elektron ámbar

Original Article

Revista elektron, Vol. 5, No. 2, pp. 112-119 (2021)
https://doi.org/10.37537/rev.elektron.5.2.137.2021

Sincronización de relojes embebidos en Internet (E-SIC) sin hardware dedicado

Embedded Synchronizing Internet Clocks (E-SIC) without dedicated hardware

Javier Alejandro Atadia*[1], Ramiro Alonso*[2], Leonardo Martín Carducci*[3]

* *Universidad de Buenos Aires, Facultad de Ingeniería, Argentina*
[1] jaatadia@fi.uba.ar, [2] ralonso@fi.uba.ar, [3] lcarducci@fi.uba.ar

Recibido: 09/10/21; Aceptado: 08/11/21

***Abstract*—E-SIC is an algorithm designed to synchronize the clocks of embedded controllers through a Local Area Network (LAN) using 802.11 (WI-FI) technology. It's based on the SIC algorithm that synchronizes frequency between the clocks by taking advantage of the symmetry of the networks round trip time. By considering an historic window of measurements and the usage of the mode as an estimation tool, the E-SIC algorithm has achieved absolute clock sync between two devices with a higher accuracy than two milliseconds without the requirement of extra hardware.**

Keywords: clock; synchronization; WI-FI; IoT.

***Resumen*— E-SIC es un algoritmo diseñado para la sincronización de relojes en controladores embebidos a través de una red local (LAN) con tecnología 802.11 (WI-FI). Dicho algoritmo esta basado en el algoritmo SIC que sincroniza frecuencia entre relojes suponiendo la simetría en los caminos de ida y retorno de los mensajes. Al utilizar una ventana de tiempo histórica y la moda como herramienta de estimación, el algoritmo E-SIC logra sincronización entre dispositivos con una exactitud mejor que dos milisegundos sin necesidad de hardware extra.**

Palabras clave: relojes; sincronización; WI-FI; IoT.

I. Introducción

La Facultad de Ingeniería de la Universidad de Buenos Aires, ha comenzado el desarrollo de redes de sensores para la medición de distintas variables físicas (por ejemplo: aceleración triaxial o temperatura) en diversas estructuras civiles. Dichas mediciones presentan un período de muestreo de 2 ms por lo que debemos asegurar una sincronización absoluta con un error menor a 1 ms para poder correlacionar los datos recolectados por los diferentes dispositivos que supervisan una estructura.

Un sistema distribuido de tiempo real consiste en un conjunto de nodos que están interconectados por un red local y se comunican solamente mediante el intercambio de mensajes. Cada nodo es una computadora auto contenida (o microcontrolador) y contiene un reloj de tiempo real de la precisión que puede dar un cristal de cuarzo [1] de uso estándar en dichos microcontroladores. Dependiendo del uso específico que se le de a cada sistema, se le exigirá a los relojes que estén sincronizados con la exactitud requerida en cada caso. La misma puede ser desde el orden de los segundos, hasta los microsegundos.

Presentamos a continuación un análisis de los sistemas de sincronización existentes. Cabe destacar que la bibliografía existente sobre la sincronización de relojes en sistemas distribuidos tanto en redes cableadas como inalámbricas [2] es bastante extensa por lo que solo citaremos la necesaria para comprensión del marco general del tema.

a) Global Navigation Satellite System (GNSS): Los algoritmos de Sincronización por GNSS [3] utilizan la señales de una constelación satélites geosincrónicos para para estimar el tiempo. Cada nodo debe tener el hardware necesario para recibir dichas señales y debe contar con disponibilidad constate de al menos cuatro satélites suficientemente dispersos por el cielo y no debe haber obstrucciones para que la sincronización sea confiable. La visibilidad de los satélites en posiciones alejadas hace casi impracticable este sistema en muchas situaciones. Trabajos previos han logrado sincronizaciones menores a los 3 μs [4], pero debido a la aplicación de nuestros dispositivos en estructuras civiles donde la señal es fácilmente obstruida y principalmente el costo extra en la construcción de los sensores hacen que esta opción no sea viable.

b) Network Time Protocol (NTP): NTP [5] es el algoritmo de facto utilizado en Internet. Nodos clientes se sincronizan contra un nodo servidor enviándole paquetes a través de UDP y esperando su respuesta. NTP realiza un análisis estadístico de los datos recolectados y suele sincronizar en forma absoluta cada un minuto, modificando el RTC (*Real Time Clock*) local, que a su vez se traduce en variaciones del reloj del orden de los cientos de milisegundos. Dichas variaciones impiden la correlación de los datos con una precisión en el orden de 1 ms.

c) Precision Time Protocol (PTP): El algoritmo PTP [6] cuenta con una arquitectura maestro-esclavo donde el proceso de sincronización es comenzado por el nodo maestro. Este algoritmo se usa en redes locales cableadas y con hardware dedicado logra una precisión de microsegundos. La implementación de dicho protocolo en sistemas embebidos e inalámbricos ha sido estudiada por diversos autores y se han logrado resultados aceptables con eficiencia energética [7] [8] [9]. Sin embargo, el *hardware* necesario para que el algoritmo alcance la exactitud requerida aumentan el costo de los nodos, lo que imposibilita su uso en nuestro sistema.

d) Synchronizing Internet Clocks (SIC): El algoritmo SIC [10] trabaja en modo cliente-servidor. El cliente realiza consultas periódicas al servidor y analiza estadísticamente los resultados para estimar parámetros de conversión para el reloj interno, en donde se utiliza una función que provee la hora con la corrección. Este algoritmo no cambia el

reloj interno por lo que se evitan saltos de tiempo cuando se realizan las consultas. Sin embargo, este algoritmo fue concebido para sincronización en frecuencia en Internet, y por lo tanto no es una sincronización absoluta, como es necesario en este caso. Los errores reportados de valores absolutos rondan las décimas de milisegundos, con algunos microsegundos en la sincronización de frecuencia.

El protocolo SIC cuenta con un máquina de estados que puede tomar tres diferentes valores, *NO-SYNC*, *PRE-SYNC* y *SYNC*. SIC empieza en estado *NO-SYNC*, toma una muestra cada un segundo, y a los seiscientos segundos transiciona al estado *PRE-SYNC* debido a que ya puede calcular un primer valor de sincronismo. Finalmente luego de sesenta segundos de encontrarse en el estado *PRE-SYNC* el estado cambia al estado *SYNC* porque tiene mediciones suficientes para calcular con precisión la diferencia de relojes. De esta manera, una vez inicializado el algoritmo éste demora once minutos hasta que comienza a producir aproximaciones. Si en cualquier momento se detecta un cambio sustancial en los tiempos de las muestras recolectadas, o se pierden suficientes muestras, el estado vuelve a *NO-SYNC* (porque estima un cambio de ruta) y comienza nuevamente el proceso.

II. Algoritmo de sincronización E-SIC

El algoritmo E-SIC toma como base el algoritmo SIC [10] y le realiza modificaciones para lograr los requerimientos de sincronización absoluta esperados.

A. Protocolo de intercambio de paquetes

El protocolo de intercambio de paquetes del algoritmo E-SIC es el mismo que el presentado por el algoritmo SIC. El cliente intercambia paquetes sobre UDP (del inglés, *User Datagram Protocol*) como podemos apreciar en la Fig. 1.

El intercambio de paquetes comienza cuando el cliente emite un paquete con el *timestamp* del instante de su envió al servidor (t_1). El servidor recibe el paquete luego de un tiempo desconocido que denominaremos $t(c \rightarrow s)$ y le adjunta su *timestamp* correspondiente a la recepción de dicho paquete (t_2). Una vez procesado el paquete, el servidor lo reenvía al cliente adjuntándole previamente su *timestamp* actual (t_3). Finalmente el cliente recibe el paquete después de un tiempo desconocido que denominaremos $t(s \rightarrow c)$ y le agrega su *timestamp* correspondiente a la recepción (t_4). De esta manera el cliente ahora cuenta con cuatro valores que procesará para poder realizar correcciones al momento de informar el tiempo actual. Nótese que los *timestamps* t_1 y t_4 fueron creados usando el reloj del cliente, mientras que los *timestamps* t_2 y t_3 refieren al reloj del servidor. Si tanto el mensaje o su respuesta se pierden en el camino, esto es detectado y se procede a descartar la presente medición. Si llegasen a ocurrir una determinada cantidad de estos fallos sucesivos el algoritmo se reinicia, comenzando nuevamente el proceso de sincronización.

En la linea 11 del Algoritmo 1 se desencadena el intercambio de paquetes descriptos en pseudocódigo del Algortimo 2, y finalmente el Algoritmo 3 muestra el pseudocodigo del intercambio de paquetes del lado del servidor.

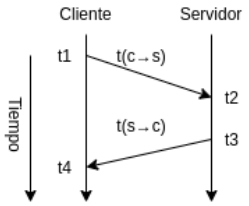

Fig. 1. Intercambio de paquetes del algoritmo SIC.

B. Cambios al algoritmo SIC

En la implementación y definición del algoritmo E-SIC se realizaron los siguientes cambios: eliminación del análisis de cambio de caminos, eliminación de la verificación de firmas de paquetes y del reloj local para el cambio de estados, y modificación del estimador del desfasaje instantáneo. El objetivo es hacer el protocolo más simple para adecuarlo así al uso de microcontroladores y al entrono de un red de área local (del inglés, LAN) que es un entorno controlado. Todas estas modificaciones atacan temas transversales al proceso de la sincronización

1) Análisis de cambio de caminos: El algoritmo SIC cuenta con un mecanismo que detecta el cambio de rutas que recorren los paquetes entre el cliente y el servidor, y en caso de detectar dicho cambio procede a limpiar es estado interno del modelo y recomenzar el proceso de sincronización desde su primer fase. Debido a la aplicación por la cual se desarrolló este trabajo, basta con que nuestro sistema funcione dentro de una red local (LAN), por lo que no habrá cambio de las rutas entre los nodos del sistema. Por este motivo procedemos a remover esta sección del algoritmo.

2) Firma de paquetes y validaciones de identidad: Para asegurar que los paquetes que reciben el servidor y los clientes sean autentificados, la especificación de algoritmo SIC propone un esquema de validación de paquetes y firmas; de esta manera el modelo se encuentra protegido en caso ante ciertos ataques maliciosos. Suponiendo que el tráfico del algoritmo no debería salir de la LAN sobre la que tenemos completo control, estos procesamientos adicionales de seguridad pueden ser obviados sin suponer una vulnerabilidad en el sistema; ya que no se atraviesa ningún enrutador intermediario.

3) Uso del reloj local: El algoritmo SIC supone que se colecta una muestra por segundo y utiliza el transcurso del tiempo para controlar la transición de estados y no la cantidad de muestras recolectadas. Debido a esto, En caso de demoras o fallos en la recolección de las marcas de tiempos (*timestamps*), se produce una transición de estados antes de que este posea todas las muestras que requiere para realizar una buena estimación. Como solución a estos problemas introduciremos una variable que funcionara como contador de los ciclos que han transcurrido, por lo que el uso del reloj local quedara limitado sólo a la recolección de *timestamps*.

4) Cálculo del desfasaje instantáneo: En esta sección introducimos un método de cálculo de desfasaje y su corrección, ver el Algoritmo 1 utilizado. La primera gran diferencia entre el algoritmo SIC y su variante E-SIC es la introducción de una constante para modelar la diferencia

entre los tiempos de ida y de regreso de los paquetes al servidor. El cálculo de la diferencia instantánea de los relojes se basa en la siguiente ecuación:

$$t_c = t_s + \phi \ , \tag{1}$$

donde t_c representa el tiempo según el reloj del cliente, t_s representa el tiempo según el reloj del servidor, y ϕ representa el desfasaje entre ellos. Consideramos que el valor de ϕ varia en el tiempo siguiendo el *Simple Skew Model* [11]:

$$\phi(t) = K + F \cdot t \ , \tag{2}$$

donde K es una constante que representa la diferencia absoluta entre el tiempo de los dos relojes y F que representa la diferencia en frecuencia de los dos relojes. K y F se podrían modelar como variables aleatorias pero eso está fuera del alcance de este trabajo.

Si aplicamos los valores de la interacción entre el cliente y servidor que describimos en la sección anterior a la ecuación 1 obtenemos las siguientes ecuaciones:

$$t_1 = t_2 - t(c \to s) + \phi \tag{3}$$

$$t_4 = t_3 + t(s \to c) + \phi \ , \tag{4}$$

donde:

- t_1: *timestamp* utilizando el reloj del cliente del momento en el que el mensaje de sincronización fue enviado por el cliente.
- t_2: *timestamp* utilizando el reloj del servidor del momento en el que el mensaje de sincronización fue recibido por el servidor.
- t_3: *timestamp* utilizando el reloj del servidor del momento en el que la respuesta del mensaje de sincronización fue enviada por el servidor.
- t_4: *timestamp* utilizando el reloj del cliente del momento en el que la respuesta del mensaje de sincronización fue recibida por el cliente.
- $t(c \to s)$: tiempo que demoró el envío del paquete entre el cliente y el servidor.
- $t(s \to c)$: tiempo que demoró el envío de la respuesta entre el servidor y el cliente.

Suponiendo la constante ρ como la relación entre la demora del envío del paquete sobre la demora de la recepción obtenemos la siguiente ecuación:

$$t(c \to s) = \rho \cdot t(s \to c) \ . \tag{5}$$

Despejando las demoras de envío de paquetes de las ecuaciones 3 y 4 obtenemos las ecuaciones 6 y 7 respectivamente:

$$t(c \to s) = \phi - t_1 + t_2 \tag{6}$$

$$t(s \to c) = -\phi - t_3 + t_4 \ , \tag{7}$$

de esta manera, reemplazando 6 y 7 en 5 obtenemos

$$\phi - t_1 + t_2 = \rho \cdot (-\phi - t_3 + t_4) \ . \tag{8}$$

Despejando ϕ de la ecuación 8 llegamos al resultado:

$$\phi = \frac{t_1 - t_2 - \rho \cdot t_3 + \rho \cdot t_4}{\rho + 1} \ . \tag{9}$$

Dicha ecuación se puede observar en la línea 14 del Algoritmo 1.

El uso de ρ responde a que en los procesos de redes WiFi puede existir asimetría entre el comportamiento del punto de acceso y los clientes, afectando así al ϕ. Si supusiésemos simetría entre los caminos de envío y recepción de los paquetes, es decir $\rho = 1$, llegamos a la ecuación 10 propuesta por el algoritmo SIC. Dicha constante la consideramos una variable conocida del sistema:

$$\phi = (t_1 - t_2 - t_3 + t_4)/2 \ . \tag{10}$$

5) Estimación del desfasaje: Una vez que obtenemos el desfasaje instantáneo, este es guardado en un vector junto con las últimas 599 muestras (línea 15 del Algoritmo 1).

Debido a que el modelado del ruido en la mediciones no es sencillo, ya que depende de muchos factores que podrían tener correlación entre ellos, sólo trabajamos con las mediciones estadísticas de los tiempos. Cada sesenta muestras tomamos la moda de las estimaciones instantáneas, esto nos asegura que estemos tomando los valores más frecuentes de ϕ. La moda la calcularemos utilizado el algoritmo *Half Sample Mode (HSM)* [12], que analiza recursivamente el vector de muestras hasta encontrar la posición de la moda. Una vez encontrada la moda, tomaremos los quince valores ubicados a su alrededor para luego ser guardados junto con las modas que hemos ido extrayendo durante los últimos treinta minutos (lineas 18-23). Finalmente, las modas guardadas son interpoladas linealmente y promediadas con las anteriores interpolaciones (lineas 36-26). Tras lograr la sincronización inicial luego de 12 minutos de su inicialización para el algoritmo SIC y aproximadamente 30 minutos para el algoritmo E-SIC, en el caso que no se hayan perdido mediciones, estos son capaces de estimar el tiempo del servidor usando el reloj interno y la siguiente formula que se desprende de la ecuación 1:

$$t_s = (1 - pendiente) \cdot t_c - ordenada \ , \tag{11}$$

done $pendiente$ y $ordenada$ son la pendiente y ordenada al origen productos de las interpolaciones del algoritmo, relacionando el tiempo en el servidor t_s con el tiempo en el cliente t_c.

El algoritmo E-SIC introduce las siguientes diferencias al calculo de la estimación del valor de ϕ con respecto a su contra parte SIC:

- Guardado del tiempo correspondiente al ϕ instantáneo. Por cada muestra recolectada, SIC extrae la mediana de las estimaciones instantáneas y las guarda para su interpolación junto con el t_1 de la muestra entrante. Sin embargo, ya que ϕ varia en función del tiempo, debido a la diferencia de frecuencias entre los relojes, esto introduce error en el cálculo. E-SIC propone guardar junto con cada estimación instantánea el valor de tiempo al que corresponde dicha estimación, de esta manera eliminamos esa fuente de error.
- Uso de la moda en vez de la mediana. SIC propone el uso de la mediana, ya que al estar diseñado para funcionar sobre Internet, dicha medida proporciona mayor robustez ante las variaciones de demoras causadas por el tráfico de datos (que presenta una estadística con

grandes variaciones). Como E-SIC funciona sobre una única LAN, el problema de las grandes variaciones de tráfico desaparece, entonces la moda y el entorno de muestras alrededor de ella representan mejor el estado actual del desfasaje. La diferencia entre las dos medidas pueden ser observadas en la Fig. 2, donde vemos que la moda se acerca más al pico de ϕ instantáneo que la mediana, situación distinta a la que se observa usando SIC en Internet. Dicha figura fue realizada deteniendo el algoritmo tras el agregado de una nueva iteración del algoritmo, recolectando los valores de las últimas 600 estimaciones instantáneas de ϕ y realizando un histograma con ellas, donde marcamos la mediana y la moda de las mencionadas muestras.

- Cambio de la frecuencia de extracción del vector de muestras. El algoritmo SIC, plantea que por cada muestra insertada realizamos una extracción de la mediana o la moda y ese valor es apartado junto con los últimos 59 valores para el uso posterior en la interpolación lineal. El algoritmo E-SIC plantea espaciar esta extracción a una vez por minuto y conservar las últimas 30 extracciones para realizar la interpolación.

III. Experimentos

El algoritmo fue implementado en un microcontrolador ESP32 [13] utilizando el lenguaje de programación C, por medio de su compilador GCC. Para mayores detalles sobre la implementación, el código fuente se encuentra disponible en https://github.com/jaatadia/tic-toc-sic.

Para su implementación, el algoritmo fue modularizado de manera tal que la recolección y el procesamiento de los *timestamps* sean independientes. Ya que la interacción con los sistemas externos suele depender de llamadas al sistema, esta decisión nos permite la portabilidad del procesamiento de los *timestamps* a cualquier sistema de manera directa.

A. Banco de pruebas

Las mediciones se realizaron utilizando el banco de pruebas de la Fig. 3. En esta podemos ver:

1. ESP32 funcionando en modo servidor.
2. Arduino funcionando como generador de una onda cuadrada con un período de diez segundos.

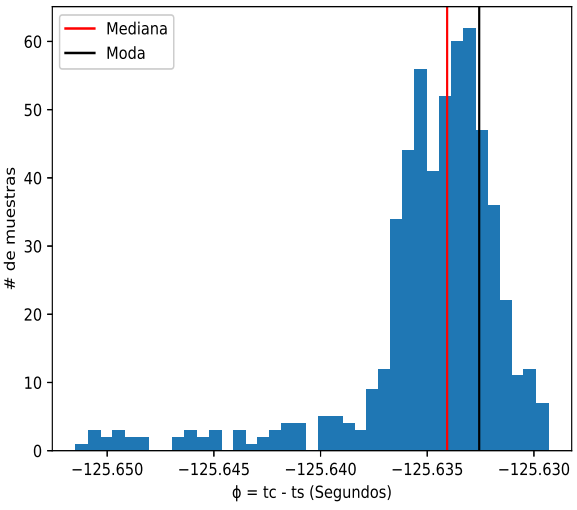

Fig. 2. Histograma de ϕ recolectados con distinción de la moda y la mediana.

Algoritmo 1 E-SIC Cliente.

$ALPHA \leftarrow 0.95$ // Coeficiente de ponderación de las interpolaciones.
$NRO\_MODAS \leftarrow 15$ // Cantidad de muestras al rededor de la moda a tener en cuenta.
$PERIODO \leftarrow 60$ // Cantidad de ciclos entre interpolaciones.
$MAX\_MUESTRAS \leftarrow 600$ // Tamaño del vector que contienen las muestras.
$MAX\_MODAS \leftarrow 30 * NRO\_MODAS$ // Tamaño del vector que contienen las modas.
$estado \leftarrow NO\_SYNC$
$contador \leftarrow 0$
$muestras \leftarrow 0$
$modas \leftarrow 0$
while *true* **do**
  $(t_1, t_2, t_3, t_4) \leftarrow obtenerTimestamps()$
  $contador \leftarrow contador + 1$

  $\phi \leftarrow (t_1 - t_2 - \rho \cdot t_3 + \rho \cdot t_4)/(\rho + 1)$
  $vectorDeMuestras[muestras] \leftarrow (t_1, \phi)$
  $muestras \leftarrow (muestras + 1)\%MAX\_MUESTRAS$

  if $(contador\%PERIODO == 0)$ **then**
    $modasAUtilizar \leftarrow hsm(ordenar(vectorDeMuestras), NRO\_MODAS)$
    while $(t_i, \phi_i) \leftarrow modasAUtilizar$ **do**
      $vectorDeModas[modas] \leftarrow (t_i, \phi_i)$
      $modas \leftarrow (modas + 1)\%MAX\_MODAS$
    end while
  end if

  if $(estado == NO\_SYNC\ y\ contador == MAX\_MUESTRAS)$ **then**
    $contador \leftarrow 0$
    $(pendiente, ordenada) \leftarrow interpolacion\_lineal(vectorModas)$
    $estado \leftarrow PRE\_SYNC$
  else if $((estado == PRE\_SYNC\ o\ estado == SYNC)\ y\ contador == PERIODO)$ **then**
    $contador \leftarrow 0$
    $(aux\_pendiente, aux\_ordenada) \leftarrow interpolacion\_lineal(vectorMedianas)$
    $pendiente \leftarrow ALPHA \cdot aux\_pendiente + (1 - ALPHA) \cdot pendiente$
    $ordenada \leftarrow ALPHA \cdot aux\_ordenada + (1 - ALPHA) \cdot ordenada$
    $estado \leftarrow SYNC$
  end if
  $dormirHastaElProximoSegundo()$
end while

Algoritmo 2 E-SIC Cliente: obtenerTimestamps().

$t_1 \leftarrow epoch()$ // tiempo del reloj del cliente
$enviarServirdor(t_1)$
$(t_1, t_2, t_3) \leftarrow recivirDelServidor()$
$t_4 \leftarrow epoch()$ // tiempo del reloj del cliente
return (t_1, t_2, t_3, t_4)

Algoritmo 3 E-SIC Servidor

while *true* **do**
  $(cliente, t_1) \leftarrow recibir()$
  $t_2 \leftarrow epoch()$ // tiempo del reloj del servidor
  ... // operaciones entre la recepción y el envío propias de la implementación
  $t_3 \leftarrow epoch()$ // tiempo del reloj del servidor
  $enviar(cliente, (t_1, t_2, t_3))$
end while

3. LED que se ilumina cuando la onda cuadrada generada por el Arduino se encuentra en su valor alto.
4. Divisor de resistivo para adaptar la tensión de la onda cuadrada a la tensión de entrada de los ESP32.
5. ESP32 funcionando en modo cliente.

Los ESP32 se alimentan a través de un cable USB

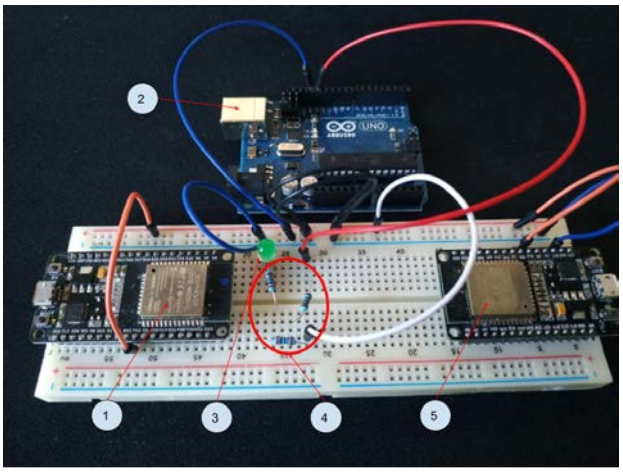

Fig. 3. Banco de pruebas.

que también funciona como conexión a una computadora para poder recolectar la información que éstos generan. El Arduino genera una onda cuadrada que en su flanco ascendente ocasiona una interrupción externa en los ESP32. Los microcontroladores al recibir dicha interrupción imprimen su reloj interno y para el caso de los clientes también imprimen su estimación de la hora del servidor. El cliente y el servidor se comunican a través de la red WiFi para su sincronización. Este intercambio de mensajes se transforman en los t_1 t_2 t_3 y t_4 enunciados en la sección anterior. En el nodo cliente, estos valores son utilizados por el algoritmo, además de ser impresos en la salida serial del programa. Adicionalmente en el servidor se imprime el tiempo t_1 de cada mensaje recibido, ya que esta salida nos permite relacionar las interrupciones externas en el cliente y el servidor. Una vez recolectadas las salidas del cliente y el servidor, tomamos las interrupciones y procedemos a relacionarlas, calculando así el desfasaje real entre ellas y el error. Debido a que t_1 t_2 t_3 y t_4 son parte de la salida del cliente, podemos realizar modificaciones al algoritmo que no influyan en el intercambio de información con el servidor y observar cómo dichos cambios afectan la sincronización.

B. Mediciones y resultados

Suponiendo condiciones de simetría de los caminos de envío y recepción de los mensajes, es decir fijando con valor constante $\rho = 1$, procedimos a recolectar t_1 t_2 t_3 y t_4. Una vez recolectados podemos realizar comparaciones directas entre las diferentes implementaciones SIC y E-SIC con el mismo conjunto de datos. Para realizar las mediciones presentadas en esta sección, los dispositivos ESP32 fueron configurados para que sus placas de red no utilicen el modo bajo consumo, explicaremos brevemente por que en la próxima sección.

Para medir el error de los algoritmos, comparamos el tiempo en el reloj servidor contra el tiempo corregido por el algoritmo. Dicho error se lo conoce como TE o Error de tiempo (del inglés, *Time Error*), ver [14]:

$$TE(t) = |t_s(t) - t_c(t)| \ , \tag{12}$$

donde $t_s(t)$ y $t_c(t)$ representan la hora provista por los relojes de servidor y cliente respectivamente en un instante t. Sin embargo, tanto el algoritmo SIC como el E-SIC, no modifican el valor del reloj local del cliente, sino que cuando se desea conocer la hora se calcula el valor de ϕ y se resta al valor de la hora local. Debido a esto el error se calcula como:

$$TE(t) = |t_s(t) - (t_c(t) - \phi(t_c(t)))| \ . \tag{13}$$

Como podemos ver en la Fig. 4, tanto el algoritmo SIC como el algoritmo E-SIC, se acercan a los valores reales del desfasaje y se mantienen a una distancia casi constante de los valores. Sin embargo el algoritmo E-SIC presenta un error mucho menor que el error del algoritmo SIC. Los valores del error de la Ecuación 13 se pueden observar en la Fig. 5 y en la Tabla I.

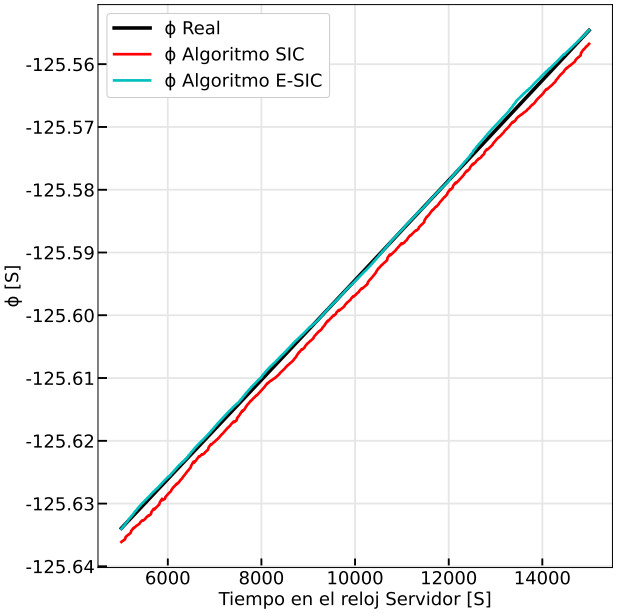

Fig. 4. Aproximaciones de ϕ utilizando los algoritmos SIC y E-SIC, y su comparación con el valor real.

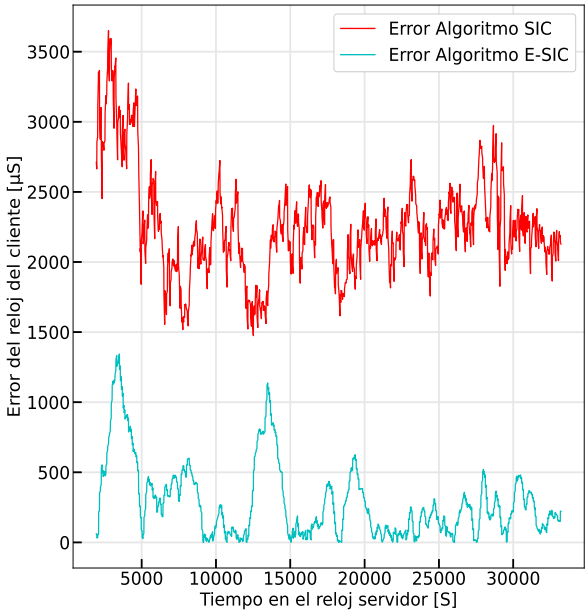

Fig. 5. Error en las estimaciones de los algoritmos SIC y E-SIC.

Como vemos en las figuras previamente mencionadas, el algoritmo E-SIC presenta errores mayores en su comienzo que disminuyen gradualmente hasta un mínimo luego de treinta minutos del inicio. Estos errores son atribuidos a que el algoritmo no cuenta con la cantidad de valores necesarios para la correcta sincronización. Sin embargo, si considerásemos que el algoritmo se encuentra listo para realizar aproximaciones recién al cabo de los treinta minutos de su inicialización, que es cuando sus estructuras internas se terminan de completar, veremos que el pico máximo del error pasa de los 2342 μs a los 1344μs como podemos ver en la Tabla II. Por último, utilizaremos el MTIE (del ingles, *Maximum Time Interval Error*) o Error Máximo del intervalo de tiempo, para caracterizar el sistema de sincronización. Dicha medida es la que propone el estándar de la ITU G.6260 [14], basado en el trabado [15] y es la medida utilizada para comparar distintos métodos de sincronización. Esta medida representa la desviación máxima entre los picos del error de los relojes. La Fig. 7 presenta el MTIE en función del tamaño de la ventana de análisis S, medida en segundos. Notar que si bien E-SIC necesita una ventana mayor para lograr el mismo error que SIC en valores bajos de S, su mayor valor está prácticamente acotado en 1300 ms, mientras que SIC continúa creciendo.

Tabla I
ERROR SEGÚN LAS ESTIMACIONES DEL ALGORITMO SIC Y E-SIC (ECUACIÓN 13).

	SIC	E-SIC
Error Mínimo	1475 μs	< 1 μs
Error Máximo	3650 μs	2342 μs
Error Medio	2272.40 μs	330.22 μs
Desviación Estándar	363.72 μs	354.29 μs

Tabla II
ERROR GENERADO SEGÚN LAS ESTIMACIONES DEL ALGORITMO SIC Y E-SIC TRANSCURRIDOS LOS 30 MINUTOS (ECUACIÓN 13).

	SIC	E-SIC
Error Mínimo	1475 μs	< 1 μs
Error Máximo	3650 μs	1344μs
Error Medio	2256.04 μs	290.38 μs
Desviación Estándar	359.28 μs	272.27 μs

Nuestros análisis contienen parte de esos treinta minutos iniciales debido a que el algoritmo SIC original comienza a proveer valores de sincronización a partir del minuto once. Verificamos de esta manera que las aproximaciones provistas por el algoritmo E-SIC son más exactas que el algoritmo SIC original. Sin embargo, como el error en dicho estado transitorio inicial no cumple con el requerimiento de nuestro sistema y no tenemos limitaciones de tiempo de inicialización el resto de los análisis no consideraran dicho estado. Si analizamos una sección de tiempo más reducida podemos ver en detalle el comportamiento de los dos algoritmos y cómo afectan los cálculos que éstos realizan al cálculo del desfasaje. La Fig. 6 es el tramo correspondiente a las muestras tomadas entre los 5000 segundos a 6200 segundos del servidor de la Fig. 4. En el gráfico, para estos veinte minutos podemos ver dos cosas:

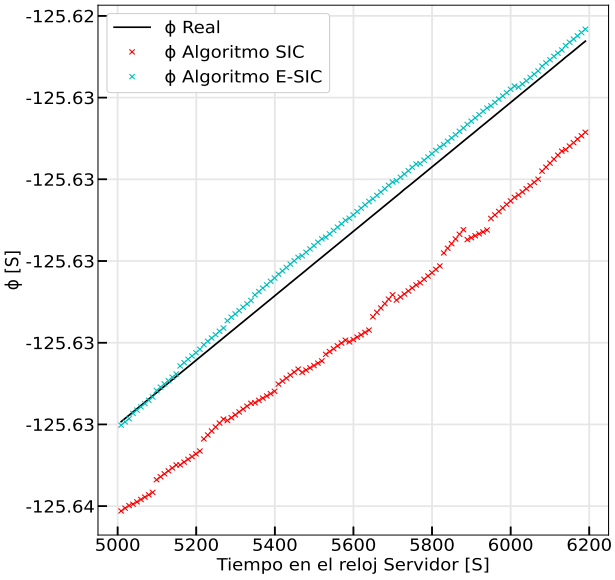

Fig. 6. Sección de las aproximaciones utilizando el algoritmo SIC y E-SIC.

- Cada seis muestras tomadas vemos un salto. Este salto se debe a los ajustes realizados por los dos algoritmos cada un minuto. Los saltos del algoritmo SIC son mas pronunciados a diferencia de su contra-parte que presenta saltos casi imperceptibles. Esto se debe a que el resultado de la interpolación lineal le otorga mayor significancia a la ordenada al origen mas que a la pendiente en el algoritmo SIC, ya que solo tienen en cuenta las medianas de los últimos sesenta segundos. Las modificaciones para trabajar con un intervalo de mediciones centrado en la moda y la utilización de un mayor rango de tiempo, permite relacionar mejor las pendientes de la relación entre los relojes y que no sean tratadas como constantes.
- La diferencia entre las aproximaciones y la recta real permanece casi constante, sin embargo, a diferencia de SIC, el algoritmo E-SIC se aproxima más a la recta real y en instantes hasta se superpone entregando estimaciones con errores menores al microsegundo.

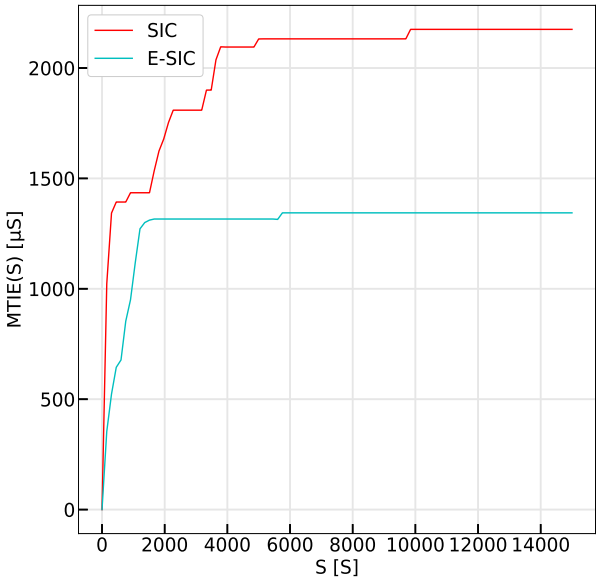

Fig. 7. MTIE(S) para SIC y E-SIC.

C. Corrección del error debido a configuración del ESP32

Los dispositivos ESP32, en su configuración por defecto, pueden llegar a producir latencias indeseadas que debieron ser identificadas y corregidas. Para comprobar esto se generó una señal de disparo para producir interrupciones simultáneas cada 10 s, en los puertos de entrada de propósito general (GPIO) de ambos módulos. Se desarrolló el firmware de prueba de cada dispositivo para registrar el error del tiempo.

Durante las primeras pruebas, una vez transcurridos los treinta minutos para alcanzar el estado de sincronización completa con E-SIC, se registró en forma sistemática un error considerablemente alto (alrededor de 500 ms) como se observa en la Fig. 8. Se observó que este inconveniente se mantenía de manera consistente pese a utilizar dispositivos del mismo fabricante y modelo, descartando además que se tratara de problemas del algoritmo. Naturalmente, esa diferencia resulta inaceptable, ya que uno de los requerimientos más importantes es garantizar un alineamiento en forma absoluta de los períodos de muestreo entre todos los dispositivos manteniendo un error inferior a los 2 ms.

Se logró determinar que para los módulos ESP32 existe una configuración predeterminada que habilita un modo de bajo consumo para el transceiver de WiFi, el cual es administrado por capas de bajo nivel del sistema operativo del dispositivo. Esta configuración produce que los sucesivos encendidos y apagados del transceiver WiFi estén sincronizados con las señales de *beacon* del access point. Este efecto elimina la aleatoriedad en la medición de los tiempos, haciéndolos *beacon* dependientes. Como resultado, se generan latencias que impactan de manera diferente entre los distintos dispositivos produciendo diferencias de tiempos apreciables que se mantienen desde el inicio del sincronismo. Dado que el consumo de energía de los nodos no representa un problema en este caso, ya que para la aplicación final los dispositivos WiFi tendrán garantizado el suministro de energía desde la red eléctrica, se procedió a deshabilitar el ahorro de energía de WiFi de los ESP32, manteniendo la comunicación encendida en forma permanente. Con esta modificación, según se puede apreciar en la Fig. 9 el error se reduce a menos de 2 ms, manteniéndose así luego del transitorio inicial, Fig. 10.

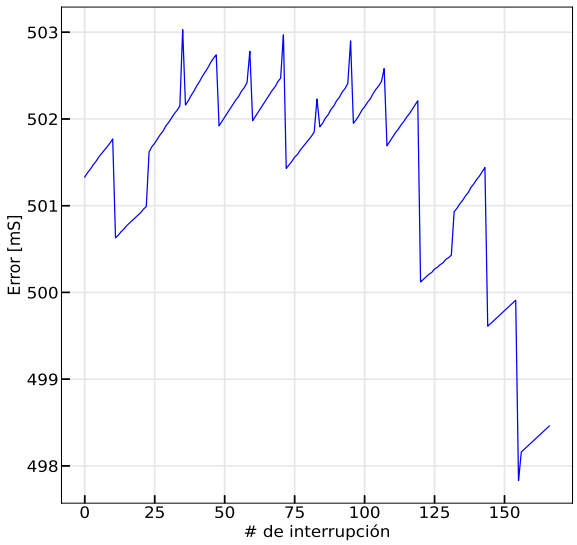

Fig. 8. Error con el modo de bajo consumo de WiFi encendido.

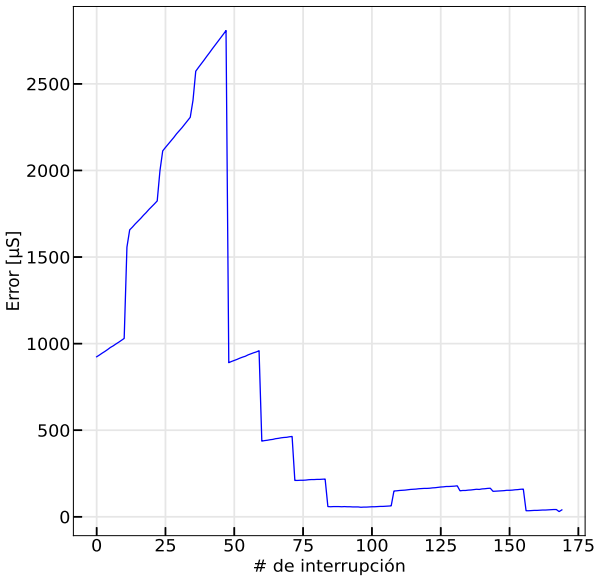

Fig. 9. Error con el modo de bajo consumo de WiFi apagado.

IV. Conclusión

Tras analizar diferentes algoritmos de sincronización y considerar las restricciones de costos, decidimos utilizar el algoritmo SIC como base de nuestra sincronización de relojes. Debido a que dicho algoritmo no contaba con el grado de exactitud requerido procedimos a realizarle modificaciones para adaptarlo a nuestros requerimientos. Nuestras modificaciones en el cálculo del ϕ instantáneo proveen un mecanismo explícito para el caso donde la simetría de los caminos no puedan ser garantizada pero exista una relación constante entre ellos, modelado por el parámetro ρ. La determinación dinámica de dicha constante y su adaptación para sistemas donde la asimetría no sea constante puede ser parte de los trabajos futuros.

Por otro lado, las modificaciones al algoritmo de estimación del desfasaje, al cambiar el uso de la mediana por la moda y al utilizar muestras de una ventana de tiempo más grande, permitieron mejorar la exactitud de la sincronización, reducir su error y suavizar la curva de las estimaciones evitando así micro saltos en las consultas de tiempo. Concluimos de esta manera que el algoritmo E-SIC alcanzó los objetivos planteados y demuestra la posibilidad de la sincronización de los dispositivos con un error menor a un milisegundo en una red inalámbrica, sin utilizar hardware adicional y manteniendo una baja complejidad de cálculo, compatible con un microcontrolador que debe realizar otras tareas.

Referencias

[1] H. Kopetz and W. Ochsenreiter, “Clock Synchronization in Distributed Real-Time Systems,” *IEEE Transactions on Computers*, vol. C-36, no. 8, pp. 933–940, 1987.

[2] M. Mock, R. Frings, E. Nett, and S. Trikaliotis, “Continuous clock synchronization in wireless real-time applications,” in *Proceedings 19th IEEE Symposium on Reliable Distributed Systems SRDS-2000*, 2000, pp. 125–132.

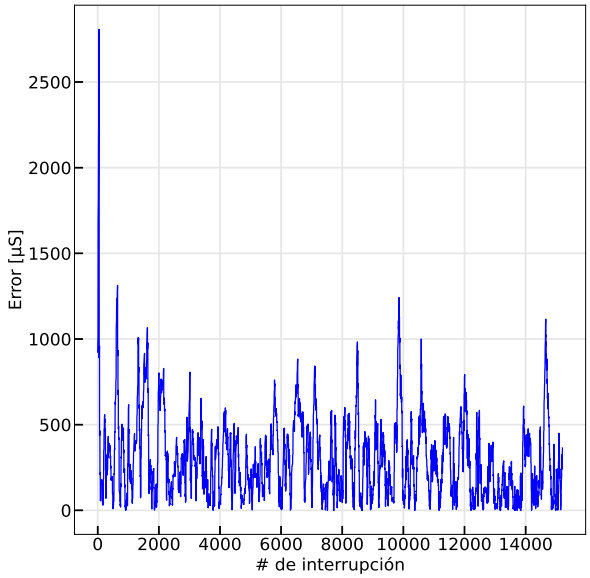

Fig. 10. Error con el modo de bajo consumo de WiFi apagado. Se observa que la reducción del error se mantiene más allá del transitorio.

[3] Masterclock, "GPS vs. GNSS: Understanding PNT Satellite Systems," Jan 2019. [Online]. Available: https://www.masterclock.com/company/masterclock-inc-blog/gps-vs-gnss

[4] T. Yokoyama, A. Matsubara, and M. Yoo, "A real-time operating system with gnss-based tick synchronization," in *2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications*, 2015, pp. 19–24.

[5] J. Martin, J. Burbank, W. Kasch, and P. D. L. Mills, "Network Time Protocol Version 4: Protocol and Algorithms Specification," RFC 5905, Jun. 2010. [Online]. Available: https://rfc-editor.org/rfc/rfc5905.txt

[6] "Ieee standard for a precision clock synchronization protocol for networked measurement and control systems," *IEEE Std 1588-2019 (Revision ofIEEE Std 1588-2008)*, pp. 1–499, 2020.

[7] M. Bansal and A. Gupta, "Out-degree based clock synchronization in wireless networks using precision time protocol," in *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*. IEEE, 2018, pp. 1–6.

[8] R. Holler, T. Sauter, and N. Kero, "Embedded synutc and ieee 1588 clock synchronization for industrial ethernet," in *EFTA 2003. 2003 IEEE Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No.03TH8696)*, vol. 1, 2003, pp. 422–426 vol.1.

[9] L. Li, B. Li, and H. Wang, "Clock synchronization of wireless distributed system based on ieee 1588," in *2010 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, 2010, pp. 205–209.

[10] J. I. Alvarez-Hamelin, D. Samaniego, A. A. Ortega, and R. Geib, "Synchronizing Internet Clock frequency protocol (sic)," Internet Engineering Task Force, Internet-Draft draft-alavarez-hamelin-tictoc-sic-06, Oct. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-alavarez-hamelin-tictoc-sic-06

[11] D. Veitch, J. Ridoux, and S. B. Korada, "Robust synchronization of absolute and difference clocks over networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 2, pp. 417–430, 2009.

[12] D. R. Bickel and R. Frühwirth, "On a fast, robust estimator of the mode: Comparisons to other robust estimators with applications," *Computational Statistics & Data Analysis*, vol. 50, no. 12, pp. 3500–3530, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167947305001581

[13] *ESP32 Series Datasheet*, Espressif Systems, 2021, ver. 3.6.

[14] T. S. S. O. ITU, "Definitions and terminology for synchronization in packet networks (Recommendation ITU-T G.8260)," August 2015.

[15] S. Bregni, "Measurement of maximum time interval error for telecommunications clock stability characterization," *IEEE transactions on instrumentation and measurement*, vol. 45, no. 5, pp. 900–906, 1996.