffset        60000
sw1pt2gap                    50100
sw2pt1003gap                 47900
sw1001pt1gap                 53400
sw1004pt2gap                 57800
sw1002pt1gap                 56700
maxgap                      265900
st0fr0sw1001pt1offset         1490
st0fr0sw1001pt1window0hp0            1
st0fr0sw1001pt1window0hp1            0
st0fr0sw1pt2offset           1600
st0fr0sw1pt2window0hp0             1
st0fr0sw1pt2window0hp1             0
st0fr0sw2pt1003offset         1710
st0fr0sw2pt1003window0hp0            1
st0fr0sw2pt1003window0hp1            0
st0fr1sw1001pt1offset        11490
st0fr1sw1001pt1window1hp0            1
st0fr1sw1001pt1window1hp1            0
st0fr1sw1pt2offset           11600
st0fr1sw1pt2window1hp0             1
st0fr1sw1pt2window1hp1             0
st0fr1sw2pt1003offset        11710
st0fr1sw2pt1003window1hp0            1
st0fr1sw2pt1003window1hp1            0
st0fr2sw1001pt1offset        21490
st0fr2sw1001pt1window2hp0            1
st0fr2sw1001pt1window2hp1            0
st0fr2sw1pt2offset           21600
st0fr2sw1pt2window2hp0             1
st0fr2sw1pt2window2hp1             0
st0fr2sw2pt1003offset        21710
st0fr2sw2pt1003window2hp0            1
st0fr2sw2pt1003window2hp1            0
st0fr3sw1001pt1offset        31490
st0fr3sw1001pt1window3hp0            1
```

```
st0fr3sw1001pt1window3hp1            0
st0fr3sw1pt2offset          31600
st0fr3sw1pt2window3hp0             1
st0fr3sw1pt2window3hp1             0
st0fr3sw2pt1003offset         31710
st0fr3sw2pt1003window3hp0            1
st0fr3sw2pt1003window3hp1            0
st0fr4sw1001pt1offset         41490
st0fr4sw1001pt1window4hp0            1
st0fr4sw1001pt1window4hp1            0
st0fr4sw1pt2offset          41600
st0fr4sw1pt2window4hp0             1
st0fr4sw1pt2window4hp1             0
st0fr4sw2pt1003offset         41710
st0fr4sw2pt1003window4hp0            1
st0fr4sw2pt1003window4hp1            0
st0fr5sw1001pt1offset         51490
st0fr5sw1001pt1window5hp0            1
st0fr5sw1001pt1window5hp1            0
st0fr5sw1pt2offset          51600
st0fr5sw1pt2window5hp0             1
st0fr5sw1pt2window5hp1             0
st0fr5sw2pt1003offset         51710
st0fr5sw2pt1003window5hp0            1
st0fr5sw2pt1003window5hp1            0
st1fr0sw1002pt1offset          2590
st1fr0sw1002pt1window0hp0            1
st1fr0sw1002pt1window0hp1            0
st1fr0sw1pt2offset           2700
st1fr0sw1pt2window0hp0             1
st1fr0sw1pt2window0hp1             0
st1fr0sw2pt1003offset          2810
st1fr0sw2pt1003window0hp0            1
st1fr0sw2pt1003window0hp1            0
st1fr1sw1002pt1offset         22590
st1fr1sw1002pt1window1hp0            1
st1fr1sw1002pt1window1hp1            0
```

```
st1fr1sw1pt2offset          22700
st1fr1sw1pt2window1hp0          1
st1fr1sw1pt2window1hp1          0
st1fr1sw2pt1003offset       22810
st1fr1sw2pt1003window1hp0          1
st1fr1sw2pt1003window1hp1          0
st1fr2sw1002pt1offset       42590
st1fr2sw1002pt1window2hp0          1
st1fr2sw1002pt1window2hp1          0
st1fr2sw1pt2offset          42700
st1fr2sw1pt2window2hp0          1
st1fr2sw1pt2window2hp1          0
st1fr2sw2pt1003offset       42810
st1fr2sw2pt1003window2hp0          1
st1fr2sw2pt1003window2hp1          0
st2fr0sw1004pt2offset         500
st2fr0sw1004pt2window0hp0          1
st2fr0sw1004pt2window0hp1          0
st2fr0sw2pt1003offset         610
st2fr0sw2pt1003window0hp0          1
st2fr0sw2pt1003window0hp1          0
st2fr1sw1004pt2offset       30500
st2fr1sw1004pt2window1hp0          1
st2fr1sw1004pt2window1hp1          0
st2fr1sw2pt1003offset       30610
st2fr1sw2pt1003window1hp0          1
st2fr1sw2pt1003window1hp1          0
st0timeMargin                9670
st1timeMargin               19670
st2timeMargin               29780
st0fr0timeMargin             9670
st0fr1timeMargin             9670
st0fr2timeMargin             9670
st0fr3timeMargin             9670
st0fr4timeMargin             9670
st0fr5timeMargin             9670
st1fr0timeMargin            19670
```

| | |
|---|---|
| st1fr1timeMargin | 19670 |
| st1fr2timeMargin | 19670 |
| st2fr0timeMargin | 29780 |
| st2fr1timeMargin | 29780 |
| st0fr0delay | 330 |
| st0fr1delay | 330 |
| st0fr2delay | 330 |
| st0fr3delay | 330 |
| st0fr4delay | 330 |
| st0fr5delay | 330 |
| st1fr0delay | 330 |
| st1fr1delay | 330 |
| st1fr2delay | 330 |
| st2fr0delay | 220 |
| st2fr1delay | 220 |
| st0avgDelay | 330.001 |
| st1avgDelay | 330 |
| st2avgDelay | 220 |
| st0fr0jitter | 0.00066 |
| st0fr1jitter | 0.00066 |
| st0fr2jitter | 0.00066 |
| st0fr3jitter | 0.00066 |
| st0fr4jitter | 0.00066 |
| st0fr5jitter | 0.00066 |
| st1fr0jitter | 0.00033 |
| st1fr1jitter | 0.00033 |
| st1fr2jitter | 0.00033 |
| st2fr0jitter | 0 |
| st2fr1jitter | 0 |
| st0jitter | 0.00066 |
| st1jitter | 0.00033 |
| st2jitter | 0 |
| maxjitter | 0.00066 |

## B.3. Implementación de la planificación en la plataforma

Configuración Bridge 1:

```
    qdisc taprio 100: dev s1-eth3 root refcnt 9 tc 4
    map 0 1 2 3 0 0 0 0 0 0 0 0 0 0 0 0
queues offset 0 count 1 offset 1 count 1 offset 2 count 1 offset 3 count 1
clockid REALTIME base-time 0 cycle-time 0 cycle-time-extension 0 base-time
1721988236209191740 cycle-time 60000000 cycle-time-extension 0
index 0 cmd S gatemask 0x1 interval 1100000
index 1 cmd S gatemask 0x2 interval 1100000
index 2 cmd S gatemask 0x4 interval 1100000
index 3 cmd S gatemask 0x1 interval 7800000
index 4 cmd S gatemask 0x2 interval 1100000
index 5 cmd S gatemask 0x1 interval 8900000
index 6 cmd S gatemask 0x2 interval 1100000
index 7 cmd S gatemask 0x4 interval 1100000
index 8 cmd S gatemask 0x1 interval 7800000
index 9 cmd S gatemask 0x2 interval 1100000
index 10 cmd S gatemask 0x1 interval 8900000
index 11 cmd S gatemask 0x2 interval 1100000
index 12 cmd S gatemask 0x4 interval 1100000
index 13 cmd S gatemask 0x1 interval 7800000
index 14 cmd S gatemask 0x2 interval 1100000
index 15 cmd S gatemask 0x1 interval 7800000

qdisc pfifo 0: dev s1-eth3 parent 100:8 limit 1000p
qdisc pfifo 0: dev s1-eth3 parent 100:7 limit 1000p
qdisc pfifo 0: dev s1-eth3 parent 100:6 limit 1000p
qdisc pfifo 0: dev s1-eth3 parent 100:5 limit 1000p
qdisc pfifo 0: dev s1-eth3 parent 100:1 limit 1000p
qdisc netem 30: dev s1-eth3 parent 100:3 limit 1000 delay 99us
qdisc netem 40: dev s1-eth3 parent 100:4 limit 1000 delay 99us
qdisc netem 20: dev s1-eth3 parent 100:2 limit 1000 delay 99us
```

Configuración Bridge S2:

```
    qdisc taprio 100: dev s2-eth1 root refcnt 9 tc 4
    map 0 1 2 3 0 0 0 0 0 0 0 0 0 0 0 0
queues offset 0 count 1 offset 1 count 1 offset 2 count 1 offset 3 count 1
clockid REALTIME base-time 0 cycle-time 0 cycle-time-extension 0 base-time
1721988236209191740 cycle-time 60000000 cycle-time-extension 0
index 0 cmd S gatemask 0x1 interval 110000
index 1 cmd S gatemask 0x8 interval 1100000
index 2 cmd S gatemask 0x2 interval 1100000
index 3 cmd S gatemask 0x4 interval 1100000
index 4 cmd S gatemask 0x1 interval 7800000
index 5 cmd S gatemask 0x2 interval 1100000
index 6 cmd S gatemask 0x1 interval 8900000
index 7 cmd S gatemask 0x2 interval 1100000
index 8 cmd S gatemask 0x4 interval 1100000
index 9 cmd S gatemask 0x1 interval 6700000
index 10 cmd S gatemask 0x8 interval 1100000
index 11 cmd S gatemask 0x2 interval 1100000
index 12 cmd S gatemask 0x1 interval 8900000
index 13 cmd S gatemask 0x2 interval 1100000
index 14 cmd S gatemask 0x4 interval 1100000
index 15 cmd S gatemask 0x1 interval 7800000
index 16 cmd S gatemask 0x2 interval 1100000
index 17 cmd S gatemask 0x1 interval 7690000

qdisc pfifo 0: dev s2-eth1 parent 100:8 limit 1000p
qdisc pfifo 0: dev s2-eth1 parent 100:7 limit 1000p
qdisc pfifo 0: dev s2-eth1 parent 100:6 limit 1000p
qdisc pfifo 0: dev s2-eth1 parent 100:5 limit 1000p
qdisc pfifo 0: dev s2-eth1 parent 100:1 limit 1000p
qdisc netem 30: dev s2-eth1 parent 100:3 limit 1000 delay 99us
qdisc netem 40: dev s2-eth1 parent 100:4 limit 1000 delay 99us
qdisc netem 20: dev s2-eth1 parent 100:2 limit 1000 delay 99us
qdisc clsact ffff: dev s2-eth1 parent ffff:fff1
qdisc noqueue 0: dev s2-eth2 root refcnt 2
qdisc noqueue 0: dev s2-eth3 root refcnt 2
```

```
Configuración h1:
iptables -t mangle -A POSTROUTING -p udp --dport 6666 -j CLASSIFY --set-class 0:1
iptables -t mangle -A POSTROUTING -p udp --dport 7777 -j CLASSIFY --set-class 0:2
iptables -t mangle -A POSTROUTING -p udp --dport 8888 -j CLASSIFY --set-class 0:3

qdisc noqueue 0: dev lo root refcnt 2
qdisc netem 100: dev h1-eth0 root refcnt 9 limit 1000 delay 99us
qdisc noqueue 0: dev h1-eth0.10 root refcnt 2

    Configutación h2:
iptables -t mangle -A POSTROUTING -p udp --dport 6666 -j CLASSIFY --set-class 0:1
iptables -t mangle -A POSTROUTING -p udp --dport 7777 -j CLASSIFY --set-class 0:2
iptables -t mangle -A POSTROUTING -p udp --dport 8888 -j CLASSIFY --set-class 0:3
qdisc noqueue 0: dev lo root refcnt 2
qdisc netem 100: dev h2-eth0 root refcnt 9 limit 1000 delay 99us
qdisc noqueue 0: dev h2-eth0.10 root refcnt 2

    Configuración h4:
iptables -t mangle -A POSTROUTING -p udp --dport 6666 -j CLASSIFY --set-class 0:1
iptables -t mangle -A POSTROUTING -p udp --dport 7777 -j CLASSIFY --set-class 0:2
iptables -t mangle -A POSTROUTING -p udp --dport 8888 -j CLASSIFY --set-class 0:3
qdisc noqueue 0: dev lo root refcnt 2
qdisc netem 100: dev h3-eth0 root refcnt 9 limit 1000 delay 99us
qdisc noqueue 0: dev h3-eth0.10 root refcnt 2
```

Solo se han configurado las interfaces de red necesarias para el *Use Case*

# Anexos C

# IEI DRPC-240-TGL

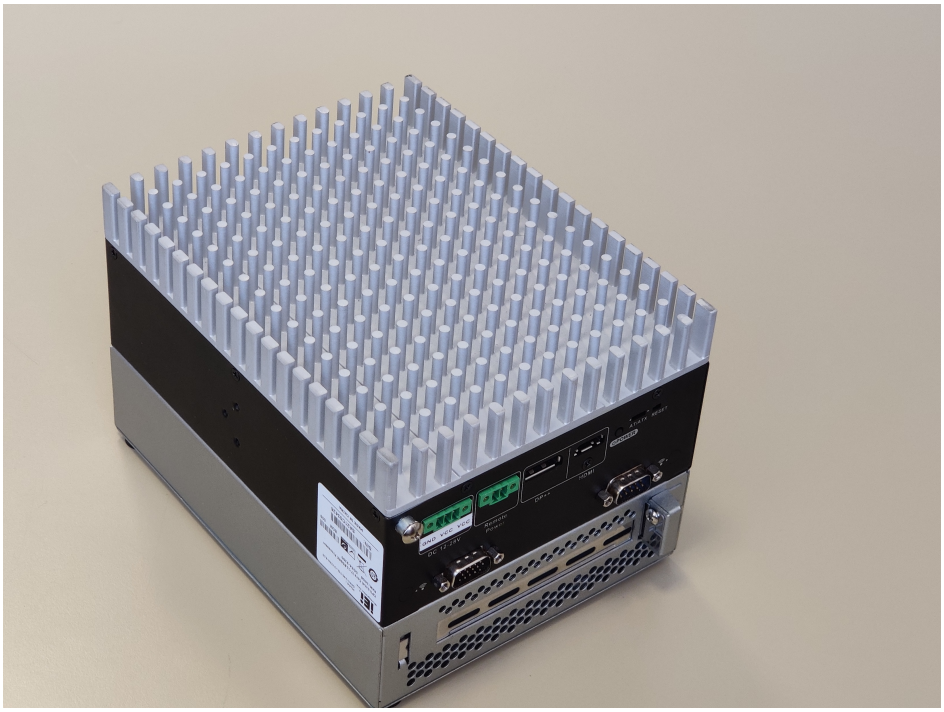


Figura C.1: IEI DRPC-240 TGL

- **Model**: DRPC-240-TGL-U-i7RD-R10
- **Manufacturer**: `https://www.ieiworld.com/en/product/model.php?II=871`
- **CPU**: 11th Gen Intel® Core™ i7-1185GRE CPU @ 2.80GHz 4 cores (core *Tigerlake*)
- **RAM**: 16 GB DDR4-3200

# Anexos D

# Diagramas de medición de latencias

En este anexo se incluyen los diagramas que representan las metodologías de medida definidas en el Cap.4.

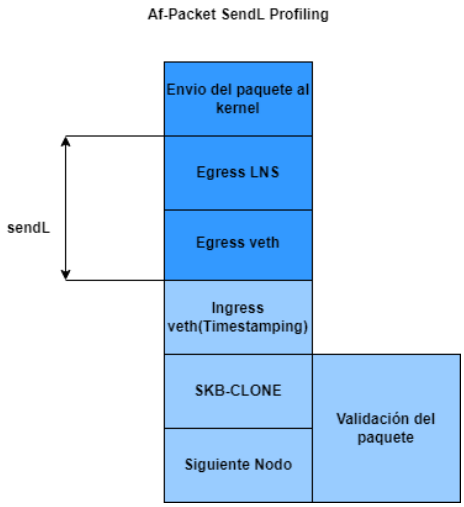



Figura D.1: Medición de la latencia *sendL* con la solución basada en `AF_PACKET`

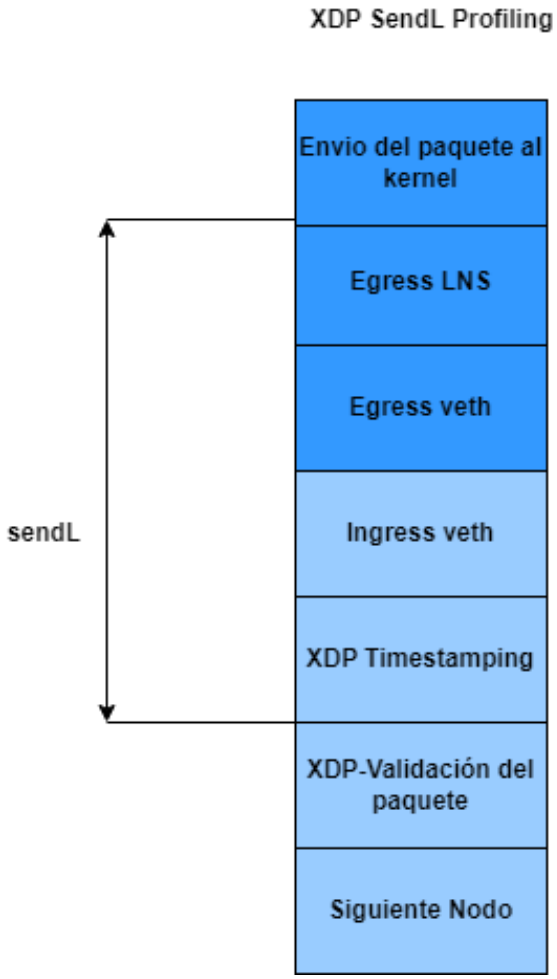



Figura D.2: medición de la latencia *sendL* con la solución basada en XDP

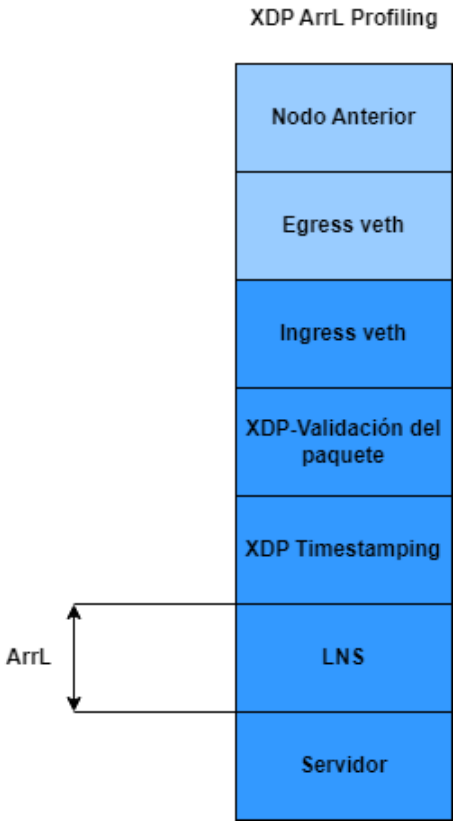



Figura D.3: Medición de la latencia $arrL$ con la solución basada en XDP

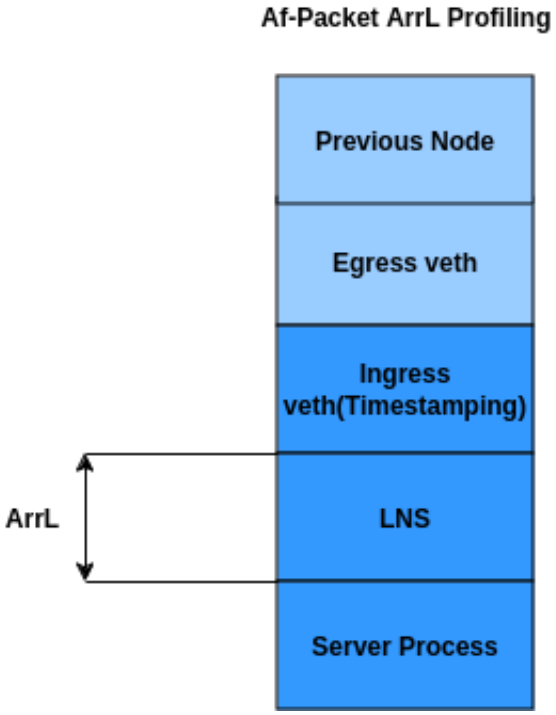



Figura D.4: Medición de la latencia $arrL$ con la solución basada en `AF_PACKET`

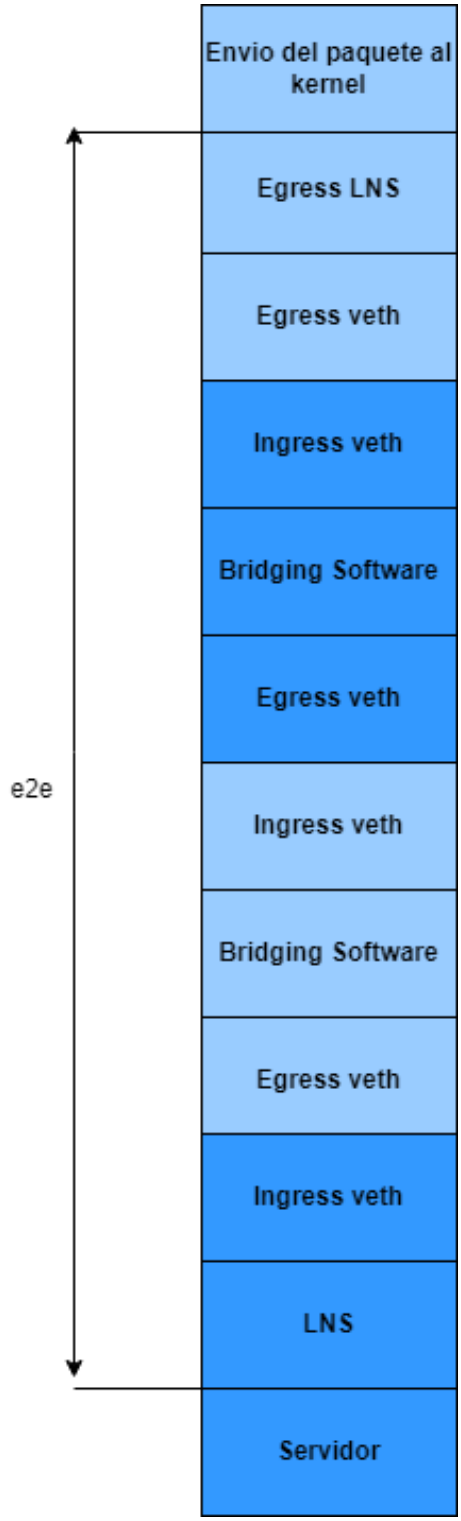



Figura D.5: Medición de la latencia *e2e*

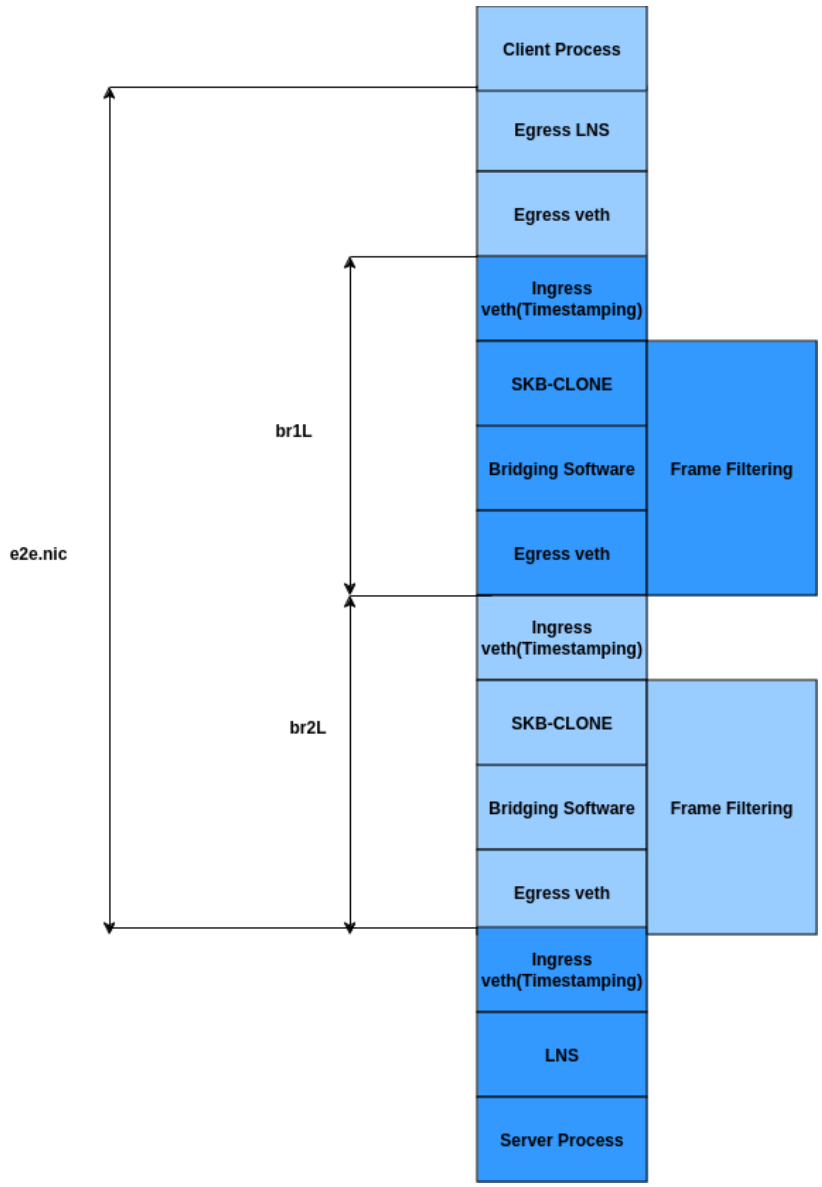



Figura D.6: Medición de la latencia *e2e.nic* con la solución basada en `AF_PACKET`