FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Tesis de Maestría

Sincronización de relojes en Internet.

Autor:
Ing. David Sebastián Samaniego Bermeo

Director:
Dr. Ing. José Ignacio Alvarez-Hamelin

Tesis realizada como parte de los requisitos para la obtención del título de Mágister en Ingeniería en Telecomunicaciones

Grupo de redes complejas y comunicación de datos (CoNexDat)
Facultad de Ingeniería

11 de Mayo de 2018

UNIVERSIDAD DE BUENOS AIRES

Resumen

Grupo de redes complejas y comunicación de datos (CoNexDat)
Facultad de Ingeniería

Mágister en Ingeniería en Telecomunicaciones

Sincronización de relojes en Internet.

por Ing. David Sebastián Samaniego Bermeo

Este trabajo describe el desarrollo de un algoritmo de Sincronización de Relojes en Internet. Inicialmente se hará una revisión de los trabajos previos sobre algoritmos de sincronización en diferentes entornos. Luego se describirán los bancos de ensayo utilizados, así como el *hardware* considerado. Mas adelante se explicará la metodología, el modelo y el desarrollo del algoritmo de sincronización de relojes que se ha denominado SIC (*Synchronizing Internet Clocks*). Luego se mostrará la implementación, junto con los resultados obtenidos. Finalmente se realiza las conclusiones.

Agradecimientos

A mi familia que me apoya incondicionalmente en todos mis proyectos, de manera especial a mis padres. A mi Director Dr. Ing. José Ignacio Alvarez-Hamelin, un agradecimiento infinito, tuvo la paciencia para irme guiando en cada momento, enseñandome hasta a tener estilos de escribir, además de brindarme su amistad. A todos los integrantes del grupo Conextdat, que estuvieron siempre ofreciendome su apoyo cada vez que lo solicité, especialmente a Esteban que en reiteradas ocasiones me ayudó en varios temas relacionados a mi tesis. A las instituciones que permitieron utilizar su infraestructura de red para el desarrollo de este proyecto SMN, EWAY, Innova-Red y USC. Al Instituto de Fomento al Talento Humano, institución que financió mis estudios de Maestría en la Universidad de Buenos Aires mediante una beca. No puedo terminar sin agradecer a mis amigos cercanos, que me han acompañado durante estos años de estudio en Argentina, gracias por su apoyo incondicional.

Índice general

Índice de figuras

Abreviaturas

NTP	**N**etwork **T**ime **P**rotocol
TSC	**T**ime **S**tamp **C**ounter
NCS	**N**etworked **C**ontrol **S**ystem
UTC	**U**niversal **T**ime **C**oordinated
UDP	**U**ser **D**atagram **P**rotocol
IP	**I**nternet **P**rotocol
WAN	**W**ide **A**rea **N**etwork
DNS	**D**omain **N**ame **S**erver
GPS	**G**lobal **P**ositioning **S**ystem
PTP	**P**recision **T**ime **P**rotocol
RTT	**R**ound **T**rip **T**ime
PPS	**P**ulse **P**er **S**econd
CPU	**C**entral **P**rocessing **U**nit
RDTSC	**R**ea**D** **T**ime **S**tamp **C**ounter
DAG	**D**ata **A**cquisition and **G**eneration
DTP	**D**atacenter **T**ime **P**rotocol
CDC	**C**lock **D**omain **C**rossing
LSE	**L**east **S**quare **E**rror
ME	**M**aximum **E**ntropy
GPIO	**G**eneral **P**urpose **I**nput **O**utput
UART	**U**niversal **A**synchronous **R**eceiver **T**ransmitter
IQR	**I**nterquartile **R**ange
ARMA	**A**uto**R**egressive **M**oving **A**verage
RTC	**R**eal **T**ime **C**lock
MTIE	**M**aximum **T**ime **I**nterval **E**rror

TE **T**ime **E**rror

Capítulo 1

Introducción

En la actualidad Internet se ha convertido en la red más grande que se conoce. Su crecimiento y evolución hace que cada vez sea más necesario medirla y supervisarla para poder asegurar un funcionamiento sin problemas y también ayudar al diseño de futuros protocolos de servicio.

Caracterizar con precisión la dinámica del Internet de extremo a extremo resulta excepcionalmente difícil debido a la inmensa heterogeneidad de la red.

Dentro de una red es importante conocer varios aspectos que permitan tener una idea de lo que ocurre cuando existe un flujo de información circulando por ella. Latencia, pérdida de paquetes, cuellos de botella entre otros, son parámetros importantes que determinan las características de una red.

Realizar un monitoreo continuo del tráfico nos permite conocer su comportamiento y dinámica, además nos brinda información de niveles de congestión. Una forma de medir la congestión entre dos extremos de una red es midiendo el tiempo de tránsito de un paquete entre un extremo y otro. Entonces, comparando estos tiempos a lo largo de un período, por ejemplo un día, podremos detectar las horas más congestionadas por ser más elevados dichos tiempos de tránsito. Para medir los ya citados tiempos se utilizan las marcas de tiempo *(timestamps)* en ambos extremos, lo que nos permitirá conocer las demoras en cada sentido particular; sin embargo esta información es útil solamente si estos dos extremos mantienen sus relojes sincronizados.

Una de las formas de sincronizar los relojes es por medio de dispositivos GPS. Estos módulos en la actualidad son utilizados gracias a su precisión y se pueden acoplar a equipos para ser utilizados como referencias de tiempo. Lamentablemente estos dispositivos necesitan estar ubicados físicamente en lugares con vista directa al cielo para poder captar la señal satelital y mantener su hora correcta; es así que esta alternativa no es muy viable en la mayor parte de los casos.

Otra alternativa que podría considerarse es el protocolo NTP, siendo el estándar utilizado para sincronizar relojes. Este protocolo es usado en gran parte de equipamiento de red, servidores y computadores. Sin embargo su baja precisión (en Internet alrededor de 5ms hasta 100ms [1]) y su frecuencia de actualización de un mínimo de $64s$, no resulta útil para realizar mediciones continuas de congestión.

Si consideramos estimar el nivel de congestión, en cada sentido, entre dos extremos interconectados por medio de Internet, es necesario contar con un método que pueda tomar mediciones para obtener información y poder hacer estimaciones de dicha congestión. Como ya hemos dicho, esto nos lleva a la sincronización de los relojes involucrados, ya que no es posible determinar los tiempos de manera independiente si no se conoce la diferencia entre ambos relojes.

En la literatura de sincronización de relojes se habla de distintos términos, en general la relación entre dos relojes se puede expresar como [2]:

$$C_A = k + f \times C_B + d \times C_B + \cdots + x_n C_B \ + \ldots, \tag{1.1}$$

dónde C_A es el reloj en el extremo A y C_B en el extremo B, k es la diferencia neta de tiempo en cierto instante dado tomado de referencia, f representa a la relación entre las frecuencias de ambos, d es conocido como deriva (*drift* en inglés) y modela el cambio en la frecuencia, x_n representa otros términos de orden superior. Dependiendo del modelo que se use, se estimarán los términos necesarios de la ecuación 1.1.

1.1. Organización de la tesis

Este trabajo está organizado en seis capítulos: Introducción (Capítulo 1), Estado del arte (Capítulo 2), Bancos de ensayo (Capítulo 3), Desarrollo del algoritmo de sincronización SIC (Capítulo 4), Implementación de SIC y resultados (Capítulo 5) y finalmente Conclusiones (Capítulo 6).

El capítulo 2 introduce el marco teórico de este trabajo. Se hace una descripción de algunos de los protocolos de sincronización utilizados en diferentes ámbitos y con diferentes esquemas. Inicialmente se describe el protocolo PTP (*Precision Time Protocol*) [3] del estándar IEEE-1588, el cual permite la sincronización de relojes para sistemas de mediciones y control implementados sobre redes de comunicaciones, pudiendo ser aplicable sobre redes de conmutación de paquetes. También se menciona el protocolo NTP (*Network Time Protocol*) [4], el cual se mantiene como el estándar, siendo hoy en día parte de muchos sistemas operativos y equipamiento de red. Otra alternativa que también ha sido tomada en cuenta, ha sido el uso de TSC (*Time Stamp Counter*) [5], que se encuentra disponible a partir de la incursión de procesadores Intel Pentium, muy comunes en equipos de cómputo orientados a usuarios finales. Para los ambientes de datacenter, se describe el protocolo DTP (*Datacenter Time Protocol*) [6] el cual utiliza la capa física de la pila de protocolos de red, siendo capaz de lograr precisión en resoluciones de nanosegundos. Finalmente se hace una reseña sobre una metodología para el cálculo de retardos unidireccionales utilizando mediciones en toda una red.

En el capítulo 3 se describen los bancos de ensayo utilizados para desarrollar y probar el algoritmo, además se describen las herramientas de programación y el *hardware* utilizados para su implementación.

La metodología propuesta, el modelo utilizado y la descripción del algoritmo de sincronización SIC, junto con las pautas consideradas son descritas en el capítulo 4.

Luego de explicar el desarrollo del algoritmo, se describe su implementación junto con los resultados en el capítulo 5.

Finalmente se culmina este trabajo presentando las Conclusiones en el Capítulo 6, además de la Bibliografía.

Capítulo 2

Estado del Arte

En este capítulo se introduce el marco teórico de este trabajo. Inicialmente se hace una descripción sobre el protocolo PTP del estándar IEEE 1588 [3]. Luego se realiza una descripción sobre el protocolo de sincronización estándar NTP [4] el cual es mayormente utilizado en la actualidad por aplicaciones y equipos de uso común. También se explica el protocolo de sincronización DTC [6], el cual está diseñado para entornos de datacenter mediante el uso de la capa física del stack de protocolo de red. Posteriormente, se explica el uso de registros TSC [7] presentes en procesadores modernos para proveer una referencia de tiempo precisa, la cual ha sido probada sobre redes LAN mediante el desarrollo de TSC clock. Finalmente, en [8] se describe una metodología para el calculo de retardos unidireccionales por medio mediciones en una red de nodos interconectados.

2.1. Protocolos de sincronización

2.1.1. Introducción

El protocolo PTP, originalmente fue definido en el estándar IEEE 1588-2002 siendo revisado en el año 2008 en el estándar IEEE-1588-2008. Este protocolo permite la sincronización de relojes para sistemas de mediciones y control implementados sobre redes de comunicaciones, pudiendo ser aplicable sobre redes de conmutación de paquetes. Según este estándar la precisión lograda se encuentra en valores del rango de los submicrosegundos [9], siendo necesario contar con *hardware* dedicado para el caso.

Uno de los protocolos dedicados para sincronización de relojes es NTP, este protocolo ha sido diseñado como un método para ajustar los relojes de manera estándar, y hoy en día forma parte de muchos sistemas operativos. Su uso es muy frecuente sobre todo en equipos y dispositivos que no requieren una alta precisión. En una red donde los retardos de propagación son altos, la precisión ofrecida por NTP se encuentra en la escala de los milisegundos en el mejor de los casos, lo cual es un valor elevado para aplicaciones como medición de congestión de tráfico en redes, sistemas de audio digital de alta calidad o en redes de sensores.

Para los ambientes de datacenter, la sincronización de relojes para las aplicaciones de red y sistemas distribuidos tiene mucha importancia. Desafortunadamente, muchos de los protocolos tradiciones de sincronización se limitan a las redes de conmutación de paquetes. Esto genera que el RTT, el cual debe medirse con precisión para sincronizar los relojes, se vea afectado por variables no deterministicas agregadas por las características propias de una red de este tipo.

Una alternativa para lograr una técnica que permita obtener valores de sincronismo mejores que NTP y que se aproximen a lo ofrecido por IEEE-1588 sin que sea necesario el uso de *hardware* dedicado, ha sido el uso de TSC, que se encuentra disponible a partir de la incursión de procesadores *Intel Pentium*, muy comunes en equipos de cómputo orientados a usuarios finales.

Ciertos servicios como, por ejemplo, las aplicaciones de streaming, dependen de gran manera de las características del camino desde el origen hacia el destino. Otros, como las transacciones entre cliente-servidor, dependen de la calidad del camino desde el servidor hacia el cliente, o, como las video conferencias, dependen de los caminos en ambos sentidos entre los puntos conectados para garantizar la calidad del servicio. Por lo anteriormente mencionado, es muy importante poder medir el retardo unidireccional del tráfico, para lo cual se necesita que los relojes de los puntos medidos se encuentren sincronizados. Si no existe sincronía, al realizar la medición del retardo unidireccional sería un problema el *rate* o frecuencia que existe entre los dos relojes.

Finalmente en [8] se describe una metodología para el cálculo del retardo unidireccional entre nodos de una red sin utilizar ninguna fuente de sincronización de tiempo. Se basa en realizar mediciones unidireccionales múltiples entre algunos pares de nodos para estimar los retardos unidireccionales, mediante la resolución de un sistema de ecuaciones

sobre determinado nodo optimizando el valor general de una función objetivo, la cual es afectada por toda la topología de la red y no únicamente por mediciones individuales. Dos de las funciones objetivo analizadas son el Error Cuadrático y la Entropía. Varios experimentos han demostrado que ambas funciones mejoran considerablemente el sincronismo con respecto al método de dividir el tiempo de ida y vuelta a la mitad; siendo este último utilizado por el servicio de NTP.

2.1.2. PTP (IEEE 1588)

El protocolo PTP, definido en el estándar IEEE-1588, provee los métodos para sincronizar computadoras dentro de una red LAN con una precisión a nivel de los sub-microsegundos. Al tener gran precisión es muy utilizado en sistemas de automatización y robótica. La sincronización la realiza mediante el intercambio de mensajes entre un reloj Maestro (o *Master*) y varios relojes Esclavos, proceso similar al modo Cliente-Servidor. El reloj Maestro es el encargado de proveer el tiempo para que los relojes esclavos se sincronicen. Un dispositivo *Grandmaster* es un Maestro que está sincronizado a una fuente de tiempo provista generalmente por GPS. Los mensajes que incluye el protocolo son:

- *Master sync*
- *Master follow up*
- *Slave clock delay request*
- *Master delay response*

Adicionalmente de los mensajes de sincronización, cuando se tiene varios dispositivos *Grandmaster*, PTP ejecuta el algoritmo BMC (*Best Master Clock*), el cual determina cual es reloj de mayor precisión en la red, en base a esto, se selecciona el reloj *Grandmaster* que sincronizará los relojes esclavos. Si este dispositivo se desconecta de la red o el algoritmo BMC determina que la precisión ofrecida no es la mejor, vuelve a definir un nuevo *Grandmaster*. En el intercambio de mensajes entre Maestro y Esclavo, los *timestamp* (marcas de tiempo) son utilizadas para determinar la latencia de la red, con lo cual se puede realizar el proceso de sincronización. Un mensaje de tipo *sync* es enviado por el Maestro típicamente cada dos segundos y el mensaje *delay request* es enviado con menor frecuencia, alrededor de uno por minuto. Este mensaje no se envía cíclicamente, pero

cada Esclavo calcula un tiempo aleatorio, típicamente entre cuatro y sesenta segundos, y lo usa para activar el envío del mensaje. De esta manera, todas las solicitudes no llegan al mismo tiempo en el Maestro, evitando una posible sobrecarga del nodo. Es importante señalar que la precisión de la sincronización depende de los intervalos de sincronización. Los intervalos más cortos llevan a una mayor precisión, ya que los algoritmos de control en los esclavos proporcionan un mayor número de timestamps recientes. Cuatro *timestamps*

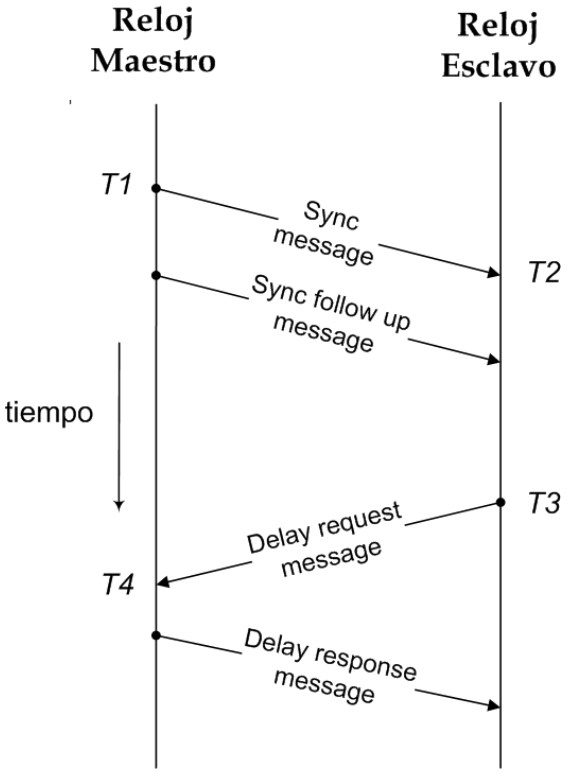

FIGURA 2.1: Intercambio de mensajes de sincronización entre un nodo Maestro y un nodo Esclavo correspondientes al protocolo PTP [10]

son intercambiados entre Maestro y Esclavo, los mismos que se refieren como $T1, T2, T3$ y $T4$ (ver figura 2.1). Dos *offset* del camino $Maestro - Esclavo$ y $Esclavo - Maestro$ deben ser calculados. Para encontrar el *offset* en el nodo Esclavo se deben seguir los siguientes pasos:

- $T1$ es el primer *timestamp*: este tiempo es enviado por el Maestro en el mensaje *follow up*. Este tiempo es tomado cuando el mensaje de sincronización es enviado hacia la interfaz de red.
- $T2$ es el segundo *timestamp*: este contiene el tiempo al momento de la recepción en el Esclavo. Esta diferencia de tiempo (Maestro-Esclavo) puede ser calculada cuando

$T1$ y $T2$ estén disponibles en el Esclavo:

$$Maestro - Esclavo = T2 - T1,$$

- En el mensaje *delay request* enviado por el esclavo se envía el tiempo $T3$.
- El tiempo $T4$ es marcado cuando el mensaje *delay request* es recibido por el Maestro. La diferencia de tiempo Esclavo-Maestro es calculada cuando $T3$ y $T4$ estén disponibles en el Esclavo:

$$Esclavo - Maestro = T4 - T3,$$

- Después de esto, el retardo unidireccional puede ser calculado como:

$$retardo - unidireccional = \frac{(T2 - T1) + (T4 - T3)}{2},$$

- Para la sincronización contra el reloj Maestro, se utilizan valores de *offset*, que se calculan mediante la siguiente ecuación:

$$offset = T2 - T1 - retardo - unidireccional = \frac{T2 + T3 - T4 - T1}{2},$$

Nótese que este protocolo también usa la hipótesis de los caminos simétricos. El protocolo IEEE 1588 no define como implementar los algoritmos en los nodos Maestro y Esclavo. Como todos los algoritmos de sincronización de relojes, PTP puede hacer uso de *timestamps* realizados por *software* o *hardware*.

2.1.3. NTP

El protocolo NTP, es uno de los más antiguos de Internet y aún sigue vigente hoy en día. En funcionamiento desde 1985, lleva más de 30 años de uso continuo. NTP está diseñado para sincronizar el tiempo en una red de ordenadores y equipos. Inicialmente fue pensado para el sistema operativo Linux, pero luego fue migrado también hacia Windows, aunque sigue siendo instalado por defecto en muchos sistemas Unix. Por este motivo existe una gran cantidad de servidores NTP que utilizan Linux debido a su kernel especializado y sus algoritmos de tiempo. Actualmente, la versión que se está utilizando es NTPv4, la

cual es el estándar propuesto en la RFC 5905 [11] y todas las versiones son compatibles entre sí. La única modificación entre la versión 3 y 4 es una variación en la cabecera para incluir IPv6. Utiliza un algoritmo para seleccionar un servidor de tiempo preciso, y está diseñado para mitigar los efectos variables de la latencia en una red. Usualmente se describe en términos de cliente-servidor, pero puede ser utilizado también en conexiones *peer-to-peer*, en donde cualquiera de los *peers* considera al otro como una fuente potencial de tiempo. Este protocolo utiliza la capa de aplicación que usa la capa de transporte UDP en el puerto 123 a través de una red IP, y su funcionamiento es basado en el intercambio de paquetes que contienen *timestamps* entre el cliente y el servidor.

Funcionamiento de NTP. El modo cliente-servidor es la configuración más común en Internet, de esta forma, un cliente envía una solicitud al servidor y espera una respuesta del mismo. El funcionamiento básico del protocolo, en cada envío de paquetes es bastante simple. El cliente arma el paquete, indicando el valor del campo *Origin Timestamp* ($t1$), y procede con el envío. El servidor intercambia direcciones y puertos, escribe el valor del campo *Receive Timestamp* ($t2$) y *Transmit Timestamp* ($t3$), re calcula la suma de comprobación y devuelve el mensaje inmediatamente. Al llegar el paquete de respuesta al cliente, este escribe el campo *Destination Timestamp* ($t4$), obteniendo así los cuatro valores de tiempo, tal como se muestra en la figura 2.2. Con la información recibida en el paquete es posible obtener dos cosas fundamentales para realizar el proceso de sincronización.

- *Clock Offset*: especifica la diferencia entre la hora del sistema local y la referencia externa de reloj.
- *Round-trip delay*: especifica las latencias de tiempo medidas durante la transferencia de paquetes dentro de la red.

Con los datos obtenidos permiten al cliente determinar la hora del servidor con respecto a la hora local y en consecuencia ajustar el reloj local. Además, el mensaje incluye información para seleccionar el mejor servidor.

La precisión de cada servidor esta dada por lo que se conocen como estratos, que configuran una jerarquía de niveles con un número asignado, siendo 0 el estrato más alto, y en el que se encuentran los las fuentes de tiempo como son los relojes atómicos. Los estratos

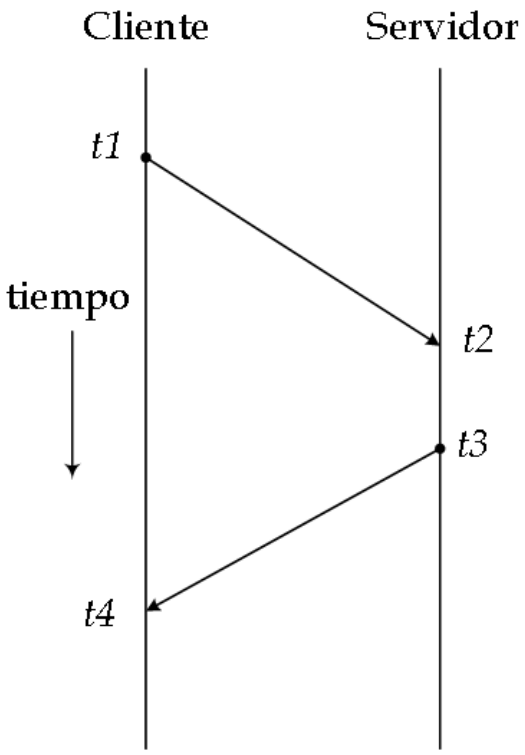

FIGURA 2.2: Intercambio de paquetes entre el cliente y el servidor NTP

inferiores se conectan con su nivel superior, formando un árbol de los consumidores de NTP. Para evitar el cuello de botella, solo se permiten consultas por parte de servidores previamente autorizados. Con ello, todo servidor que esté conectado a un servidor estrato 1 será automáticamente estrato 2 y así sucesivamente. Además del modo de conexión como cliente, un servidor NTP permite la conexión como *peer*, en cuyo caso servidores de estrato 1 en adelante pueden configurarse como un arreglo de servidores para mejorarse mutuamente la exactitud de la hora que suministrarán. Cuando un cliente se conecta a la red, lo hace a través de un servidor de estrato más alto que el propio y lo hará como estrato 16. Con el tiempo, alcanzará un estrato menos del servidor al que se conecta.

Los servidores que proporcionan sincronización a un grupo considerable de clientes normalmente operan como un grupo de tres o más servidores redundantes mutuamente, cada uno de los cuales opera con tres o más servidores estrato 1 o estrato 2 en los modos cliente-servidor, así como todos los demás miembros del grupo en modos simétricos. Esto proporciona protección contra fallos en los que uno o más servidores no funcionan o proporcionan el tiempo incorrecto. NTP evita de dos maneras la sincronización con una máquina cuyo tiempo puede no ser exacto. En primer lugar, NTP nunca se sincroniza

con una máquina que no está sincronizada. En segundo lugar, NTP compara el tiempo reportado por varias máquinas, y no se sincroniza con una máquina cuyo tiempo es significativamente diferente de los demás, incluso si su estrato es menor.

Cada cliente puede asociarse a uno o mas servidores NTP, mediante un archivo de configuración donde se pueden definir los nombres de los servidores. En una red LAN (*Local Área Network*) que posea varios servidores NTP, se pueden utilizar mensajes de difusión (*broadcast*) [1] en vez de mensajes únicos con cada servidor. Esta alternativa reduce la complejidad de la configuración porque cada máquina puede configurarse para enviar o recibir mensajes de difusión, sin embargo la exactitud de la hora se reduce notablemente porque el flujo de información es solamente unidireccional. Los algoritmos NTP están diseñados para resistir los ataques cuando una de las fuentes de sincronización configuradas, accidental o deliberadamente, proporcionan un tiempo incorrecto. En estos casos, se utiliza un procedimiento especial de votación para identificar fuentes falsas y descartar sus datos.

Existen varios modos en los cuales los servidores NTP se pueden asociar con otros.

- Cliente/Servidor.
- Simétrico Activo/Pasivo.
- *Broadcast* y/o *Multicast*.

Modo Cliente-Servidor. Los clientes y servidores dependientes normalmente operan en modo cliente-servidor, en el que un cliente o servidor dependiente se puede sincronizar con un miembro del grupo, pero ningún miembro del grupo puede sincronizarse con el cliente o el servidor dependiente. Esto proporciona protección contra ataques de protocolo.

Modo Simétrico Activo-Pasivo. El modo simétrico activo-pasivo está destinado a configuraciones en las que un grupo de *peers* de estratos bajos actúan como respaldos mutuos entre sí. Cada *peer* dispone de una o más fuentes de referencia primarias, como

[1] Este tipo de mensajes permite enviar un solo paquete y que lo reciban todos los destinatarios a la vez.

un reloj de radio o un subconjunto de servidores secundarios fiables. Si uno de los compañeros pierde todas las fuentes de referencia o simplemente deja de operar, los otros *peers* se reconfiguran automáticamente para que los valores de tiempo puedan fluir de los compañeros supervivientes a todos los demás del circulo. La configuración de una asociación en modo simétrico-activo, se realiza en el archivo de configuración del peer, se indica al servidor remoto que se desea obtener tiempo y que también está dispuesto a suministrar tiempo al servidor remoto si es necesario. Este modo es apropiado en configuraciones que involucran un número de servidores redundantes interconectados a través de diversos caminos de red, lo cual es el caso para la mayoría de los servidores estrato 1 y estrato 2 en Internet. Los modos simétricos se utilizan con más frecuencia entre dos o más servidores que funcionan como un grupo mutuamente redundante. En estos modos, los servidores miembros del grupo organizan las rutas de sincronización para obtener el máximo rendimiento, en función de la fluctuación de la red y del retardo de propagación. Si uno o más de los miembros del grupo fallan, los miembros restantes se reconfiguran automáticamente según sea necesario. Un *peer* se configura en modo simétrico-activo utilizando el comando *peer* y especificando el nombre DNS o la dirección IP del otro *peer*. El otro *peer* también se configura en modo simétrico-activo de esta manera. Si el otro *peer* no está específicamente configurado de esta manera, una asociación simétrico-pasivo se activa al llegar un mensaje simétrico-activo, puesto que un intruso puede suplantar a un *peer* simétrico-activo e inyectar valores falsos de tiempo, es por esto que el modo simétrico siempre debe ser autenticado.

Modo *Broadcast* y/o *Multicast*. Cuando los requisitos de precisión y fiabilidad son modestos, los clientes pueden configurarse para utilizar modos de *broadcast* y/o *multicast*. Normalmente, estos modos no son utilizados por servidores con clientes dependientes. La ventaja es que los clientes no necesitan estar configurados para un servidor específico, permitiendo que todos los clientes operativos utilicen el mismo archivo de configuración. El modo de *broadcast* requiere un servidor de *broadcast* en la misma subred. Dado que los mensajes de *broadcast* no se propagan por routers, sólo se utilizan servidores de *broadcast* en la misma subred. Este modo está destinado a configuraciones que implican a uno o varios servidores y una población de clientes potencialmente grande.

2.1.4. NTP vs PTP

Luego de analizar los protocolos PTP y NTP, es posible notar cierta similitud entre ambos. Un punto en común entre los dos es el uso de *timestamps*, mediante el intercambio de mensajes entre los puntos a sincronizar, con lo cual es posible estimar la diferencia entre sus relojes. Podemos fijarnos, que en el caso de PTP, el intercambio lo inicia el reloj Maestro, mientras que en NTP, es el cliente quien inicia este proceso. Respecto a la precisión ofrecida, en la tecnología utilizada por PTP, en la mayoría de los casos se dispone de *hardware* para el *timestamping* en los nodos Maestro y Esclavos, aunque también es posible hacerlo por *software* en los nodos Esclavos. pero no con igual precisión. NTP si bien se permite *hardware timestamping* en los dispositivos cliente y servidor, no hay muchos dispositivos que lo tengan implementado. Además, la mayoría de los errores en la sincronización de la red, es a menudo debido a los procesos de encolado de paquetes en *switches* y *routers*. NTP no tiene una solución para este problema, mientras que PTP lo tiene solucionado mediante conmutadores llamados relojes de frontera, Un conmutador de este tipo, actúa como un nodo Esclavo para un nodo Maestro y a su vez se convierte un nodo Maestro para los Esclavos que se sincronizarán con él. Ambos protocolos funcionan sobre redes LAN, sin embargo NTP también se extiende hacia redes más extensas como lo es una WAN. Esta característica hace que hoy en día, su uso sea estándar tanto para equipos de computo de usuarios finales, como para dispositivos de red y servidores en los cuales el nivel de sincronización de sus relojes no requiere de mucha precisión. En cambio PTP se desempeña mejor en redes LAN, con lo cual es más utilizado en ambientes en los que los niveles de sincronización son críticos, como lo es en la automatización y la robótica.

2.1.5. DTP

DTP, es un protocolo de sincronización de relojes diseñado para redes dentro de un datacenter, capaz de lograr una precisión de nanosegundos. En esencia, utiliza la capa física de dispositivos de red para implementar un protocolo de sincronización de relojes descentralizado. Al hacerlo, elimina la mayoría de los elementos no deterministicos de los protocolos comunes de sincronización. Además, utiliza los mensajes de control en la capa física para comunicar cientos de miles de mensajes de protocolo sin interferir con los paquetes de capas superiores.

Datacenter Time Protocol. DTP, hace uso de la capa física (PHY) del stack de protocolos de red, para lograr una precisión a nivel de nanosegundos. Aprovecha el hecho que dos *peers* se sincronizan mediante la PHY al momento de transmitir y recibir los flujos de bits. Un *peer* toma el flujo de *bits* de recepción (RX) transmitidos por el *peer* que envía mediante el flujo (TX). De esta manera se tienen dos relojes físicos en dos dispositivos de red, virtualmente ambos en el mismo circuito.

Un *switch* utiliza un oscilador como reloj para alimentar su chip de conmutación, con lo cual todos los flujos TX que atraviesan dicho *switch*, utilizan la misma fuente de reloj. Un *switch* con N dispositivos de red conectados a él, tiene N+1 osciladores físicos para sincronizar. Como los errores producto del retardo y fluctuación de una red se pueden minimizar ejecutando el protocolo en el nivel más bajo, PHY es el mejor lugar para reducir dichos errores.

Tres ventajas del uso de PHY en la sincronización de relojes son:

- Permite realizar el proceso de *timestamping* (marcado de tiempo) en escala de los sub-nanosegundos.
- Al ser la capa física la más baja de todo el stack de protocolos, siempre hay un retardo determinístico [6] entre realizar el *timestamping* y transmitir el mismo.
- Al tener unicamente un medio físico de comunicación entre dos dispositivos de red, la variación del retardo es muy pequeña, cuando no hay transmisión de paquetes, el retardo entre dos dispositivos conectados mediante el PHY, corresponde al tiempo de transmisión de *bits* sobre el cable que los conecta, además del tiempo de procesamiento de los bits de PHY.

Una interfaz de red continuamente genera frames de caracteres especiales para mantener la conexión hacia un *peer*. Dichos caracteres son utilizados para entregar los mensajes del protocolo, sin utilizar un ancho de banda considerable como lo haría un paquete de una capa superior.

En DTP, cada puerto de red tiene un contador local que se va incrementando en cada ciclo del reloj. La secuencia de mensajes entre *peers* por medio de los puertos de red son:

- Un puerto de red envía un mensaje DTP con la marca de su contador local hacia su *peer* y ajusta su reloj local al recibir el valor del contador de su *peer*.

- Dado que DTP trabaja y mantiene los contadores locales en la capa física, los *switches* juegan un rol importante en el crecimiento del número de dispositivos de red sincronizados por el protocolo. Como consecuencia, sincronizar todos los puertos de un *switch* requiere un paso adicional ya que es necesario sincronizar todos los contadores locales de cada puerto local.

- Específicamente, DTP mantiene un contador global que va incrementando en cada ciclo del reloj, pero además toma el valor máximo del contador global y los contadores locales. El algoritmo 1 muestra el proceso para sincronizar los contadores locales entre dos *peers*.

Algoritmo DTP El protocolo DTP utiliza el algoritmo 1 para sincronizar los contadores locales entre dos *peers*, corriendo en dos fases que son: INIT y BEACON.

Algoritmo 1 DTP dentro del un puerto de red

STATE:

gc: contador global
$lc \leftarrow 0$: contador local, incrementa en cada tick del reloj.
$d \leftarrow 0$: retardo unidireccional medido hacia el *peer* p.

TRANSITION:

T0: Después de establecer la conexión con p.
$lc \leftarrow gc$
Send $(Init, lc)$

T1: Después de recibir $(Init,\ c)$ desde p.
Enviar $(Init - Ack, c)$

T2: Luego, recibiendo $(Init - Ack,\ c)$ desde p.
$d \leftarrow (lc - c - \alpha)/2$

T3: Después de *timeout*.
Enviar$(Beacon, gc)$

T4: Luego, recibiendo $(Beacon,\ c)$ desde p.
$lc \leftarrow \max\ (lc, c + d)$

Fase INIT. El propósito de la fase INIT es medir el retardo unidireccional entre dos *peers*. Inicia cuando dos puertos se encuentran físicamente conectados y empiezan una comunicación (establecimiento de conexión). Cada *peer* mide el retardo unidireccional calculando el tiempo entre enviar un mensaje INIT y recibir su correspondiente INIT-ACK, este *RTT* se lo divide entre dos (T0, T1, y T2 en el algoritmo 1).

Fase BEACON. Durante esta fase, periódicamente dos puertos intercambian sus contadores locales para realizar la re-sincronización (T3 y T4 en el algoritmo 1). Debido a la desviación del oscilador, el *offset* entre dos contadores locales se incrementa con el tiempo. Un puerto ajusta su contador local seleccionando el máximo de los contadores locales y remotos al recibir un mensaje BEACON de su *peer*. Dado que los mensajes de BEACON se intercambian con frecuencia, cientos de miles de veces por segundo (cada pocos microsegundos), el *offset* se puede mantener al mínimo.

Escalabilidad y saltos múltiples. Los *switches* multipuertos contienen entre dos a noventa y seis puertos en un solo dispositivo que deben ser sincronizados. DTP siempre selecciona el máximo de todos los contadores locales como el valor para el contador global (T5 en el algoritmo 2). Luego, cada puerto transmite el contador global en un mensaje

Algoritmo 2 DTP dentro del un dispositivo de red

STATE:
gc: contador global.
$\{lc_i\}$: contador local.
TRANSITION:
T5: para cada tick del reloj
$gc \leftarrow \max(gc + 1, \{lc_i\})$

BEACON (T3 en el algoritmo 1). Seleccionando el valor máximo permite a cualquier contador incrementarse de manera monótona a la misma velocidad y permite escalar a DTP: el valor máximo del contador se propaga por todos los dispositivos de red por medio de mensajes BEACON y su envío frecuente mantiene sincronizados los contadores globales. En redes dinámicas cuando un dispositivo se enciende, los contadores locales y globales se establecen en cero. El contador global empieza a incrementarse cuando algún contador local incrementa. Sin embargo el contador global vuelve a cero cuando todos los puertos se quedan inactivos. En consecuencia, los contadores globales y locales de un dispositivo que se inserta en la red siempre tienen los valores más bajos respecto a otros dispositivos en la red DTP. En este caso se utiliza el mensaje especial BEACON_JOIN para hacer un ajuste considerable de los contadores. Este mensaje es comunicado después del mensaje INIT_ACK, para asegurar el valor máximo entre dos contadores locales. Cuando un dispositivo de red con múltiples puertos recibe un mensaje BEACON_JOIN en uno de sus puertos, este ajusta su contador global y propaga el BEACON_JOIN con este nuevo valor del contador global hacia lo otros puertos.

2.1.6. TSC

TSC es un modelo de registro de 64-bits que realiza el conteo de los ciclos del oscilador de la CPU desde su arranque. Se encuentra presente en todos los procesadores de arquitectura x86. Este registro es accesible realizando una llamada a la instrucción del lenguaje ensamblador RDTSC, la cual devuelve la llamada cargando los registros EDX:EAX. Para obtener el tiempo en segundos, basta con dividir el valor devuelto por TSC entre la frecuencia del procesador (en Hz). Por ejemplo, para un procesador que corre a una frecuencia de 800 MHz, el registro de TSC puede proveer una resolución de tiempo de 1.25 nanosegundos [12].

Algunas de las ventajas del uso de TSC como fuente de referencia de tiempo son:

- La operación de lectura se lleva a cabo mediante una instrucción de código máquina, con lo que el procesamiento requerido ante cada lectura es mínimo.
- Tiene la resolución de la frecuencia de reloj del procesador, con lo que en la actualidad, con procesadores con relojes del orden del Gigahertz se obtienen resoluciones del orden de los sub-nanosegundos.

En la actualidad los relojes de software en PC's se ajustan por medio del protocolo NTP. Sin embargo, esta precisión es suficiente para aplicaciones estándar, siendo una solución de bajo costo que puede ser utilizada en redes LAN o Internet. El estándar IEEE-1588 para redes LAN provee métodos para sincronización muy precisos, para tener valores de precisión en el nivel de microsegundos, pero utilizando hardware específico para el caso. Una implementación de este tipo tiene sus limitaciones, el costo que requiere el uso de este tipo especial de hardware y la incompatibilidad con el hardware antiguo. Estas restricciones hacen que esta solución no sea la mejor opción como protocolo de sincronización sobre redes LAN. Una opción de término medio capaz de balancear el costo de *hardware* y obtener un alto nivel de precisión es por medio de un método que utiliza el contador de registros TSC, presente en los procesadores modernos que se encuentran en PC's de uso común.

Un reloj real, con denotación t, corre a una tasa de 1 segundo por segundo y con un origen $t = 0$ en algún instante arbitrario. En la práctica, un reloj imperfecto denotado como

'reloj', lee $C(t)$ en el instante real t. La resolución de $C(t)$, es la unidad más pequeña que debe ser actualizada. El *offset* $\theta(t)$ del reloj, es el error frente al tiempo real t:

$$\theta(t) \equiv C(t) - t. \tag{2.1}$$

El *skew* o desvío γ es aproximadamente la diferencia entre la velocidad del reloj y la velocidad de referencia. El modelo que captura esta idea en esta forma se lo llama *Simple Skew Model* (SKM) [5]. Este asume que:

$$\textbf{SKM: } \theta(t) = \theta_0 + \gamma\, t. \tag{2.2}$$

Refinando el concepto de *skew* considerando el siguiente modelo general

$$\theta(t) = \theta_0 + \gamma\, t + \omega(t), \tag{2.3}$$

donde el *simple skew* γ es el coeficiente determinístico de la parte lineal, $\omega(t)$ es el residuo en $\omega(0) = 0$, el cual encapsula las desviaciones del SKM, pudiendo tener componentes aleatorios y determinísticos. La estabilidad del oscilador parcialmente caracteriza $\omega(t)$ clasificando por escala de tiempo τ, los errores relativos de *offset*.

$$y_\tau(t) = \frac{\theta(t+\tau) - \theta(t)}{\tau} = \gamma + \frac{\omega(t+\tau) - \omega(t)}{\tau}. \tag{2.4}$$

En otras palabras, $y_\tau(t)$ es el *skew* promedio en el tiempo t, cuando se mide en la escala de tiempo τ y consta principalmente de *skew* γ más las variaciones no lineales que impactan en la escala de tiempo τ.

Con el uso de TSC, el valor de este registro en el tiempo t es denotado como $\mathrm{TSC}(t)$, considerando $\mathrm{TSC}_0 = \mathrm{TSC}(0)$. La construcción del reloj $C(t)$ de este contador es basado en el modelo de *simple skew*, para el cual el oscilador tiene un periodo constante p, insinuando que $t = (\mathrm{TSC}(t)\text{-}\mathrm{TSC}_0)p$. Con lo que podemos estimar $\hat{p}$ de p y $\widehat{\mathrm{TSC}_0}$ de TSC_0, la definición del reloj $C(t)$ es:

$$\textbf{SKM: } C(t) \equiv (\mathrm{TSC}(t) - \widehat{\mathrm{TSC}_0})\hat{p} = \mathrm{TSC}(t)\hat{p} + k, \tag{2.5}$$

donde la constante $k \equiv -\widehat{\mathrm{TSC}_0}\hat{p}$ intenta alinear al inicio de $C(t)$ y t, pero con cierto error. Si se observa que el error $p_\epsilon \equiv \hat{p} - p$ en el periodo estimado y $\mathrm{TSC}_\epsilon \equiv \widehat{\mathrm{TSC}_0} - \mathrm{TSC}_0$

en el inicio estimado, se obtiene un *offset* $\theta(t) = p_\epsilon/p \cdot t - \hat{p}\mathrm{TSC}_\epsilon$, el cual comparado con la ecuación 2.2, identifica a $\gamma = p_\epsilon/p = \hat{p}/p$ y $\theta_0 = C(0) = \mathrm{TSC}_0\hat{p} + k$.

Como la idea del SKM es, no mantenerse en todo el tiempo, sino que la estimación puede ser en diferentes tiempos. Una consecuencia de esto es, que la variación del *offset* sobre el tiempo, no es más una función de γ, sino que esta debe ser medida de forma independiente. Esto quiere decir que el *drift* (deriva) del reloj debe ser seguido. En la práctica podemos encontrar en dos formas un reloj corregido, dependiendo si se requiere diferencia de tiempo o tiempo absoluto, para lo cual tenemos:

$$\text{Reloj diferencia: } C_d(t) \equiv \mathrm{TSC}(t)\hat{p}(t), \tag{2.6}$$

$$\text{Reloj absoluto: } C_a(t) \equiv \mathrm{TSC}(t)\hat{p}(t) + k, \tag{2.7}$$

donde $\hat{p}(t)$ es el periodo actual de estimación y $\hat{\theta}(t)$ es la estimación actual del *offset* de $C(t)$, que se pretende corregir. Únicamente definiendo dos relojes de esta forma se puede ofrecer un reloj absoluto sin alterar el *offset* regular del reloj de TSC. Este reloj absoluto menos preciso, debe ser utilizado cuando realmente se requiera. El reloj diferencia, que sirve para medir diferencias de tiempo resulta mucho más preciso cuando es utilizado en intervalos de medición $\Delta(t)$, los cuales son más pequeños respecto a la escala τ del SKM. Sobre esta escala, el *drift* del reloj es significativo y la diferencia de tiempo será más precisa utilizando el reloj absoluto.

2.1.7. Estimación unidireccional de retardos sin sincronismo de relojes

Realizar mediciones de tráfico, resulta de mucha importancia para conocer el desempeño que pueden ofrecer ciertos servicios como, por ejemplo, aplicaciones de streaming, video conferencias, transacciones entre cliente servidor, etc. Dichos servicios, dependen de las características del camino entre el origen y el destino. Por esta razón, es necesario para ciertos casos, medir el retardo unidireccional del tráfico, para lo cual se necesita que los relojes de los puntos medidos se encuentren sincronizados. Si los relojes entre estos puntos, no se encuentran sincronizados, al realizar la medición del retardo unidireccional, la diferencia de relojes generaría impresiciones en la medición. Un enfoque para la estimación del retardo unidireccional entre nodos de una red, sin utilizar ninguna fuente de sincronización de tiempo se muestra en [8]. Se basa en realizar mediciones unidireccionales múltiples entre nodos vecinos (ver figura 2.3) para estimar los retardos

unidireccionales, mediante la resolución de un sistema de ecuaciones sobre determinado nodo, optimizando el valor general de una función objetivo. La misma que es afectada por toda la topología de la red y no únicamente por mediciones individuales entre nodos. Se aprovecha las mediciones para obtener los límites necesarios en los retardos unidirec-

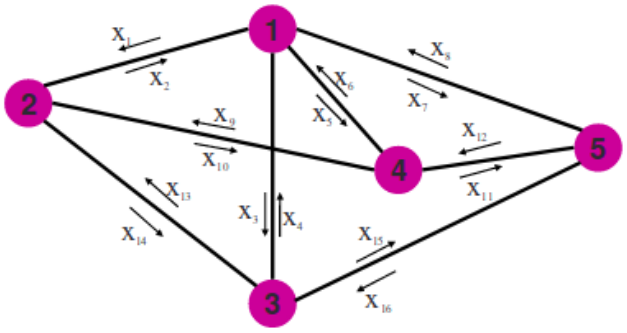

FIGURA 2.3: Red de 5 nodos con 16 conexiones [8]

cionales y derivar el número de restricciones independientes que puedan ser obtenidas. Estas restricciones son usadas para la optimización, junto con una función objetivo que provee la "mejor" estimación para el retardo unidireccional.

Se han examinado dos funciones objetivo. La primera función está basada en principio del Error de Cuadrados Mínimos (LSE). Según este principio, lo que se busca con esta es minimizar el error cuadrático. La segunda función objetivo es basada en el principio de la Entropía Máxima (ME), y lo que se busca es maximizar la entropía. Una de las formas para la implementación esta solución es mediante el uso de paquetes NTP o ICMP. Se ha demostrado luego de varias experimentaciones que ambos esquemas superan considerablemente el método de dividir el tiempo de ida y vuelta a la mitad.

El modelo de la topología de la red se conforma de un grupo de N nodos N donde cada nodo es referido como Λ_i con $i = 1, 2, \cdots, N$; los enlaces $e_{i,j}$ que conectan directamente dos nodos Λ_i y Λ_j, llamando $\mathcal{E}$ al conjunto de estos enlaces. Se denota como G_i al conjunto de nodos vecinos de Λ_i. Mientras los nodos en la red no se encuentren sincronizados, se llama τ_i al *offset* del reloj de Λ_i respecto al "Tiempo Universal"y τ_{ij} al *offset* relativo entre el reloj de Λ_i y el de Λ_j.

Con componentes determinísticos y estocásticos que forman parte del retardo, se puede asumir que el tiempo de inserción y propagación de los bits en la red son constantes. Mientras el encolamiento forma parte del componente estocástico, se denota como x_{ij}

el retardo unidireccional entre los nodos Λ_i y Λ_j. Siendo c_{ij} y v_{ij} la parte constante y variable respectivamente ($x_{ij} = c_{ij} + v_{ij}$).

Mediante el envío de paquetes de prueba k, el tiempo de transmisión desde Λ_i hacia el receptor Λ_j, es posible obtener dos *timestamps*, denotando como $T_{ij}^{[k]}$ al *timestamp* del paquete transmisor y $R_{ij}^{[k]}$ al *timestamp* del paquete receptor, así como $x_{ij}^{[k]}$ el retardo unidireccional de k. La diferencia de estos tiempos se denota:

$$\Delta T_{ij}^{[k]} = T_{ij}^{[k]} - R_{ij}^{[k]},$$

Esta diferencia no corresponde al retardo unidireccional, sin embargo si se suma el retardo experimentado por el paquete de prueba k, se tiene

$$\Delta T_{ij}^{[k]} = x_{ij}^{[k]} - \tau_i + \tau_j = x_{ij}^{[k]} - \tau_{ij}.$$

Es importante observar que la suma de $T_{ij}^{[k1]} + T_{ji}^{[k2]}$ para algún paquete arbitrario $k1$ y $k2$ representa retardo en ambos sentidos de un paquete virtual, siendo $k1$ el paquete en el sentido Λ_i hacia Λ_j y regresa como $k2$ en el sentido Λ_j hacia Λ_i. Esto sigue como

$$\Delta T_{ij}^{[k1]} + \Delta T_{ji}^{[k2]} = x_{ij}^{[k1]} - \tau_{ij} + x_{ji}^{[k2]} - \tau_{ji} = x_{ij}^{[k1]} + x_{ji}^{[k2]},$$

Con lo cual el *offset* de los relojes de los vecinos no se ve afectada por la expresión anterior. El mismo estado se mantiene para cualquier ruta cíclica, por ejemplo; si $\Lambda_{i1} \rightsquigarrow \Lambda_{i2} \rightsquigarrow \cdots \rightsquigarrow \Lambda_{il} \rightsquigarrow \Lambda_{i1}$ es un ciclo arbitrario, entonces la suma representa el retardo de la ruta cíclica que tiene un paquete virtual cuando es enviado a lo largo del camino.

$$\Delta T_{i1i2}^{[k1]} + \Delta T_{i2i3}^{[k2]} + \cdots + \Delta T_{ili1}^{[kl]}.$$

Capítulo 3

Bancos de Ensayo

3.1. Introducción

Este capítulo hace referencia a los elementos de *hardware* y *software* utilizados. Se mencionan tres escenarios considerados para el desarrollo del algoritmo de sincronización. El primero de ellos, escenario base, en el cual dos dispositivos se conectan directamente entre si (espalda-espalda). El segundo de ellos, escenario exterior, involucra dos ciudades lejanas como son Los Ángeles, en Estados Unidos, y la Ciudad de Buenos Aires, en Argentina. Finalmente el tercero, denominado escenario interior, se localiza únicamente en la Ciudad de Buenos Aires, Argentina.

3.2. Elementos de *hardware*

Uno de los requerimientos importantes tomados en cuenta fue el *hardware* a utilizar. Como el algoritmo de sincronización requiere de dos dispositivos para poder sincronizar sus relojes, fue necesario colocar un computador en cada sitio, provisto de un módulo GPS. En el primer escenario referido previamente, se colocó un computador en Los Ángeles y otro en la Ciudad de Buenos Aires; y en el segundo, se colocaron ambos en la Ciudad de Buenos Aires. La condición principal requerida por un módulo GPS para su funcionamiento óptimo es estar físicamente en un lugar que disponga de una vista directa al cielo, para lograr adquirir la señal de los satélites. Además de lo antes mencionado, el computador en el cual se conecta el módulo GPS, tiene que estar físicamente cerca de

éste. En esas circunstancias, se buscó una solución que se adapte a los requerimientos técnicos y que además sea de bajo costo.

Raspberry Pi. Raspberry Pi [13] es una mini-PC de bajo costo. Básicamente, se trata de una placa que soporta varios de los componentes necesarios de un computador común; fue lanzado al mercado en el año 2012. A pesar de su pequeño tamaño (ver figura 3.1),

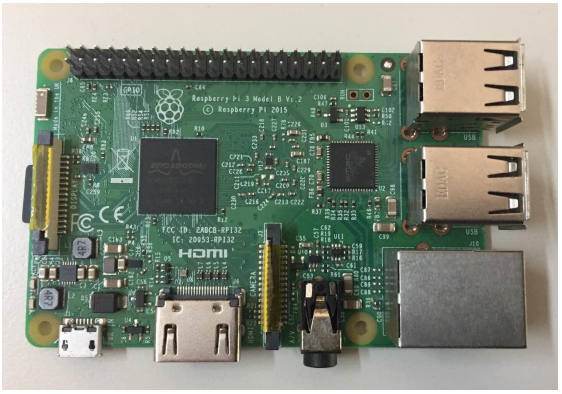

FIGURA 3.1: Equipo Raspberry Pi Modelo B, utilizado en el desarrollo de este proyecto.

sus prestaciones funcionales hacen que se sea considerada para muchos propósitos. Para este proyecto se utilizó el último modelo vigente que es el Raspberry Pi 3 modelo B. Los detalles técnicos del modelo empleado se pueden encontrar en la tabla 3.1. De acuerdo a

Raspberry Pi 3 Modelo B	
Procesador	Broadcom BCM2837 64Bit ARMv7 Quad Core Processor powered Single Board Computer running at 1250MHz
GPU	Videocore IV
Velocidad de procesador	QUAD Core @1250 MHz
Memoria RAM	1GB SDRAM @ 400 MHz
Almacenamiento	MicroSD
USB 2.0	4x USB Ports
Alimentación	2.5A @ 5V
GPIO	40 pin
Puerto Ethernet	Si
Wifi	Integrado
Bluetooth	Integrado

TABLA 3.1: Características de Raspberry Pi 3 Modelo B

las características del equipo, su rendimiento es óptimo, ajustándose a lo requerido para

el desarrollo de este proyecto. Es importante considerar que dentro de los 40 pines GPIO (*General Purpose Input/Output*) es posible habilitar UART (*Universal Asynchronous Receiver-Transmitter*), que son pines necesarios para la integración del módulo GPS.

En referencia al *software*, el mismo está diseñado para ejecutar el sistema operativo GNU/Linux, el cual es de código abierto, por lo que es posible descargarlo directamente desde el sitio web del fabricante [14]. Existen varias versiones de Linux (conocidas como distribuciones) que han sido portadas al chip BCM2837 de la Raspberry Pi, incluyendo Debian, Raspbian, Fedora Remix y Arch Linux. Para este proyecto, se utilizó la distribución Raspbian.

Módulo GPS. El módulo GPS (figura 3.2) utilizado en el desarrollo de este proyecto fue NEO 6M [15]; esta es una excelente alternativa de precisión y ecuación costo-beneficio. Es de un tamaño reducido, por lo que es fácilmente portable y se adapta a espacios

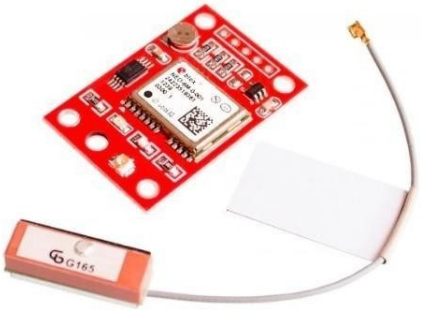

FIGURA 3.2: Modulo GPS Modelo NEO-6M, utilizado junto al dispositivo Raspberry Pi 3.

pequeños. Se comunica a través del puerto serial UART. En la tabla 3.2 se muestran sus principales características. Este modelo tiene incorporada la funcionalidad de pulso por segundo (PPS), con lo cual es posible conocer el inicio exacto de cada segundo del reloj del GPS, característica importante para disponer de precisión y estabilidad a largo plazo. La precisión del ofrecida por el PPS en este modelo es menor a 60ns en el 99 % del tiempo.

Instalación de *software*. Definidos los dos componentes físicos necesarios, lo siguiente fue la instalación del sistema operativo en el dispositivo Raspberry Pi y luego se actualizaron e instalaron los paquetes necesarios para los lenguajes de programación. Es

Modulo GPS NEO-6M	
Modelo	NEO-6M
Tipo	GPS
Alimentación	2.7-3.6v
Interface	UART
Caracteristicas	PPS

Tabla 3.2: Modulo GPS NEO-6M

importante indicar que el lenguaje de programación utilizado para el desarrollo de este proyecto fue exclusivamente C.

Para adaptar el módulo GPS en el dispositivo Raspberry Pi, fue necesario realizar algunos cambios en la configuración de los pines GPIO, de tal forma que se pudiera recibir la señal por medio de la interfaz UART. También hizo falta descargar los paquetes que permitieron interactuar con el GPS y, concretamente, con la señal PPS. Además, fue necesario instalar las herramientas de *pps-tools* [16], con la utilidad *ppstest*.

3.3. Escenarios

Según el planteamiento, el algoritmo de sincronización debió ser diseñado para su funcionamiento por medio de Internet. Por este motivo, se buscó y logró conseguir la autorización para colocar los equipos Raspberry Pi en tres diferentes redes, con lo cual se recrearon dos escenarios. Además, fue necesario crear un escenario base, con el objetivo de tener un punto de inicial de referencia. A modo de simplificación, se utilizará el término sonda, para referir a los equipos Raspberry Pi.

Escenario base. Este escenario inicial, consistió en colocar dos sondas conectadas entre si por medio de un *switch*. El objetivo de disponer de un escenario de este tipo, es para realizar mediciones que permitan calcular un valor base del error, el cual se encuentra afectado únicamente por las características de *hardware* de las sondas.

Escenario Exterior. Para este caso, el escenario consistió en tener interconexión entre redes ubicadas en dos ciudades de diferente país. Por un lado, se conectó una sonda en la red de USC (*University of Southern California*) en Los Ángeles, Estados Unidos. Y por el otro lado, la sonda se ubicó en la red del SMN (Servicio Meteorológico

Nacional) en la Ciudad de Buenos Aires, Argentina. En el extremo de USC se asignó una ip pública para el acceso desde Internet. La distancia de red entre los dos puntos en mención corresponde a 20 saltos. Los tiempos de respuesta en cada equipo de red de la traza, se muestra en la tabla 3.3.

Start: Mon Feb 13 16:03:54 2017	
HOST: raspbian-client	RTT avg
1.|– 10.10.96.1	0.346 ms
2.|– 200.16.116.171	0.891 ms
3.|– 10.199.10.1	1.290 ms
4.|– 10.0.10.9	1.631 ms
5.|– 200.0.204.154	23.336 ms
6.|– 200.0.204.58	130.279 ms
7.|– 200.0.207.10	142.215 ms
8.|– 162.252.70.47	156.388 ms
9.|– 162.252.70.44	165.275 ms
10.|– 162.252.70.14	180.125 ms
11.|– 162.252.70.21	185.468 ms
12.|– 162.252.70.22	188.124 ms
13.|– 162.252.70.24	196.684 ms
14.|– 137.164.26.200	196.989 ms
15.|– 137.164.27.249	197.455 ms
16.|– 128.125.251.227	197.295 ms
17.|– 128.125.255.146	253.186 ms
18.|– 128.125.251.148	197.548 ms
19.|– 204.57.6.1	200.874 ms
20.|– 204.57.7.3	197.397 ms

Tabla 3.3: Ruta que siguen los paquetes desde el origen hasta su destino. Origen: Buenos Aires - Destino: Los Ángeles

Escenario Interior. Este escenario se considera interior, debido a que las dos sondas utilizadas se instalaron en Ciudad de Buenos Aires, Argentina. En un extremo, una sonda fue instalada en la red de una empresa privada, localizada en el barrio de Puerto Madero. Y en el extremo contrario, se aprovechó y utilizó la misma sonda instalada en la red del SMN, la cual se ubica en el barrio de Palermo. Esto fue posible dado que los escenarios se desarrollaron en diferentes momentos de tiempo. Considerando que ambos barrios se encuentran en la misma ciudad, la cantidad de saltos de red entre los dos sitios es menor, en comparación al escenario exterior, además de que los tiempos de respuesta también son más bajos. Para ejemplificar lo antedicho, se realizaron trazas de ambos escenarios que me muestran en las tablas 3.3 y 3.4.

Start: Mon Oct 09 12:05:45 2017

HOST: raspbian-client	RTT avg
1.|– 10.10.96.1	0.366 ms
2.|– 200.16.116.171	1.447 ms
3.|– 190.104.229.33	1.631 ms
4.|– 190.104.192.129	1.861 ms
5.|– 200.0.17.130	6.434 ms
6.|– 190.210.110.233	75.352 ms
7.|– 190.210.118.85	5.764 ms
8.|– 190.210.118.129	2.721 ms
9.|– 190.210.124.154	3.447 ms
10.|– 200.123.171.153	9.554 ms

Tabla 3.4: Ruta que siguen los paquetes desde el origen hasta su destino. Origen: Buenos Aires, Localidad Palermo - Destino: Buenos Aires, Localidad Puerto Madero

Capítulo 4

Desarrollo del Algoritmo de Sincronización SIC

4.1. Introducción

En este capítulo se detalla el desarrollo del algoritmo de sincronización de relojes, que se ha denominado SIC (*Synchronizing Internet Clocks*). Se explica el modelo y la notación utilizada. Finalmente, se justifican las variables consideradas y se explica el modo de funcionamiento del algoritmo.

4.2. Metodología Propuesta

Para el desarrollo del algoritmo fue necesario definir la arquitectura a utilizar. Según el planteamiento del algoritmo, el cual considera la sincronización de relojes entre dos computadoras, la arquitectura que se ajusta para lograr este objetivo corresponde al tipo cliente-servidor. Bajo esta arquitectura, la metodología consiste en el intercambio de paquetes que contienen datos de *timestamps* entre los sitios considerados. El procedimiento se realiza como sigue (ver figura 4.1):

- En uno de los sitios, el cliente genera un paquete de información, que contiene su marca de tiempo $t1_C$ al momento de salir hacia la red.

- Cuando el servidor recibe el paquete lo marca con su tiempo $t2_S$; luego de un tiempo, envía el paquete y agrega el tiempo $t3_S$ en el instante en que lo realiza.
- Al momento de llegar el paquete al cliente, éste lo marca con un tiempo $t4_C$.

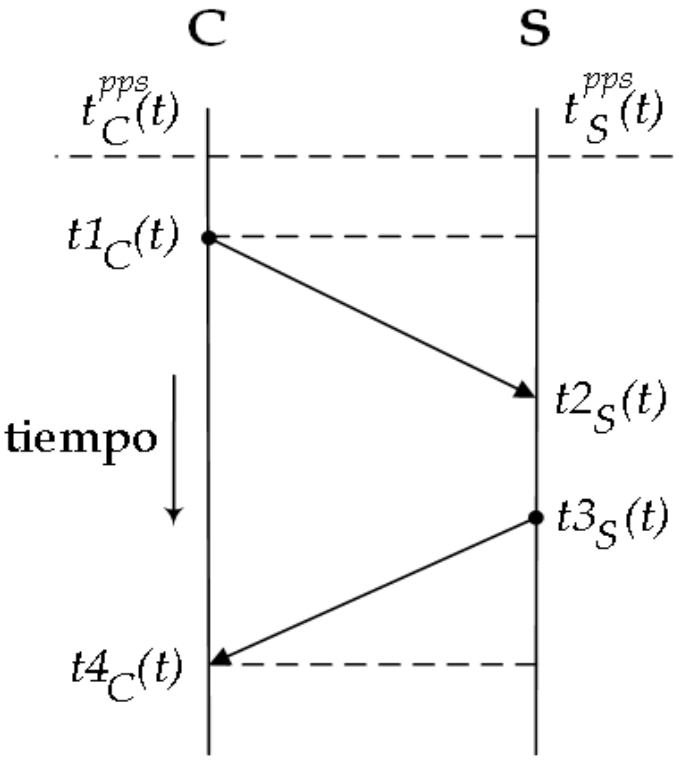

FIGURA 4.1: Arquitectura utilizada para el intercambio de mensajes de información entre el cliente y el servidor. En los casos utilizados para la verificación del sistema propuesto se utiliza un GPS que genera las señales t_C^{pps} y t_S^{pps}.

Hay que considerar que los tiempos $t2_S$ y $t3_S$ se registran con el reloj del servidor, mientras que los tiempos $t1_C$ y $t4_C$ corresponden al reloj del cliente; de allí los subindices C para el cliente y S para el servidor.

Con el fin de verificar la exactitud de la estimación de la diferencia de los relojes, se utiliza un reloj central de alta precisión, por lo que se incorporó un módulo GPS tanto en el servidor como en el cliente, el cual proveyó la señal de PPS (*Pulse Per Second*). Dicha señal recibida permite conocer con precisión el comienzo de cada segundo del tiempo ofrecido por el GPS, lo que posibilita registrar el estado de los relojes una vez por segundo. Según la figura 4.1, en el lado cliente, el tiempo obtenido por la señal PPS lo llamamos t_C^{pps}, mientras que en el lado servidor tenemos t_S^{pps}. Con estos dos valores, es posible obtener el valor de la diferencia entre los relojes cliente y servidor Φ, como se

muestra en la siguiente ecuación:

$$\Phi(i) = t_C^{pps}(i) - t_S^{pps}(i), \tag{4.1}$$

siendo i el número de segundo que se mide cuando se utiliza, en este caso, *epoch time*.

Modelo. La diferencia de tiempo, frecuencia y deriva (*rate y drift* en inglés, respectivamente) [17], son parámetros que describen el comportamiento de un reloj y difieren entre un reloj y otro. La relación entre dos relojes puede ser modelada bajo la siguiente ecuación:

$$\Phi(t) = k + f\,[C_C(t) - C_S(t)] + d\,[C_C(t) - C_S(t)]^2 + \ldots, \tag{4.2}$$

siendo t el instante de tiempo considerado, k es la diferencia inicial, f representa la relación entre el *rate* de C_C (reloj en el extremo C o cliente) y C_S (reloj en el extremo S o servidor), d representa la relación de los *drifts* y modela el cambio en la frecuencia. Si un paquete desde su origen toma un camino de ida, su regreso no necesariamente será por el mismo camino. Uno de los elementos que determinan la ruta que debe tomar un paquete son los equipos de red (*routers* y *switches*). Sabemos que Internet es una red que cuenta con gran cantidad de *routers* y *switches*, lo que hace imposible conocer la trayectoria del tráfico, ocasionando lo que llamamos asimetría de caminos. Esta asimetría es un problema que resulta imposible de cuantificar conociendo sólo los tiempos de los relojes de ambos extremos. Al no poder determinarla, nos basaremos en la hipótesis de simetría de los caminos, lo que redunda en que el término k de la ecuación 4.2 no podrá ser determinado con precisión. Sin embargo, si se logra estimar correctamente f de la ecuación 4.2, se puede conocer la diferencia de frecuencias de los relojes y, por lo tanto, se pueden corregir las mediciones en este sentido.

Como se observa en la figura 4.1, durante el intercambio del $i-$ésimo paquete entre C y S, se obtienen un registro de cuatro *timestamps*: $t1_C(i)$, $t2_S(i)$, $t3_S(i)$ y $t4_C(i)$. Estos tiempos requeridos para los análisis y experimentaciones, fueron obtenidos mediante la realización de mediciones entre C y S, con una frecuencia de 1 por segundo, durante 7 días.

De esta forma, cuando el paquete viaja en el sentido $C \rightarrow S$, el retardo del enlace corresponde a:

$$\tau_{CS} = t2_S(i) + \Phi(i) - t1_C(i). \tag{4.3}$$

De la misma manera, el retardo del enlace, cuando el paquete toma el camino de retorno en el sentido $S \rightarrow C$, es:

$$\tau_{SC} = t4_C(i) - t3_S(i) - \Phi(i). \tag{4.4}$$

De esta forma, el tiempo total de tránsito RTT (*Round-Trip Time*) de un paquete en el sentido ida $C \rightarrow S$ y regreso $S \rightarrow C$ es:

$$RTT = \tau_{CS} + \tau_{SC} + (t3_S(i) - t2_S(i)). \tag{4.5}$$

Como el objetivo es lograr sincronizar los relojes entre dos dispositivos, puede definirse:

$$\hat{\Phi}(i) = t1_C(i) + \frac{RTT(i) - (t3_S(i) - t2_S(i))}{2} - t2_S(i), \tag{4.6}$$

que es el estimador de Φ de la ecuación 4.1, en donde i es el instante de la medición y $t(i)$ es el tiempo de cada reloj tomando en cuenta el instante en el que se inicia la medición. Finalmente, realizando los reemplazos correspondientes en las ecuaciones, se obtiene:

$$\begin{aligned}
\hat{\Phi}(i) &= t1_C(i) + \frac{\tau_{CS} + \tau_{SC}}{2} - t2_S(i), \\
\hat{\Phi}(i) &= t1_C(i) + \frac{t2_S(i) + \Phi(i) - t1_C(i) + t4_C(i) - t3_S(i) - \Phi(i)}{2} - t2_S(i), \\
\hat{\Phi}(i) &= t1_C(i) + \frac{t2_S(i) - t1_C(i) + t4_C(i) - t3_S(i)}{2} - t2_S(i).
\end{aligned} \tag{4.7}$$

En condiciones de ausencia de tráfico, la ecuación 4.7 permitiría mantener los valores de $\hat{\Phi}$ sin mucha variación en el corto y largo plazo. Sin embargo, existen diversos factores, como puede ser el retardo, entre otros, que ocasionan que la estimación de Φ bajo la ecuación planteada no sea suficiente para mitigar dichos factores.

En [18] se describen los componentes que conforman el retardo de la red, donde éste aumenta salto a salto. Es importante recordar que los caminos de ida no necesariamente son los mismos que para la vuelta, por lo cual los caminos pueden no ser simétricos. Sin embargo, en este análisis se asumen los caminos simétricos, debido a que la problemática de la asimetría de caminos no ha podido ser resuelta aún. Si nos referimos al RTT, es importante destacar que, para ello, se suman todos los saltos a la ida y a la vuelta, tal

como lo muestra la ecuación 4.8, la cual es una fórmula conocida de Van Jacobson [18].

$$RTT = \sum_{j=1}^{N} t_{ins}(j) + t_{prop}(j) + t_q(j) + t_{cpu}(j), \tag{4.8}$$

donde N es la cantidad de saltos de ida y vuelta; $t_{ins}(j)$ es el tiempo de inserción del paquete de datos en la red; $t_{prop}(j)$ es el tiempo que demora en propagarse las ondas electromagnéticas desde un punto físico a otro; $t_q(j)$ es el tiempo de encolado del paquete en el equipo de red; $t_{cpu}(j)$ es el tiempo que demoran los routers del camino en leer el paquete y designar cuál es el camino a seguir. La sumatoria de estas variables en cada salto de la red ofrece el tiempo total de tránsito de un paquete entre dos extremos de la red en su recorrido de ida y vuelta. De las variables en mención, t_{ins} y t_{prop} son variables deterministicas t_{ins} está en función de la longitud del paquete (L) y el ancho de banda (BW) y t_{prop} depende de la distancia entre origen y destino (d) y la velocidad de propagación (v). Con las capacidades del equipamiento de red y las velocidades de conexión de hoy en día, estas variables se vuelven constantes. Sin embargo, t_q y t_{cpu} son variables aleatorias, ya que t_q es un parámetro complejo debido a que depende de la cantidad de tráfico en el dispositivo de red, siendo mayor el tiempo de encolado del paquete cuando mayor tráfico existe. En cuanto a t_{cpu}, depende del tiempo que tarda el router en determinar el puerto de salida de acuerdo con la dirección de destino. Sin embargo, si realizamos un análisis de la distribución estadística de las variables aleatorias t_q y t_{cpu} sobre los valores de RTT en cada segundo, como se muestra en la figura 4.2, se puede observar que el RTT, como cualquier latencia, tiene una distribución de cola pesada hacia la derecha, con lo cual se asume que se distribuye como una distribución Estable [19]. Esta distribución puede ser caracterizada por cuatro parámetros $S(\alpha, \beta, \gamma, \delta)$, donde $\alpha \in (0, 2]$, es el parámetro de que regula como decae la cola. $\beta \in (-1, 1)$, es el parámetro de desvío, $\gamma \in \mathbb{R} > 0$, es el parámetro de escala y $\delta \in \mathbb{R}$ es el parámetro de localización. La ecuación 4.9 muestra la función característica $g(k)$ de la distribución Estable, entre varios posibles parámetros para definirla.

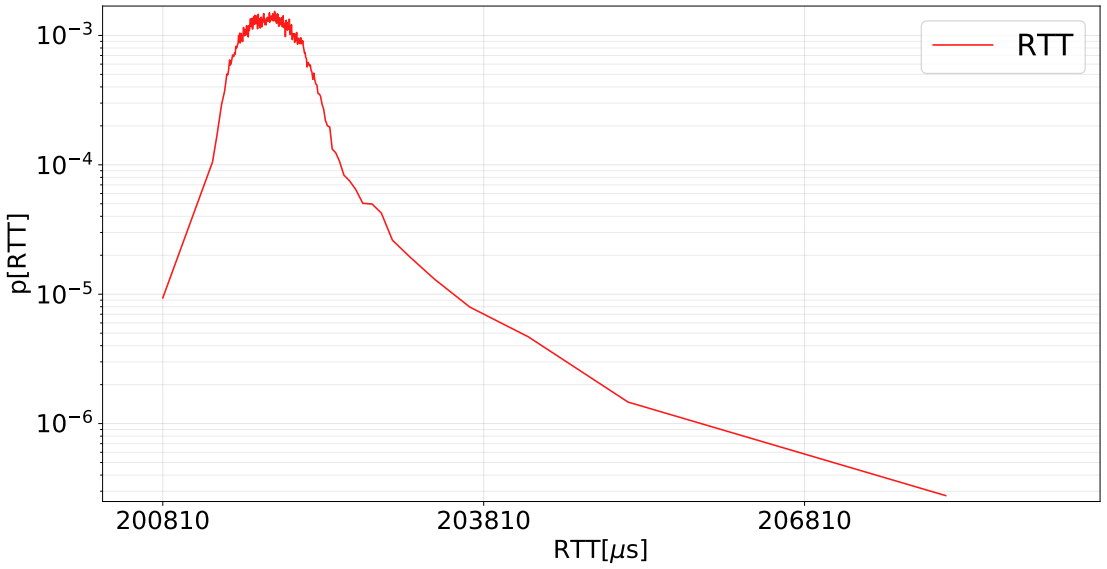

FIGURA 4.2: Distribución de frecuencias sobre valores de RTT; se observa que tiene la forma de una distribución de cola pesada hacia la derecha, sobre un total de 52378 mediciones.

$$g(k) = \exp\{\delta[ik\gamma - |k|^{\alpha}\omega(k;\alpha,\beta)]\} \tag{4.9}$$

$$\omega(k;\alpha,\beta) = \begin{cases} exp[-i\beta\Phi(\alpha)\text{sign}(k)], & \alpha \neq 1 \\ \pi/2 + i\beta \log|k|\text{sign}(k), & \alpha = 1 \end{cases}$$

$$\Phi(\alpha) = \begin{cases} \alpha\pi/2 & \alpha < 1 \\ (\alpha-2)\pi/2 & \alpha > 1 \end{cases}$$

En la ecuación 4.9 se puede entrever la complejidad de la distribución Estable, que también presenta algunos inconvenientes matemáticos. Ni las distribuciones ni las densidades de la Estable, se pueden expresar en términos de funciones elementales (como son polinomios, logaritmos, etc.), aunque con ciertas excepciones. La distribución Normal, es una de las pocas excepciones que tienen expresión para la distribución en términos de una función elemental, ya que la distribución Normal pertenece a la familia de distribución Estable. Esta información, es útil para aclarar el significado de los parámetros, ya que cuando la distribución Estable cede a la distribución Normal, $\alpha = 2$, $\beta = 0$ y γ y δ funcionan de la misma manera que σ (la varianza) y μ (la media) respectivamente.

Ciertos protocolos de sincronización, funcionan en base a las estimaciones del RTT mínimo. Tal como observamos en la figura 4.3, el RTT mínimo varía considerablemente. Dicha figura muestra el error del RTT mínimo en ventanas de 10 minutos, calculada de la siguiente manera:.

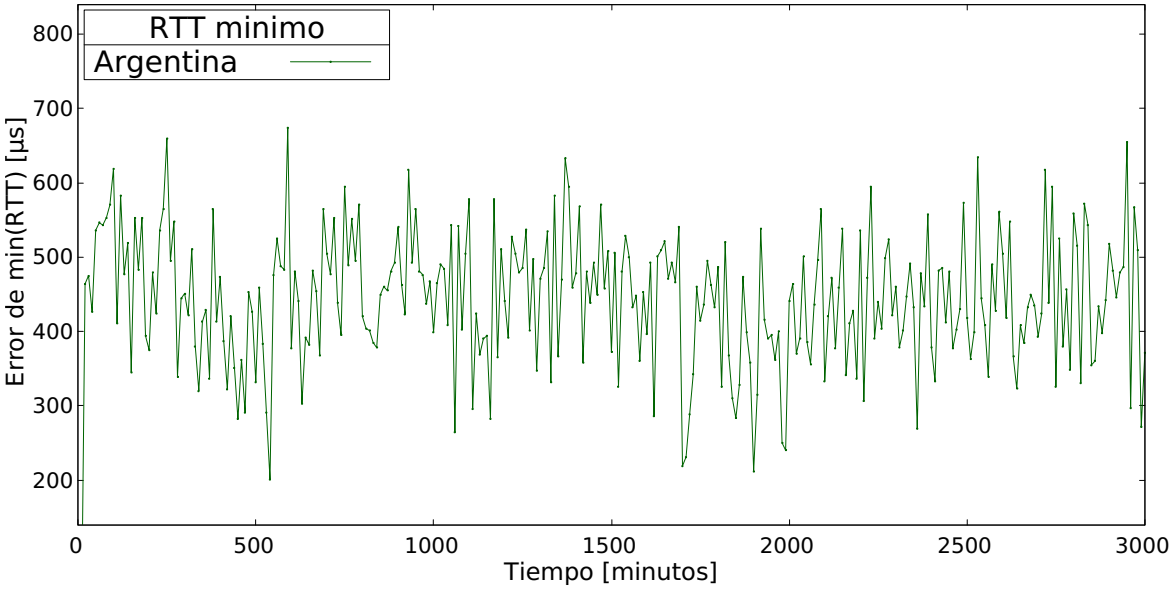

FIGURA 4.3: Error de RTT mínimo en ventanas de 10 minutos menos el RTT mínimo de una semana.

- En base a una semana de medición, se calcula el RTT mínimo ($mRTTw$) sobre todos los datos.
- Luego, separando los datos en ventanas de 10 minutos, se estima el RTT mínimo sobre cada una ($mRTT10m$).
- Finalmente, el error estimado es obtenido mediante la diferencia de: $mRTT10m - mRTTw$.

Para este caso, se ha utilizado los datos de las mediciones correspondientes al escenario interior, donde el tiempo más bajo en una semana de medición fue de 9431 microsegundos ($mRTTw$). Se puede notar que en algunos casos el valor de error puede ser mayor a 800 microsegundos. Este error es consecuencia del comportamiento estadístico del RTT que, como se mencionó anteriormente sigue una distribución estable.

Una propiedad matemática, como la suma y la resta de variables aleatorias con distribución Estable, dan lugar a otra con distribución Estable [20]. Además, la misma propiedad

aplica para dos variables aleatorias independientes e idénticamente distribuidas, da lugar a una distribución de probabilidades simétrica. En este caso la resta de retardos con distribución Estable hacia la derecha da lugar a una distribución simétrica. Como una propiedad de las distribuciones simétricas, entonces lo es para las estables simétricas, que la mediana se ubica en el valor central de la distribución, siendo también la moda de la misma. En nuestro caso, hemos observado que la ecuación 4.7 resulta en una distribución con cierta simetría. Según lo observado en la figura 4.4, obtenida a partir de la ecuación 4.7 aplicada a los mismos datos de la figura 4.2, observamos cierta simetría y, por lo tanto, usaremos la mediana como estimador de Φ.

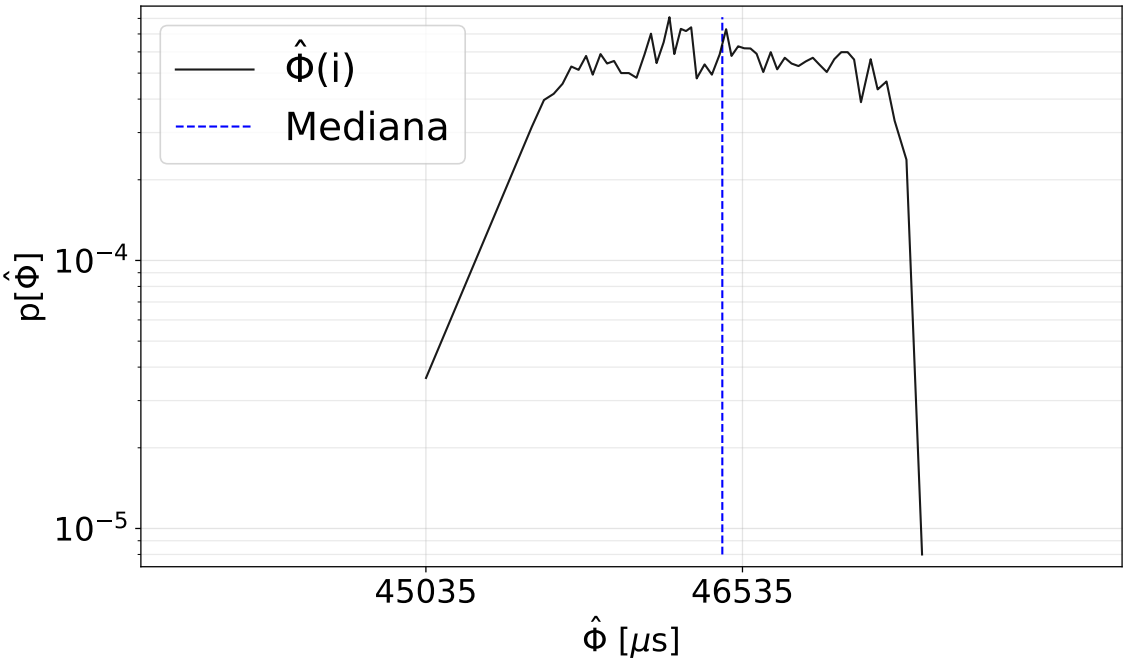

FIGURA 4.4: Distribución de frecuencias sobre valores de Φ.

Si observamos la figura 4.5, los valores instantáneos de Φ a lo largo del tiempo, son muy irregulares. Si bien, estos datos corresponden a mediciones reales, realizadas una vez por segundo, se presume que dicha variabilidad puede ser consecuencia de las variables que componen el retardo. Como sabemos, t_c y t_{cpu} siguen una distribución estable, esta distribución no tiene una representación funcional. Por lo tanto, en nuestro modelo representado en la ecuación 4.2, no lo hemos podido incluir como parte de la forma general que representamos la diferencia de relojes.

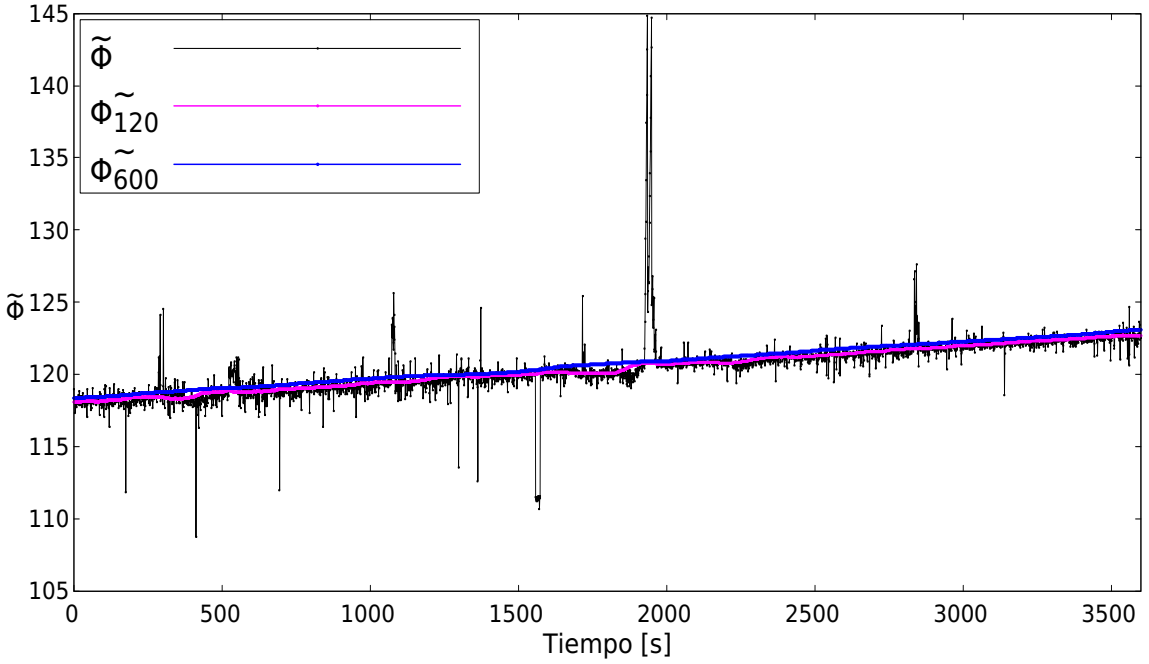

FIGURA 4.5: Error instantáneo y corrección mediante la inclusión de la ventana de sincronización de 120 y 600 mediciones.

Con la inclusión de la mediana aplicada sobre los valores de Φ, se pretende en cierta forma suavizar las variaciones que ocurren sobre la red. Inicialmente, y de forma arbitraria, para calcular la mediana en el momento $t(i)$, se usó la ventana $[t(i-120), t(i)]$, pudiendo observarse una mejora considerable. No obstante, se pudo experimentar y observar que al ampliar la ventana de estimación en una ventana $[t(i-600), t(i)]$, la mejora es mucho más significativa al verse un mayor alisamiento de las variaciones. Siendo importante indicar que los valores de la ventana están expresados en segundos, y que éste es un sistema de tiempo real, por lo tanto es no-causal. Lo explicado anteriormente se puede observar en la misma figura 4.5. Estos dos procedimientos los denominamos respectivamente como $\widetilde{\Phi_{120}}$ y $\widetilde{\Phi_{600}}$.

A pesar de la mejora que aporta la estimación de la mediana, si miramos un poco más en detalle, aún se puede observar cierta variación. Según la tendencia general de los datos, consideramos posible aplicar una función de aproximación que se ajuste y corrija dicha tendencia, de tal forma que se mantenga constante. De esta manera, se realizó un ajuste mediante la estimación de la recta por mínimos cuadrados, debido a que buscamos estimar el valor de f planteado en nuestro modelo bajo la ecuación 4.2; y bajo el modelo de SKM (*Simple Skew Model*) [5], f representa el coeficiente de la pendiente de la recta estimada. Este método lo denominamos como *LinEst*. Según se puede observar en la

figura 4.6, con la estimación de la recta se logra mantener cierta regularidad sobre los valores de Φ. Nótese que la estimación de la recta se realiza en ventanas de 1 minuto, por lo que se busca corregir la pendiente en este intervalo. Si bien, para la estimación de la recta se experimentó con diferentes ventanas, se pudo observar que mientras más se ampliaba la ventana, el error aumentaba. De esta manera, se fue realizando un ajuste hasta finalmente establecer en 1 minuto, pudiéndose observar que ventanas más pequeñas a ese intervalo, el error no varía significativamente.

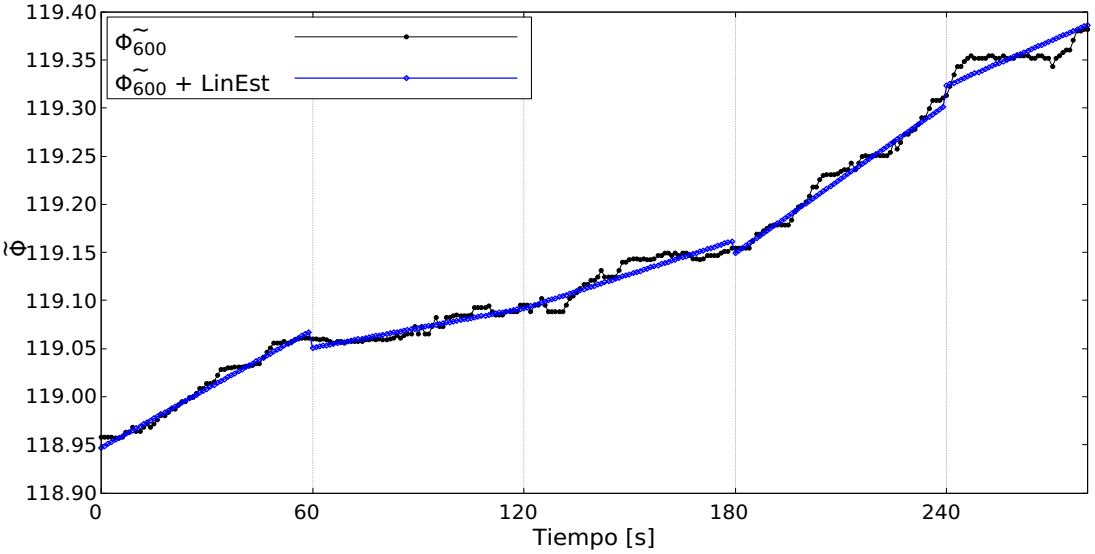

FIGURA 4.6: Comparación del método de la mediana y el de estimación de la recta en ventanas de 60 segundos.

Posteriormente a la estimación de la recta, se puede observar cómo los valores de la pendiente van variando cada minuto. Dado que el error aún sigue siendo importante, y que observamos que se debía a oscilaciones en el término de la frecuencia, decidimos agregar un filtro ARMA (*Autoregressive Moving Average*) sobre la recta estimada, lo que ayudó a suavizar la estimación por medio de la predicción de las variables de la recta; es decir, a partir del comportamiento obtenido con anterioridad, los valores de las variables del minuto actual están en función de los valores obtenidos en el minuto anterior, que a su vez dependen de los del minuto previo, como se puede observar en la figura 4.7. Este filtro, también se ha utilizado en varios protocolos de Internet, como por ejemplo, en el RTO (*Retransmission Time-Out*) de TCP. Una visión un poco mas extendida en el tiempo, como se puede observar en la figura 4.8, muestra un comportamiento oscilatorio que lo hemos observado en varias oportunidades; por lo que consideramos que el proceso ARMA era una buena solución.

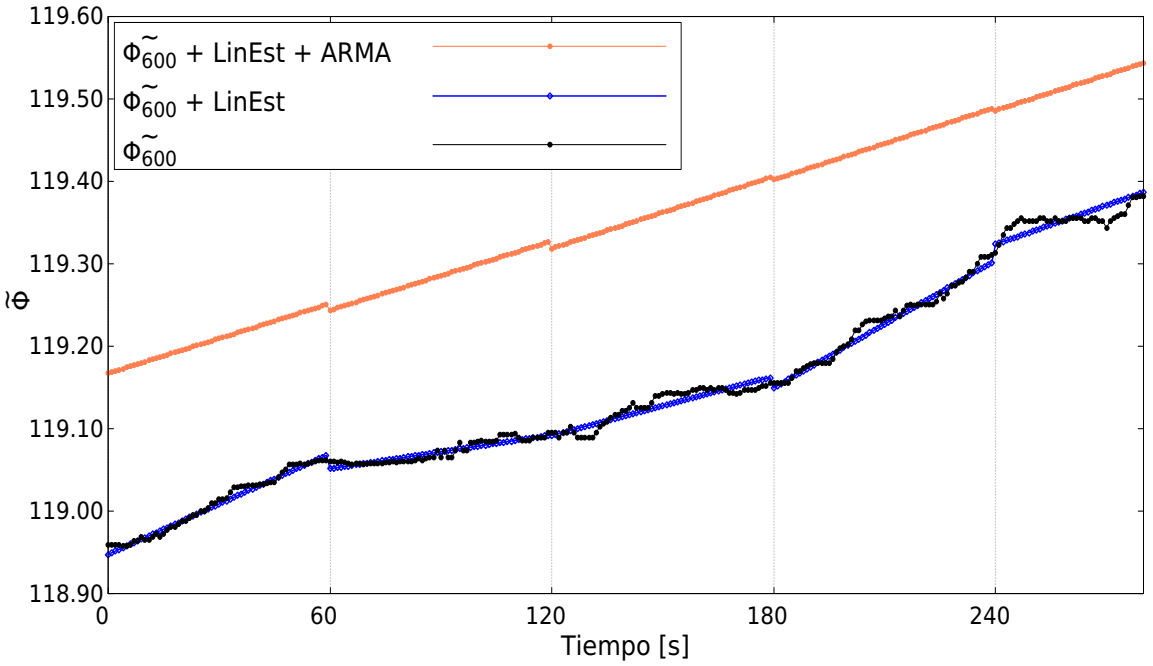

FIGURA 4.7: Inclusión del método de autorregresión (ARMA) para estabilizar los valores de Φ.

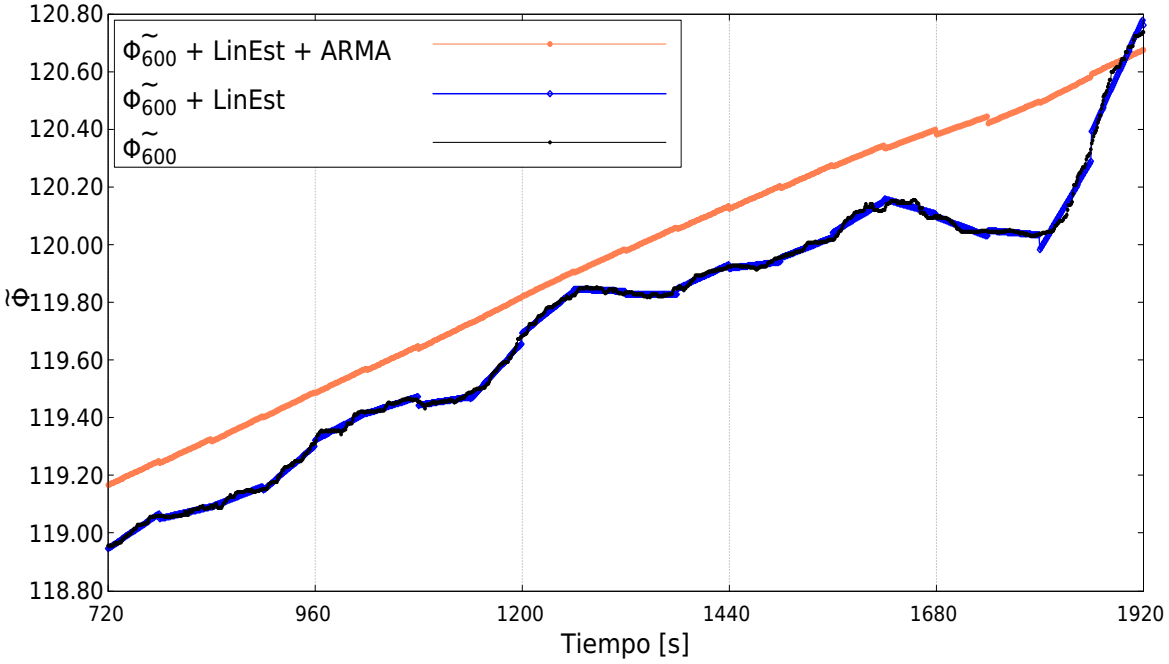

FIGURA 4.8: Inclusión del método de autorregresión (ARMA) extendida en el tiempo..

Cambios de ruta Durante el proceso de desarrollo del algoritmo se pudieron experimentar varias cuestiones; una de ellas se refiere a los cambios de ruta que toman los paquetes durante el proceso de intercambio entre el cliente y el servidor. Al realizar un análisis en base a los retardos obtenidos mediante el RTT, se pudo comprobar un comportamiento particular. Según se puede observar en la figura 4.9, en el análisis es posible notar un cambio considerable sobre estos valores. Esta variación se considera que se genera debido a un cambio en la ruta de los paquetes en la red, ya que, como se indicó anteriormente, el algoritmo trabaja sobre redes reales por medio de Internet por lo que este fenómeno no es controlable, resultando inevitable de eludirlo.

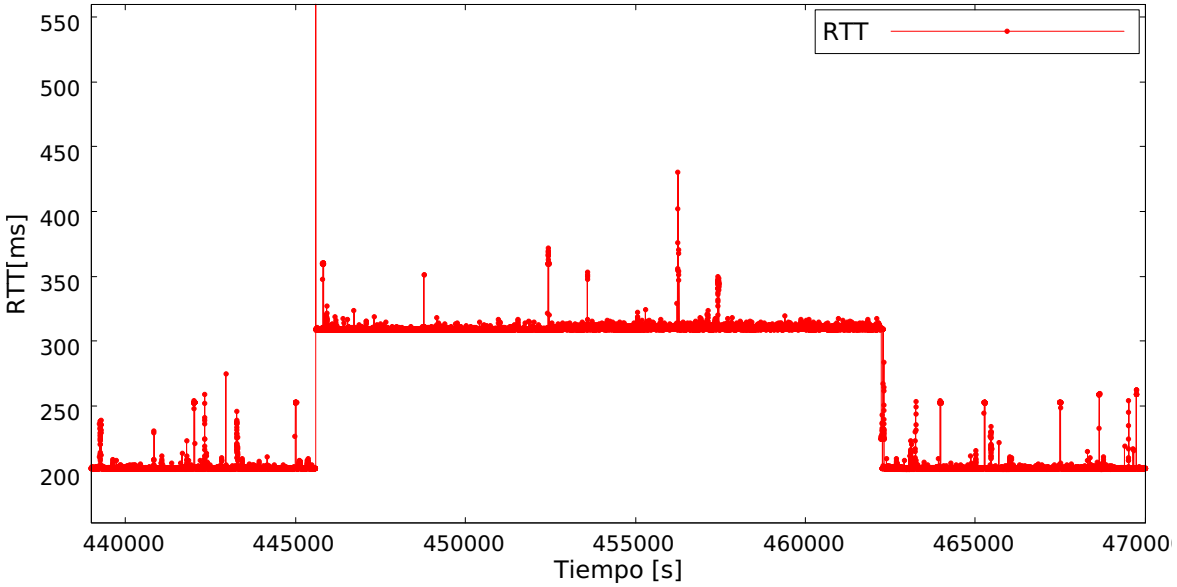

FIGURA 4.9: Variación de RTT, cuando existe un cambio en la ruta que siguen los paquetes en la red.

Poder detectar este comportamiento es muy importante, ya que de no hacerlo los valores resultantes, al generarse eventos de este tipo, hacen que el sincronismo se pierda y los factores de sincronización resulten en valores inesperados. Dichos cambios afectan al algoritmo, por lo cual es importante considerarlos ya que permiten definir estados del algoritmo, por lo que es posible conocer cuándo el cliente se encuentra o no sincronizado con el servidor

Estimación del error. Como se hizo mención en la descripción de la metodología propuesta, para la verificación de la exactitud en la estimación de la diferencia de los relojes se utiliza un reloj central de alta precisión como lo es el GPS junto con la señal

de PPS, tanto en el servidor como en el cliente; esto permite conocer con precisión el comienzo de cada segundo del tiempo ofrecido por el GPS, lo que posibilita registrar el estado de los relojes una vez por segundo. Según la ecuación 4.1, el Φ referencial corresponde a la diferencia de los tiempos ofrecidos por los GPS.

$$\Phi(i) = t_C^{pps}(i) - t_S^{pps}(i).$$

La métrica considerada para estimar el error es MTIE (*Maximum time-interval error*), que es una de las principales métricas sobre el dominio del tiempo consideradas para medir la estabilidad en la frecuencia de los relojes [21], según los estándares requeridos en telecomunicaciones. Con el uso de MTIE, podemos tener una medida aproximada de la desviación de tiempo de los relojes de C y S, con respecto a los relojes de los módulos GPS. Conociendo el valor de $\hat{\Phi}$ correspondiente a la ecuación 4.7, es posible realizar la diferencia en cada instante i, con lo cual el TE (*Time Error*) queda definido como:

$$TE(i) = \hat{\Phi}(i) - \Phi(i). \tag{4.10}$$

La variación del error en un intervalo S, empezando en t_0, se conoce como *Time Interval Error* $TIE_{t_0}(S)$ y se define como:

$$TIE_{t_0}(S) = TE(t_0 + S) - TE(t_0). \tag{4.11}$$

Con lo cual, la función MTIE se define como:

$$MTIE(S,T) = \max_{0 \leq t_0 \leq T-S} [MTIE_{t_0}(S)], \tag{4.12}$$

que nos indica la máxima variación pico a pico del TE durante el intervalo S, sobre un periodo de mediciones T (ver figura 4.10). Indicando que el límite de T para nuestro caso, corresponde a 7 días.

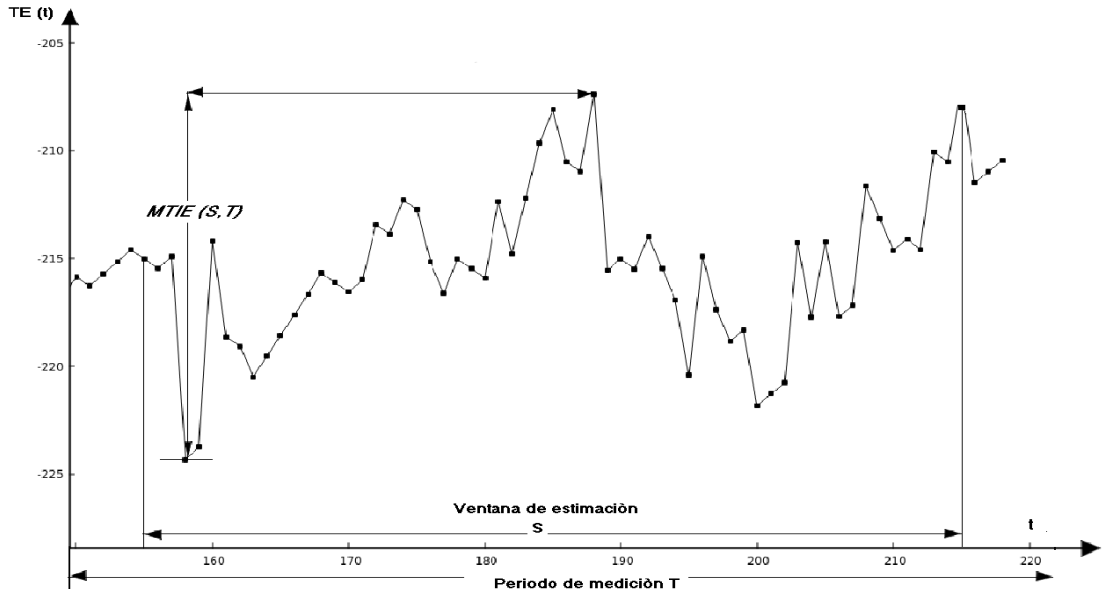

FIGURA 4.10: Métrica utilizada para la estimación del error (MTIE).

4.3. Descripción del algoritmo SIC

En esta sección se va realizar una descripción detallada del algoritmo SIC, así como las variables empleadas. En el algoritmo 3, se muestra el pseudocódigo empleado. Es importante indicar que este algoritmo se ejecuta en el lado cliente.

SIC inicia enviando y recibiendo paquetes hacia y desde el servidor de forma infinita, durante este proceso el algoritmo puede tomar tres estados: NOSYNC (no sincronizado), PRESYNC (pre-sincronizado) y SYNC (sincronizado). Para poder validar lo antedicho, se requieren dos variables que son pre_{sync} y $synck$, las mismas que se definen en la línea 1 y que sirven para realizar las validaciones y los cambios de estado. En la misma línea, pre_{sync} y $epoch_{sync}$ se establecen con el valor INT_MAX (máximo valor para una variable de tipo entera), esto, con el objetivo que dichas variables se mantengan inalcanzables, mientras el algoritmo no les asigne un valor. Además, es necesario definir las estructuras de datos W_{RTT} que sirve para almacenar los valores del RTT, W_{median} y W_m para las medianas estimadas; así como las variables $actual_m$ y $actual_c$, requeridas para almacenar los valores de la recta durante cada estimación (línea 2). Previo a entrar en este bucle infinito de iteración, es necesario iniciar el algoritmo en estado NOSYNC, siendo importante indicar que, cuando el algoritmo establece este u otro estado como

Algoritmo 3 Algoritmo SIC que realiza el proceso de sincronización entre el cliente y el servidor.

1: $pre_{sync} \leftarrow INT\_MAX,\ synck \leftarrow false,\ epoch_{sync} \leftarrow INT\_MAX$
2: $W_{median} \leftarrow 0,\ W_m \leftarrow 0,\ W_{RTT} \leftarrow 0,\ actual_m \leftarrow 0,\ actual_c \leftarrow 0$
3: $err_{sync} \leftarrow epoch$
4: set($actual_m,\ actual_c,$ NOSYNC)
5: **foreach** timer($running\_time$) == 0
6: $(epoch, t_1, t_2, t_3, t_4, to) \leftarrow$ send_sic_packet($server,\ timeout$)
7: **if** ($to == false$) **then**
8: $W_m \leftarrow t_1 + \frac{t_2\ - t_1\ + t_4\ - t_3}{2} - t_2$
9: $W_{median} \leftarrow \text{median}(W_m)$
10: $W_{RTT} \leftarrow t_4 - \ t_1$
11: $RTT_l \leftarrow \min(W_{RTT}[0,\ \frac{size(W_{RTT})}{2} - 1]))$
12: $RTT_f \leftarrow \min(W_{RTT}[\frac{size(W_{RTT})}{2}, size(W_{RTT})])$
13: **if** $((|RTT_l - RTT_f| \leq err_{RTT} \cdot min(W_{RTT})))$ **then**
14: **if** ($epoch \geq pre_{sync} + P$) **then**
15: $synck \leftarrow true$
16: **end if**
17: **else**
18: $synck \leftarrow false\ , W_{median} \leftarrow 0,\ W_m \leftarrow 0,\ err_{sync} \leftarrow epoch$
19: $epoch_{sync} \leftarrow INT\_MAX - P,\ pre_{sync} \leftarrow INT\_MAX - P$
20: $actual_m \leftarrow 0,\ actual_c \leftarrow 0$
21: set($actual_m,\ actual_c,$ NOSYNC)
22: **end if**
23: **if** $((synck == true)\ \&\&\ (epoch \geq epoch_{sync} + P))$ **then**
24: $(m, c) \leftarrow$ linear_fit(W_{median})
25: $actual_c \leftarrow (1 - \alpha) \cdot c + actual_c \cdot \alpha$
26: $actual_m \leftarrow (1 - \alpha) \cdot m + actual_m \cdot \alpha$
27: $epoch_{sync} \leftarrow epoch$
28: set($actual_m,\ actual_c,$ SYNC)
29: **else**
30: **if** ($epoch \geq err_{sync} + MEDIAN\_MAX\_SIZE$) **then**
31: $pre_{sync} \leftarrow epoch$
32: $err_{sync} \leftarrow INT\_MAX - MEDIAN\_MAX\_SIZE$
33: **end if**
34: **if** ($epoch \geq pre_{sync} + P$) **then**
35: $(actual_m\ , actual_c) \leftarrow$ linear_fit(W_{median})
36: $synck \leftarrow true\ ,\ epoch_{sync} \leftarrow epoch$
37: set($actual_m,\ actual_c,$ PRESYNC)
38: **end if**
39: **end if**
40: **else**
41: $to \leftarrow false$
42: **end if**
43: **end for**

se muestra en la línea 4, va acompañado de los valores de $actual_m$ y $actual_c$. Los mismos que corresponden al resultado de la autoregresión de la recta estimada y que son los encargados de ofrecer los factores de sincronización al momento de ajustar el reloj del cliente respecto al servidor. Luego, por cada *epoch* (iteración actual), un paquete es enviado hacia el servidor mediante la función *send_sic_packet* (línea 6; nótese que los parámetros de esta función son la dirección IP del servidor y un temporizador), la cual devuelve los datos de los tiempos correspondientes. Si dicho temporizador retorna con un tiempo expirado, el paquete es descartado (línea 40) y se espera el siguiente *epoch*. De no cumplirse lo anterior, un valor de Φ es estimado y almacenado en la estructura W_m (línea 8); luego, una mediana es estimada con los datos de esta estructura y almacenada en W_{median} (línea 9), así como también un valor de RTT (línea 10). En las lineas 11 y 12, los valores de RTT son utilizados para verificar si existe una variación considerable, es decir, si el valor absoluto del mínimo RTT es mayor a un porcentaje del valor mínimo de la ventana completa. Si dicho cambio es detectado, el algoritmo reinicia todas las variables y se establece el estado del algoritmo en $NOSYNC$ (líneas 18- 21). Para poder tener sincronismo, es necesario completar un período de tiempo definido por $MEDIAN\_MAX\_SIZE$ (valor constante definido por el usuario); cuando la cantidad de paquetes recibidos completa este valor (línea 30), pre_{sync} toma el valor del tiempo *epoch* (linea 31). Entonces, cuando se llega al tiempo *epoch* igual a $pre_{sync} + P$ (linea 34), el estado del algoritmo cambia a $PRESYNC$ (linea 37), se estiman los primeros valores de los factores de sincronización que serán utilizados posteriormente (linea 35), *synck* se establece en *true* y $epoch_{sync}$ toma el valor de *epoch* (línea 36). Pasado un período en el cual *epoch* llega ser igual a $epoch_{sync} + P$, junto con la variable *synck* en *true* (linea 23), es posible obtener los valores de la recta (linea 24), además de realizar la autoregresión (linea 25 y 26) en función de los valores estimados en la línea 35, además de quedar establecidos para ser utilizados en el siguiente período. También, el estado se establece en SYNC en la linea 28.

En la figura 4.11, se puede observar gráficamente las transiciones que toma el algoritmo. Se observa que se inicia en un estado NOSYNC en el tiempo *err_sync* igual a *epoch*.

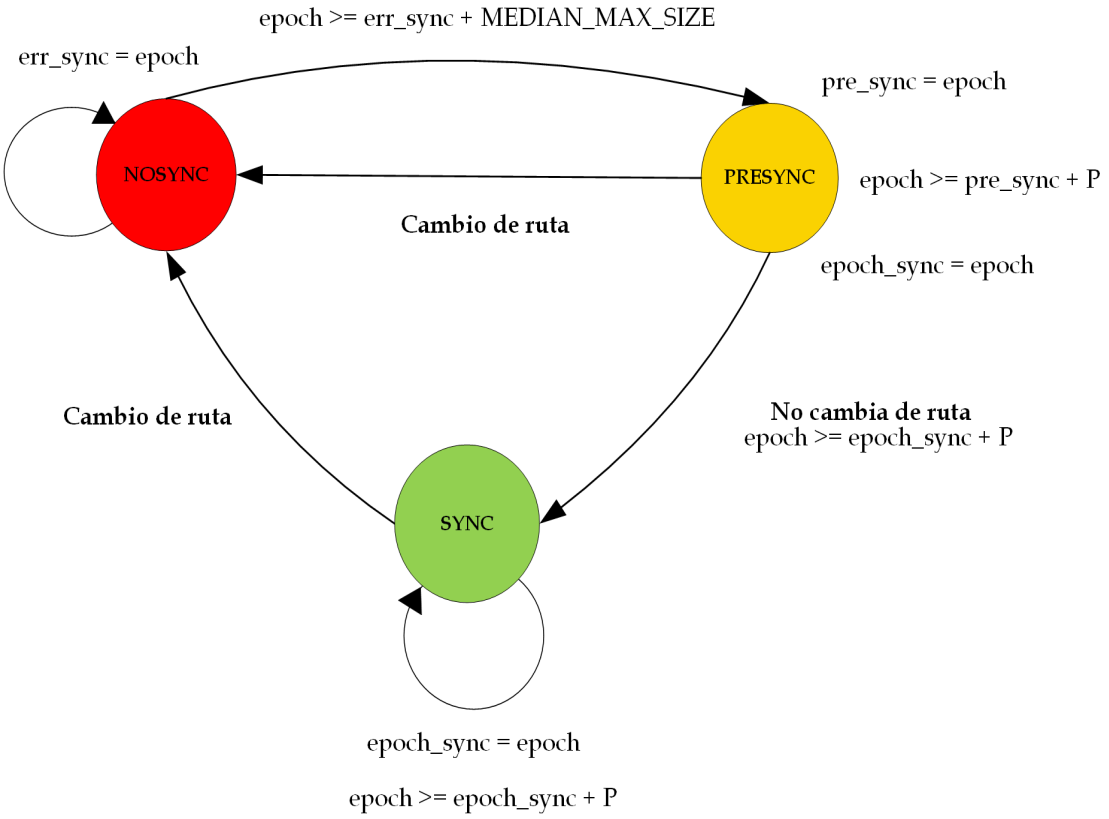

FIGURA 4.11: Estados que puede tener el algoritmo de sincronización SIC.

Luego, cuando *epoch* completa el período $err\_sync + MEDIAN\_MAX\_SIZE$, la variable pre_sync toma el valor *epoch* de ese momento. Entonces, *epoch* alcanza a pre_sync+P y el estado cambia a PRESYNC. En ese momento, $epoch\_sync$ toma el valor de *epoch*, si no existe algún cambio de ruta, el algoritmo continúa hasta alcanzar $epoch\_sync + P$. En este punto, si hay cambio de ruta, se regresa al estado NOSYNC; caso contrario, avanza hasta alcanzar SYNC. Nótese que el estado PRESYNC es alcanzado únicamente una vez, siendo SYNC el estado principal en el cual el algoritmo debe permanecer constantemente, siempre y cuando el comportamiento de la red no genere cambio en la ruta de los paquetes.

Capítulo 5

Implementación de SIC y Resultados

Este capítulo abarca los pasos seguidos para la implementación del algoritmo SIC. Se menciona principalmente la programación, además se detallan las funciones y variables principales del algoritmo. Por último, se muestran los resultados obtenidos en los escenarios tanto exterior como interior.

5.1. Organización del código

Según la arquitectura utilizada, fue necesario desarrollar los códigos para el cliente y para el servidor. Es importante señalar que la implementación completa del algoritmo ha sido desarrollada en el lenguaje de programación C. En la tabla 5.1, se muestran los archivos desarrollados para esta implementación.

Cliente	Servidor
|– fixed_window_bib.c	|– sic_server.c
|– sic_client.c	|– aux.h
|– aux.h	|– Makefile
|– Makefile	|– config.conf
|– config.conf	

Tabla 5.1

Cliente. La biblioteca fixed_window_bic.c, contiene la creación y manipulación de las estructuras de datos W_{RTT}, W_{median} y W_m, que son requeridas por SIC.

Se definen cuatro funciones:

- `fixed_window_init`: Recibe el tamaño para la estructura, crea y reserva memoria. Retorna un puntero a la estructura creada.
- `fixed_window_insert`: Recibe el puntero a la estructura a modificar, junto con el dato a insertar.
- `fixed_window_Hread`: Recibe tres parámetros, primero un puntero a la estructura a consultar, luego un entero identificador que permite calcular en base a su valor la primera o la segunda mitad de los datos de la estructura y, por último, el tercer parámetro es el puntero que contiene los datos devueltos en la consulta previa.
- `fixed_window_read`: En base al índice recibido, devuelve el valor correspondiente de la estructura recibida mediante un puntero.

El archivo aux.h simplemente contiene las variables globales y las declaraciones de las funciones requeridas por SIC.

El archivo principal, mediante el cual SIC puede cumplir su objetivo, se llama sic_client.c. Existen varias funciones contenidas dentro de este archivo, siendo las principales las que se describen a continuación:

- `start_values`: Sirve para inicializar las estructuras de datos requeridas, también lee el archivo de configuración config.conf, en el cual se encuentran establecidos todos los valores para las constantes que serán utilizadas a lo largo de la ejecución del programa. Además, establece el primer estado con el cual inicia el algoritmo, cuyo valor corresponde a $NOSYNC$.
- `get_timestamp`: Proporciona el tiempo del sistema en formato *epoch time*, con una resolución de microsegundos.
- `create_socket`: Función encargada de la comunicación entre el cliente y el servidor. Dicha comunicación se lleva a cabo por medio de *sockets* pertenecientes al dominio de comunicación de tiempo AF_INET, mediante el uso del protocolo UDP.

- `send_sic_packet`: Esta función se encarga del envío y recepción de los paquetes hacia y desde el servidor. Durante el envío de cada paquete, un temporizador es activado de tal forma que la comunicación espere dicho tiempo para recibir la respuesta; caso contrario, corta la comunicación y espera el siguiente envío.

- `validate`: Permite controlar que los datos recibidos sean secuenciales, ya que al utilizar el protocolo UDP, el cual no es orientado a conexión, los paquetes pueden llegar fuera de secuencia, pueden no llegar o contener errores. Por lo tanto, el proceso de validación permite comprobar la secuencia, eliminar duplicados y asegurar la integridad.

- `process`: De todas las funciones mencionadas anteriormente, ésta es la que realiza el procesamiento principal de SIC. Integrando al resto de funciones, lleva a cabo el proceso de sincronización, por lo cual se considera la más importante. A continuación se muestra la implementación completa de esa función:

```
1 void process(sic_data sic_receive, fixed_window *W_m, fixed_window *
    W_median, fixed_window *W_RTT, fixed_window *W_epoch)
2 {
3   double aux;
4   double aux2;
5   double a[P];
6   double RTT_l;
7   double RTT_f;
8   double m_RTT;
9   double cov00, cov01, cov11, sumasq,m,c;
10  char caux[20];
11  char maux[20];
12
13  if(sic_receive.to==false)
14  {
15    // Estimate phi //
16    aux=sic_receive.t1 - sic_receive.t2 +(sic_receive.t2 - sic_receive.t1 +
       sic_receive.t4 - sic_receive.t3)/2;
17    fixed_window_insert(W_m,aux); // Add fi into W_m window //
18    fixed_window_insert(W_median,get_median(W_m->w,MEDIAN_MAX_SIZE)); //
     Get median and insert into W_median //
19    fixed_window_insert(W_epoch,(double) sic_receive.epoch);     // Add
     epoch into W_epoch //
20    fixed_window_insert(W_RTT,sic_receive.t4 - sic_receive.t1);    // Add
     rtt into W_RTT //
21    fixed_window_Hread(W_RTT,0,a);
22    RTT_f=gsl_stats_min(a, 1, P);
23    fixed_window_Hread(W_RTT,1,a);
24    RTT_l=gsl_stats_min(a, 1, P);
25    m_RTT=gsl_stats_min(W_RTT->w,1,2*P);
26
27    if(fabs(RTT_l-RTT_f)<=err_RTT*m_RTT)
28    {
29      if(sic_receive.epoch >= (pre_sync + P))
30      {
```

```
31          printf("Sync = true\n");
32          synck=true;
33        }
34      }
35      else
36      {
37        save_syn_values("0","0", "NOSYNC");
38        synck=false;
39        epoch_sync=INT_MAX-P;
40        pre_sync=INT_MAX-P;
41        err_sync = sic_receive.epoch;
42        for(int i=0; i < MEDIAN_MAX_SIZE;i++ )
43          W_m->w[i]=0;
44        for(int i=0; i < P; i++)
45          W_median->w[i]=0;
46      }
47      if((synck==true) && (sic_receive.epoch >= (epoch_sync + P)))
48      {
49        gsl_fit_linear(W_epoch->w, 1, W_median->w, 1, P, &c, &m, &cov00, &
      cov01, &cov11, &sumasq);
50        actual_c=c;
51        actual_m=(1-alpha)*m+(actual_m*alpha);
52        epoch_sync = sic_receive.epoch;
53        sprintf(maux,"%f",actual_m);
54        sprintf(caux,"%f",actual_c);
55        save_syn_values(maux,caux, "SYNC");
56      }
57      else
58      {
59        if(sic_receive.epoch >= (err_sync + MEDIAN_MAX_SIZE))
60        {
61          pre_sync = sic_receive.epoch;
62          err_sync = INT_MAX-MEDIAN_MAX_SIZE;
63        }
64        if(sic_receive.epoch >= pre_sync + P)
65        {
66          gsl_fit_linear(W_epoch->w, 1, W_median->w, 1, P, &c, &m,&cov00, &
      cov01, &cov11, &sumasq);
67          synck = true;
68          epoch_sync = sic_receive.epoch;
69          actual_m = m;
70          actual_c = c;
71          sprintf(maux,"%f",actual_m);
72          sprintf(caux,"%f",actual_c);
73          save_syn_values(maux,caux, "PRESYNC");
74        }
75      }
76    }
77    else
78    {
79      printf("Timeout\n");
80      to=false;
81    }
82  }
```

LISTING 5.1: Función principal requerida por SIC para realizar el proceso de sincronización

Como se explicó en el capítulo 4, SIC hace uso de estimaciones de medianas y la recta por mínimos cuadrados. En la función 5.1, líneas 18, 49 y 66, se puede observar el uso

de las estimaciones en mención, siendo importante remarcar que se utilizaron funciones optimizadas, las mismas que fueron provistas por la biblioteca numérica creada para el lenguaje C *The GNU Scientific Library (GSL)* [22].

- `save_syn_values`: Sirve para almacenar los estados del algoritmo y los parámetros de sincronización. Dichos valores son escritos en el fichero *sync_ values.dat* y actualizados durante la ejecución del algortimo. Dichos valores deben estar disponibles para cuando se requiera conocer la hora local corregida.

Finalmente, el archivo de configuración config.conf contiene los parámetros principales para las constantes requeridas por SIC; este archivo es leído al iniciar el programa, con lo cual su lectura es realizada sólo una vez.

Variables utilizadas y valores recomendados. A continuación daremos un detalle de dichas constantes, además de las variables del programa, junto con los valores recomendados.

a) Constantes

1) `RUNNING_TIME`: corresponde al intervalo de envío de cada paquete (1 segundo).
2) `MEDIAN_MAX_SIZE`: se refiere al tamaño de la ventana utilizada para calcular la mediana de la medición (600 segundos).
3) `P`: es el período con el cual se realiza la estimación de la recta por mínimos cuadrados, para obtener los parámetros de sincronización (60 segundos).
4) alpha: es el coeficiente de autorregresión de la pendiente de $\Phi(t)$ (0.05).
5) `TIMEOUT`: corresponde al temporizador máximo de espera de respuesta del servidor (0.8 segundos).
6) `SERVER_IP`: es la dirección IP correspondiente al servidor.
7) `SERVER_PORT`: se refiere al número de puerto UDP, con el cual el servidor recibe las comunicaciones para los paquetes SIC (4444).
8) `err_RTT`: es el valor considerado para conocer la variación de RTT y poder detectar los cambios de ruta (0.2).

b) Estados

1) `NO_SYNC`: estado con el cual el algoritmo inicia; también retoma este estado cuando un cambio en la ruta de los paquetes es detectada.

2) `PRESYNC`: estado previo a la sincronización; permite hacer la primera estimación de la recta, la misma que será utilizada al entrar en sincronismo.

3) `SYNC`: indica cuando se la llegado a un estado de sincronismo con el servidor.

c) Variables

1) `err_sync`: variable de tipo entero con el *timestamp* en segundos del período inicial (NO_SYNC); se utiliza para conocer cuándo la ventana de mediciones W_m está completa y así estimar la mediana.

2) `pre_sync`: variable de tipo entero con el *timestamp* en segundos del período inicial de pre-sincronismo; sirve para realizar la primera estimación de la recta.

3) `synck`: variable de tipo booleana que permite validar si el algoritmo se encuentra sincronizado.

4) `epoch_sync`: variable de tipo entero con el *timestamp* en segundos de cada período de sincronismo (cada P segundos), siendo la pauta que determina cada estimación y actualización de los parámetros de sincronización.

5) `epoch`: variable de tipo entera con el *timestamp* en segundos de cada RUNNING_TIME.

6) `t1, t2, t3, t4`: variables de tipo *long long integer*, que contienen los *timestamps* tanto del cliente como del servidor en microsegundos.

7) `actual_m`: variable de tipo doble que almacena el valor de la pendiente de la recta.

8) `actual_c`: variable de tipo doble que almacena el valor del intercepto de la recta.

9) `Wm`: estructura de datos con un arreglo de tipo doble y largo MEDIAN_MAX_SIZE; sirve para almacenar los valores de Φ instantáneo.

10) `Wmedian`: estructura de datos con un arreglo de tipo doble y largo P; sirve para almacenar los valores de cada mediana estimada.

11) `WRTT`: estructura de datos con un arreglo de tipo doble y largo $2P$; se utiliza para almacenar los valores de cada RTT.

12) `RTTl`: variable de tipo doble, que contiene el valor mínimo de RTT sobre la última ventana P; se usa para determinar si ocurrió un cambio de ruta.

13) `RTTf`: variable de tipo doble, que contiene el valor mínimo de RTT sobre la ventana P previa; sirve para determinar si ocurrió un cambio de ruta.

Servidor. A diferencia del cliente, para el lado servidor fue necesario crear un código más simple, dado que el único rol que desempeña el servidor es recibir la consulta del cliente y devolver los valores de tiempo al recibir y devolver el paquete. El fichero sic_server.c utiliza algunas de las funciones utilizadas por el cliente, las mismas que se listan a continuación:.

- `start_values`: Sirve para iniciar las estructuras de datos requeridas, también lee el archivo de configuración config.conf en el cual se encuentran establecidos concretamente los valores correspondientes a la dirección IP local y el puerto mediante el cual el servidor permitirá recibir conexiones.

- `get_timestamp`: Proporciona el tiempo del sistema en formato *epoch time*, con una resolución de microsegundos.

- `create_socket`: Función encargada de la comunicación entre el servidor y el cliente, mediante el uso del protocolo UDP.

- `wait_connections`: Esta función es la principal en el lado del servidor, la misma que se muestra en el código 5.2. Como se observa, arranca y entra en un bucle infinito (línea 15), esperando recibir paquetes del cliente. Cuando este evento sucede, un paquete de 67 caracteres es recibido (línea 16), en ese instante el servidor consulta su tiempo mediante la función get_timestamp, dicho tiempo corresponde a $t2S$ (línea 17). El paquete recibido es procesado (lineas 19, 20, 21 y 22). Luego, previo a devolver el paquete, un nuevo tiempo es obtenido $t3S$ (línea 23), agregado al paquete (línea 26) y devuelto al cliente (línea 30). Este procedimiento es repetitivo mientras el cliente envíe paquetes.

```
void wait_connections()
{
  int nBytes, i;
  struct sockaddr_storage serverStorage;
  socklen_t addr_size, client_addr_size;
  long long int t2;
  long long int t3;
  char t2_str[17];
      char t2_pps_str[17];
  char t3_str[17];
  char chain_tx[67];
  char chain_rx[67];

  addr_size = sizeof serverStorage;
  while(1){
    nBytes = recvfrom(udpSocket,chain_rx,sizeof(chain_rx),0,(struct
    sockaddr *)&serverStorage, &addr_size);
    t2=get_timestamp();
    snprintf(t2_str, sizeof(t2_str), "%ld", t2);
    strncpy(chain_tx, strtok(chain_rx,"|"),sizeof(chain_tx));
    strncat(chain_tx,"|",sizeof(chain_tx));
    strncat(chain_tx,t2_str,sizeof(chain_tx));
    strncat(chain_tx,"|",sizeof(chain_tx));
    t3=get_timestamp();
    snprintf(t3_str, sizeof(t3_str),"%ld", t3);
    strncat(chain_tx,t3_str,sizeof(chain_tx));
    strncat(chain_tx,"|",sizeof(chain_tx));
    strncat(chain_tx,"0000000000000000",sizeof(chain_tx));
    memset(chain_rx,'\0',strlen(chain_rx));
    printf("%s\n",chain_tx);
    sendto(udpSocket,chain_tx,sizeof(chain_tx),0,(struct sockaddr *)&
    serverStorage,addr_size);
  }
}
```

LISTING 5.2: Función principal requerida por SIC para realizar el intercambio de paquetes en el lado servidor.

5.2. Resultados obtenidos

Tal como fue explicado en el capítulo 3, son tres los escenarios en los cuales fue desarrollado y probado el algoritmo de sincronización SIC; el primero de ellos, un escenario base, con los dispositivos conectados entre si de manera directa. El segundo considerado como escenario exterior (Buenos Aires, Argentina - Los Ángeles, Estados Unidos), y el tercero, considerado como escenario interior o local (dentro de Buenos Aires, Argentina). Todos los resultados presentados se basan en la métrica MTIE [21].

Estimación de MTIE. Como se explicó en la sección 4.2, el uso de MTIE como métrica para la estimación de los valores de error, nos permitió realizar comparaciones con diferentes ventanas, de acuerdo a la ecuación 4.12. Como metodología aplicada en los escenarios considerados, se optó por utilizar ventanas S igual a 60, 120, 180, 300 y 600 segundos. Observando estos resultados, se pudo comprobar que los valores de error obtenidos van incrementándose a medida que la ventana aumenta de tamaño. Esto se debe a que, los relojes tienden a irse separando con el tiempo, siendo notorio que en grandes ventanas los valores de error son mayores. Una manera de normalizar y hacer una comparativa, en la que todas las ventanas se encuentren en igual condición, se tomó la ventana más pequeña como referencia, S igual a 60 segundos (1 minuto). Entonces, para cada ventana, el valor de MTIE resultante es divido por su correspondiente valor de S, pero convertido en minutos.

Para todos los escenarios en cuestión, se hicieron las comparaciones para dos casos. El primero, realizar estimaciones de error tomando en cuenta una ventana de medianas con 120 mediciones. El segundo, fue que las estimaciones se realizan con una ventana de medianas con 600 mediciones. Es necesario aclarar que el valor de T, para el segundo y tercer escenario corresponde a 604800 segundos (7 días) de medición. Nos valdremos de gráficos de función de distribución acumulada (CDF) para comparar y mostrar los resultados.

Mediciones escenario base. Inicialmente, para conocer el punto de partida con el cual se puede comparar los resultados obtenidos en los escenarios interior y exterior, fue necesario recrear un escenario base. Dicho escenario, consistió en realizar mediciones entre dos dispositivos conectados de manera directa entre si y físicamente en el mismo sitio. Esta prueba nos permitió conocer el error base de SIC. Dicho error está afectado únicamente por las capacidades de *hardware* de los dispositivos Raspberry Pi y el módulo GPS, equipos con los que se realizó y probó el algoritmo. La figura 5.1, muestra el error correspondiente a la ventana de estimación de medianas con 120 mediciones. El error máximo en base al percentil 90 es de 15.62 microsegundos. Para el segundo caso, con ventanas de estimación en ventanas de 600 mediciones, se lo puede observar en la figura 5.2. Considerando el peor caso, con S igual a 60 segundos, el error alcanza los 15.80 microsegundos en el percentil 90.

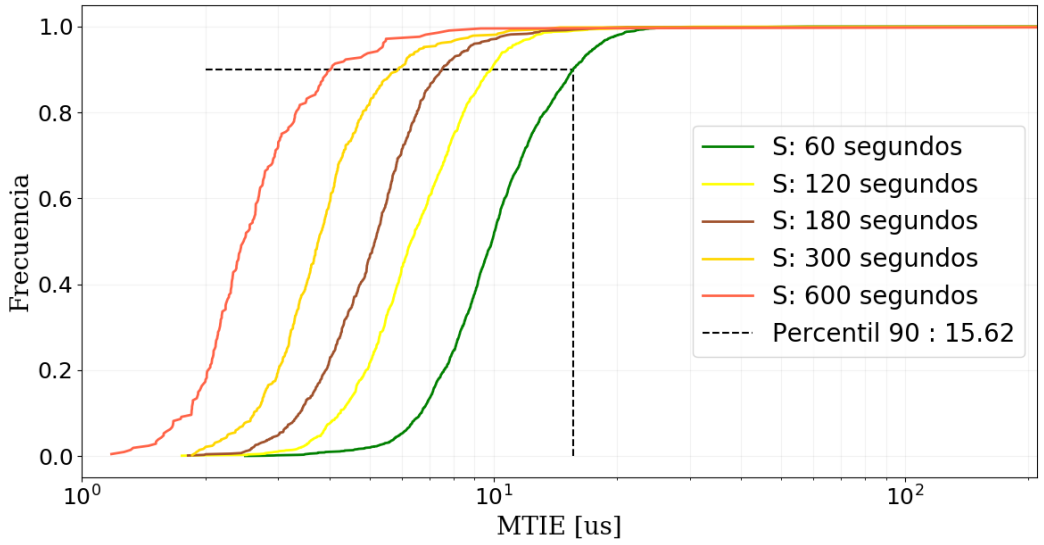

FIGURA 5.1: MTIE en ventanas de medianas de 120 mediciones, escenario base.

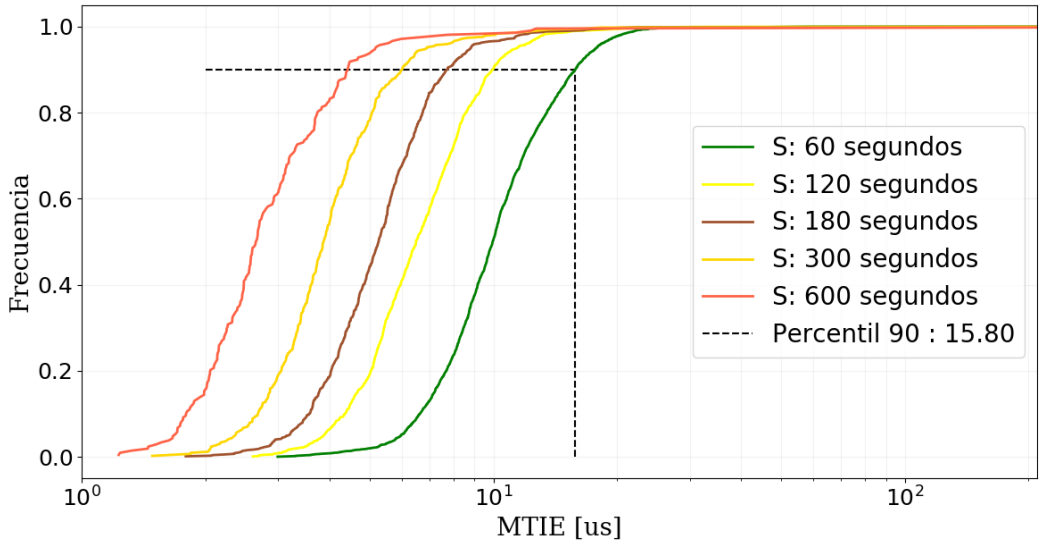

FIGURA 5.2: MTIE en ventanas de medianas de 600 mediciones, escenario base.

Mediciones escenario exterior. Este escenario es el más distante y contiene la mayor cantidad de saltos de red. Según la traza 3.3 se puede observar que el paquete cruza por 20 dispositivos de red para llegar a su destino. El RTT es de 202 milisegundos. Como se observa en la figura 5.3, la primera consideración es con la ventana de estimación de las medianas en un período de 120 mediciones. Se muestra todas las ventanas de MTIE, siendo el peor de los casos S igual 60 segundos. Se presentan todas las ventanas de MTIE, observándose que el peor de los casos corresponde a S igual a 60 segundos. En

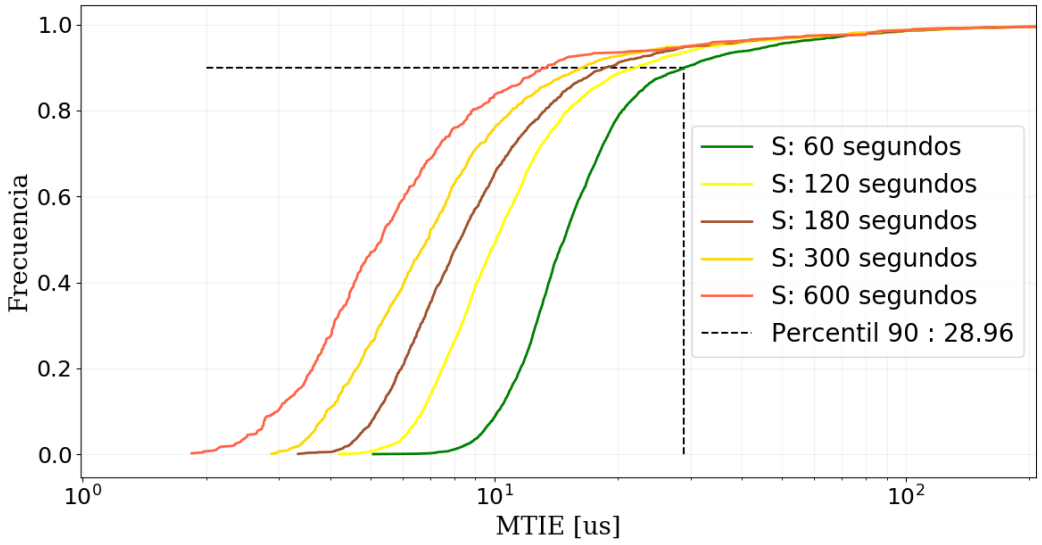

FIGURA 5.3: MTIE en ventanas de medianas de 120 mediciones, escenario exterior.

este caso se aprecia que el error máximo conseguido en base al percentil 90 es de 28.96 microsegundos.

Para el segundo caso, que corresponde a la ventana de estimación de medianas en 600 mediciones, se hizo el mismo análisis. En la figura (5.4), se observa que al ampliar la longitud de ventana de estimación ayuda un poco a mejorar los valores de error.

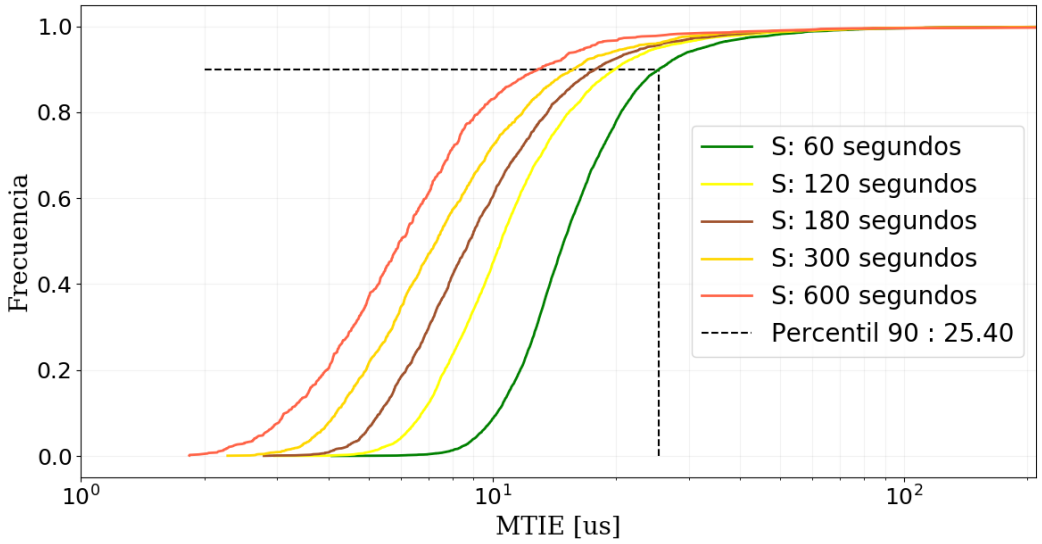

FIGURA 5.4: MTIE en ventanas de medianas de 600 mediciones, escenario exterior.

Si separamos el caso de S igual a 60 segundos (el peor de los casos) y observamos el resultado sobre el percentil 90, es de 25.40 microsegundos, con lo cual tenemos una mejoría respecto a la ventana de estimación de 120 mediciones. Comparando estos dos valores resultantes, se obtiene una diferencia de 3.56 microsegundos, quedando en evidencia que los resultados son mejores al utilizar una ventana de estimación de medianas en períodos de 600 mediciones.

Mediciones escenario Interior. En este escenario, al ser local, la cantidad de saltos de red disminuye a la mitad; según la traza 3.4 se puede observar que el paquete cruza por 10 dispositivos de red para llegar a su destino. Por ende, el RTT en este escenario es menor, llegando a 9 milisegundos. Realizando el mismo análisis en este escenario, como era de esperarse, la disminución del valor de error es notoria. Este comparativo inicia de igual forma con la ventana de estimación de 120 mediciones. En la figura 5.5, es posible apreciar mejoría respecto al escenario exterior. De la misma forma, se muestran

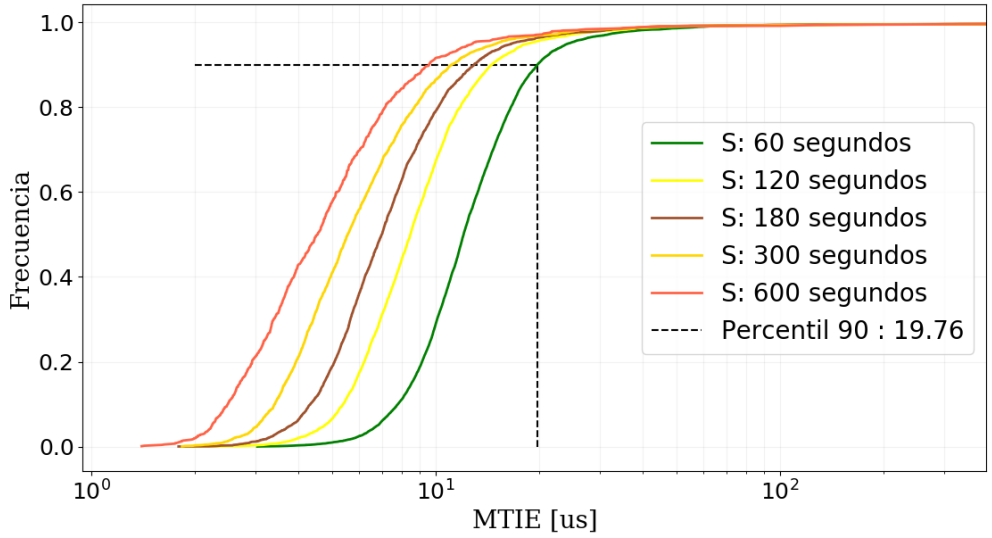

FIGURA 5.5: MTIE en ventanas de medianas de 120 mediciones, escenario interior.

los resultados con diferentes valores de S. Observando el peor caso con S igual a 60 segundos (ver figura 5.5), y bajo el percentil 90, el error en este punto llega a 19.76 microsegundos.

Bajo el segundo caso, ampliando la ventana de sincronización a 600 mediciones se observa un comportamiento muy cercano respecto a la ventana de sincronización de 120 mediciones, tal como lo muestra la figura 5.6.

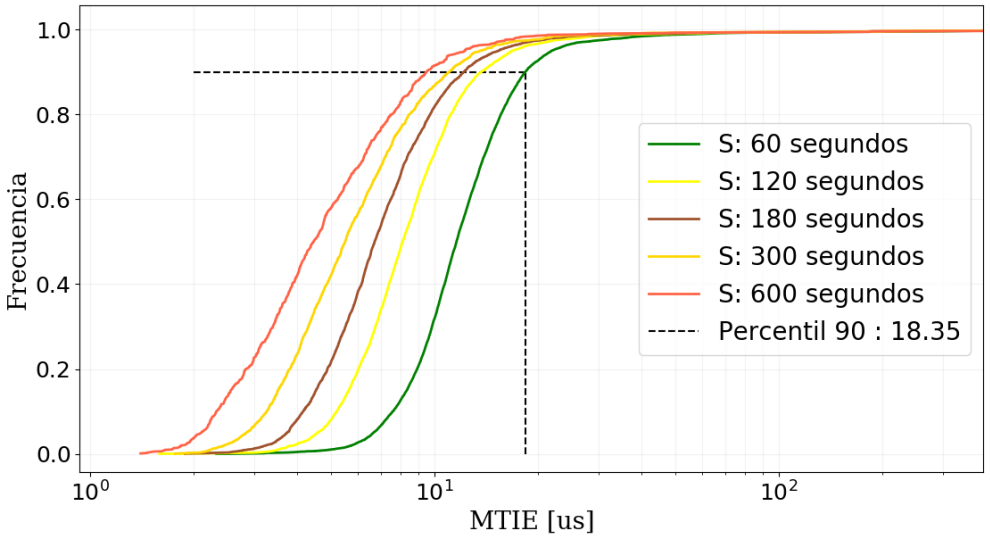

FIGURA 5.6: MTIE en ventanas de medianas de 600 mediciones, escenario interior.

Luego, si separamos el peor caso y observamos los valores en el percentil 90, tenemos como resultado un valor de 18.35 microsegundos. Al realizar una comparación entre los resultados obtenidos entre las dos ventanas de estimación tenemos una diferencia de 1.41 microsegundos. Sin embargo, a pesar que en este escenario la diferencia resultante no es tan notoria como el caso del escenario exterior; sí es importante remarcar que las utilizar una ventana de estimación de medianas en periodos de 600 mediciones, resulta más efectiva para lograr mejores resultados, razón por la cual dicho valor es el que se recomienda utilizar.

Capítulo 6

Conclusiones

Presentamos un nuevo algoritmo de sincronización de relojes, basado en el modelo cliente-servidor y que su funcionamiento sea a través en Internet. Según el análisis realizado sobre algunos de los algoritmos de sincronización utilizados hoy en día, la propuesta del algoritmo SIC (*Synchronizing Internet Clocks*) resulta una alternativa capaz de sincronizar relojes en modo relativo (frecuencia) con mayor precisión que algunos de los actuales basados en *software*. Dicha sincronización resulta útil para realizar mediciones de tráfico o congestión, localización, cripto-monedas, juegos, entre otras. Nuestro algoritmo fue diseñado para que su utilización sea simple, sin que sea necesario instalar ningún equipamiento de *hardware* especial. El desarrollo del protocolo SIC, está basado en el comportamiento del tráfico y su análisis estadístico, permitiendo una robusta sincronización en los difíciles entornos de Internet. En cambio otros protocolos, utilizan el RTT mínimo para sincronizar, el cual está fuertemente afectado por el tráfico en la red. Además, el hecho de tener la capacidad de detectar los cambios de ruta hace que SIC pueda informar en todo momento si está correctamente sincronizado, con lo que se evita proporcionar valores erróneos. Hemos obtenido un MTIE de 25,40 μs por minuto para la experiencia Buenos Aires - Los Ángeles ($RTT \simeq 200\ ms$), y de 18,35 μs para Buenos Aires - Buenos Aires ($RTT \simeq 10\ ms$), para el percentil 90. Con el valor de referencia de error conseguido de 15,80 μs en el escenario base, bajo el mismo tipo de *hardware*, es el valor mínimo del error, lo cual nos permite conocer la precisión real de SIC. Para Buenos Aires - Los Ángeles, la precisión obtenida fue de 9,59 μs, mientras que para Buenos Aires - Buenos Aires de 2,55 μs. Con estos resultados, si dispusiéramos de elementos de *hardware* con mayores prestaciones, los valores mejorarían aún más. Comparando estos

resultados frente a protocolos como PTP, TSC y NTP, SIC posee un mejor desempeño que TSC (mismo en entornos WAN con grandes retardos $RTT \simeq 200ms$), y también que NTP (entre décimas y centésimas de ms). El mejor desempeño de PTP, está basado en la utilización de *hardware* extra en la mayoría de los casos, como también podría ser el caso de la utilización de un GPS en el dispositivo Maestro, el cual se encarga de proveer el tiempo hacia el resto de dispositivos Esclavos. Además, su diseño es para operar principalmente para redes LAN, ambiente que se tiene más control en relación a lo que es Internet. Por último, proveemos una implementación en lenguaje C, con licencia GNU, que se puede descargar gratuitamente en [23]. Además este trabajo se ha enviado a la IETF con el objetivo de producir una RFC, trabajo que se inicia mediante la escritura de una versión *draft* [24].

Trabajos futuros. Al finalizar esta tesis, hay algunos aspectos que no pudieron ser incluidos dentro del alcance de la misma. Por lo que se pueden considerar como trabajos futuros.

Una de las primeras extensiones sobre SIC, es el agregado de cierto nivel de seguridad. Esto con el fin de evitar ataques del tipo MITM (hombre en el medio o *man-in-the-middle* en inglés). Este agregado de seguridad se puede realizar mediante el firmado de los paquetes, que a su vez podría generar imprecisiones en la sincronización por el procesamiento adicional que implica la verificación de firmas.

Otro aspecto importante, está relacionado con la sincronización absoluta. Como se explico durante el planteamiento del modelo que utiliza SIC, lograr este tipo de sincronización está limitado a no poder estimar con precisión la constante k, de la ecuación 4.2. Evaluar la manera de determinar la asimetría de los caminos entre el cliente y el servidor, nos permitiría estimar correctamente este valor, con lo cual se lograría una sincronización absoluta.

En cuanto a los resultados, consideramos que se puede realizar el análisis de los mismos mediante el uso de lo que se conoce como la varianza de *Allan*, con lo cual se podría observar la variación de la frecuencia de los relojes, esto, como una alternativa a la métrica MTIE que fue utilizada en este trabajo.

Bibliografía

[1] NTP Public Services Project. Network time protocol. `http://www.ntp.org/ntpfaq/NTP-s-algo.htm#Q-ACCURATE-CLOCK`. Consultado en 07/05/2018.

[2] Masoume Jabbarifar, Michel Dagenais, and Alireza Shameli-Sendi. Online incremental clock synchronization. *Journal of Network and Systems Management*, 23(4): 1034–1066, 2015.

[3] John Eidson and Kang Lee. Ieee 1588 standard for a precision clock synchronization protocol for networked measurement and control systems. In *Sensors for Industry Conference, 2002. 2nd ISA/IEEE*, pages 98–105. IEEE, 2002.

[4] David Mills, Jim Martin, Jack Burbank, and William Kasch. Network time protocol version 4: Protocol and algorithms specification. Technical report, 2010.

[5] Darryl Veitch, Julien Ridoux, and Satish Babu Korada. Robust synchronization of absolute and difference clocks over networks. *IEEE/ACM Transactions on Networking (TON)*, 17(2):417–430, 2009.

[6] Ki Suh Lee, Han Wang, Vishal Shrivastav, and Hakim Weatherspoon. Globally synchronized time via datacenter networks. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 454–467. ACM, 2016.

[7] Attila Pásztor and Darryl Veitch. Pc based precision timing without gps. In *ACM SIGMETRICS Performance Evaluation Review*, volume 30, pages 1–10. ACM, 2002.

[8] Omer Gurewitz, Israel Cidon, and Moshe Sidi. One-way delay estimation using network-wide measurements. *IEEE/ACM Transactions on Networking (TON)*, 14 (SI):2710–2724, 2006.

[9] Ruxandra Lupas Scheiterer, Chongning Na, Dragan Obradovic, Gunter Steindl, and Franz-Josef Goetz. Synchronization performance of the precision time protocol in

the face of slave clock frequency drift. In *Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on*, pages 554–559. IEEE, 2008.

[10] Protocolo ptp-1588. `https://www.endruntechnologies.com/pdf/PTP-1588.pdf`. Consultado en 08/10/2017.

[11] Jim Martin, Jack Burbank, William Kasch, and Professor David L. Mills. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905, June 2010. URL `https://rfc-editor.org/rfc/rfc5905.txt`.

[12] Guo-Song Tian, Yu-Chu Tian, and Colin Fidge. High-precision relative clock synchronization using time stamp counters. In *Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on*, pages 69–78. IEEE, 2008.

[13] Raspberry pi compute module. `https://www.raspberrypi.org/documentation/hardware/computemodule/RPI-CM-DATASHEET-V1_0.pdf`, 2016. Consultado en 07/06/2016.

[14] Raspbian. `https://www.raspberrypi.org/downloads/raspbian/`. Consultado en 07/06/2016.

[15] Neo-6 u-blox 6 gps modules, 2011. URL `https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_(GPS.G6-HW-09005).pdf`. Consultado en 10/07/2016.

[16] Package: pps-tools (1.0.2-1). `https://packages.debian.org/sid/pps-tools`. Consultado el 11/07/2016.

[17] C Ellingson and Richard Kulpinski. Dissemination of system time. *IEEE Transactions on Communications*, 21(5):605–624, 1973.

[18] Allen B Downey. Using pathchar to estimate internet link characteristics. In *ACM SIGCOMM Computer Communication Review*, volume 29, pages 241–250. ACM, 1999.

[19] J. Alvarez-Hamelin, Esteban Carisimo, and Sebastián Grynberg. Influence of traffic in stochastic behavior of latency. In *7th PhD School on Traffic Monitoring and Analysis (TMA), Irlanda Dublin*, 2017.

[20] Vladimir M Zolotarev. *One-dimensional stable distributions*, volume 65. American Mathematical Soc., 1986. Page 61.

[21] Stefano Bregni. Measurement of maximum time interval error for telecommunications clock stability characterization. *IEEE transactions on instrumentation and measurement*, 45(5):900–906, 1996.

[22] Gsl - gnu scientific library, 2016. URL `https://www.gnu.org/software/gsl/`. Consultado en 10/11/2017.

[23] José Ignacio Alvarez-Hamelin, David Samaniego, and Alfredo A. Ortega. Synchronizing internet clocks. `https://github.com/CoNexDat/SIC`, 2018.

[24] José Ignacio Alvarez-Hamelin, David Samaniego, and Alfredo A. Ortega. Synchronizing Internet Clocks (sic). Internet-Draft draft-alavarez-hamelin-tictoc-sic-00, Internet Engineering Task Force, March 2018. URL `https://datatracker.ietf.org/doc/html/draft-alavarez-hamelin-tictoc-sic-00`. Work in Progress.