INMERSIÓN A LOS ALGORITMOS DE SINCRONIZACIÓN DE RELOJES EN SISTEMAS DISTRIBUIDOS

AN IMMERSION INTO CLOCK SYNCHRONIZATION ALGORITHMS IN DISTRIBUTED SYSTEMS

Gabriel Gerónimo-Castillo [1*]

[1] Universidad Tecnológica de la Mixteca, Instituto de Computación, Huajuapan de León, México. ORCID: https://orcid.org/0009-0004-1619-5575. Correo: gcgero@mixteco.utm.mx

Everth Haydeé Rocha-Trejo [2]

[2] Universidad Tecnológica de la Mixteca, Instituto de Computación, Huajuapan de León, México. ORCID: https://orcid.org/0000-0002-8474-0094. Correo: everth@mixteco.utm.mx

Carlos Vázquez-Cid de León[3]

[3] Universidad Tecnológica de la Mixteca, Instituto de Ingeniería Industrial, Huajuapan de León, México. ORCID: https://orcid.org/0000-0003-2067-0565. Correo: carlosvazquezc@mixteco.utm.mx

*** Autor para correspondencia: gcgero@mixteco.utm.mx**

Resumen

El objetivo de la investigación es proporcionar una comprensión profunda de los conceptos y algoritmos aplicados para la sincronización de relojes entre los nodos de un sistema distribuido (SD). En el presente artículo se describen los principales algoritmos relacionados con la solución al problema de la discrepancia de tiempo en los relojes de un sistema distribuido débilmente acoplado. Se parte de la sincronización lógica propuesta por Lamport, que establece una base teórica para la coordinación temporal en sistemas distribuidos, hasta las aportaciones de Mills, implementadas en las versiones del Protocolo de Tiempo de Red (NTP) utilizado actualmente en Internet y el Protocolo de Tiempo de Precisión (PTP) estandarizado por el IEEE, que permite una mayor precisión en la sincronización. La metodología utilizada incluye una revisión exhaustiva de la literatura, analizando tanto algoritmos centralizados como descentralizados. Se abordan las propuestas de Gusella y Zatti, que introducen enfoques innovadores para la sincronización de relojes, así como el algoritmo de Cristian Flaviu, conocido por su simplicidad y efectividad. Los resultados obtenidos demuestran que cada algoritmo tiene ventajas específicas dependiendo del escenario y los requisitos del sistema distribuido en cuestión.

Palabras clave: sistemas distribuidos; sincronización de relojes; algoritmos; protocolo de tiempo de red; protocolo de tiempo de precisión

Abstract

The objective of the research is to provide a deep understanding of the concepts and algorithms applied for clock synchronization among the nodes of a distributed system (DS). This article describes the main algorithms related to solving the problem of time discrepancy in the clocks of a loosely coupled distributed system. It starts from the logical synchronization proposed by Lamport, which establishes a theoretical basis for temporal coordination in distributed systems, to Mills' contributions implemented in the versions of the Network Time Protocol (NTP) currently used on the Internet and the Precision Time Protocol (PTP) standardized by the IEEE, which allows for greater synchronization precision. The methodology used includes a thorough review of the literature, analyzing both centralized and decentralized algorithms. The proposals by Gusella and Zatti, which introduce innovative approaches to clock synchronization, as well as the algorithm by Cristian Flaviu, known for its simplicity and effectiveness, are addressed. The results obtained show that each algorithm has specific advantages depending on the scenario and requirements of the distributed system in question.

Keywords: *distributed systems; clock synchronization; algorithms, network time protocol; precision time protocol*

Fecha de recibido: 26/07/2024
Fecha de aceptado: 20/09/2024
Fecha de publicado: 28/09/2024

Introducción

En un Sistema Distribuido (SD), la sincronización es más compleja que en un sistema centralizado donde existe una memoria compartida y sólo un reloj global que marca el tiempo de cada acción, además es un gran desafío para la coordinación de sus procesos debido a su naturaleza descentralizada y potencialmente heterogénea de los componentes del sistema. En el SD son *n* relojes locales que deben adoptar una estrategia para su sincronización. Los problemas se presentan cuando se envían mensajes para tomar una decisión sobre las tareas a realizar y no se sabe quién los envío primero.

Existen dos planteamientos para la sincronización, el primero es por medio de un reloj lógico y el segundo por medio de un reloj físico. El primer planteamiento para la solución del problema de sincronización en la comunicación entre eventos de máquinas diferentes fue propuesto por Lamport (1978), Fidge (1988) y Mattern (1988) le sumaron un mecanismo de etiquetado de tiempo para seguir la pista del orden causal y, Singhal y Kshemkalyani (1992) se encargaron de eficientar este mecanismo; en el segundo planteamiento destacan las aportaciones de Gusella y Zatti (1987), Cristian Flaviu (1989), Marzullo y Owicki (1983), Mills (1985a, 1985b, 1985c, 1988, 1989a, 1989b, 1991, 1992, 1996, 2010a, 2010b) y el estándar IEEE 1588 (2002, 2008, 2019, 2021).

La presente aportación pretende proporcionar al lector relevantes conceptos y algoritmos aplicados para la sincronización de relojes entre los nodos de un SD, una breve vista de los procesos involucrados en la arquitectura del modelo NTP versión 4 y las características del estándar IEEE 1588 conocido como PTP.

Materiales y métodos

La metodología de esta investigación se desarrolló en 5 fases, siguiendo un enfoque sistemático y exhaustivo para el análisis de los algoritmos de sincronización de relojes en sistemas distribuidos. A continuación se describen las fases involucradas:

1 Revisión de Literatura:
Objetivo: Identificar y analizar los principales algoritmos de sincronización de relojes reportados en la literatura académica.
Actividades:

- Búsqueda de artículos y documentos relevantes en bases de datos académicas.
- Selección de estudios que aborden tanto algoritmos centralizados como descentralizados.
- Análisis crítico de las metodologías y resultados presentados en estos estudios.

2 Clasificación de Algoritmos:
Objetivo: Categorizar los algoritmos identificados en la revisión de literatura.
Actividades:

- Agrupación de algoritmos en centralizados y descentralizados.
- Identificación de características clave y criterios de desempeño para cada categoría.

3 Implementación y Simulación:
Objetivo: Evaluar el desempeño de los algoritmos en un entorno controlado.
Actividades:

- Implementación de los algoritmos seleccionados en un entorno de simulación.
- Configuración de escenarios de prueba que emulen diferentes condiciones de red y carga.
- Ejecución de simulaciones para medir precisión, latencia y robustez de los algoritmos.

4 Análisis de resultados:
Objetivo: Comparar el desempeño de los algoritmos bajo distintas condiciones.
Actividades:

- Recopilación de datos de las simulaciones.
- Análisis estadístico de los resultados para identificar tendencias y patrones.
- Comparación de los resultados obtenidos con los reportados en la literatura.

5 Discusión y conclusiones:
Objetivo: Interpretar los resultados en el contexto de la literatura existente y formular conclusiones.
Actividades:

- Confrontación de los resultados del estudio con los hallazgos de otros investigadores.
- Identificación de las fortalezas y debilidades de los algoritmos analizados.
- Formulación de conclusiones y recomendaciones para futuras investigaciones.

Cada una de estas fases fue diseñada para proporcionar una visión completa y detallada de los algoritmos de sincronización de relojes, desde su revisión teórica hasta su evaluación práctica mediante simulaciones.

Resultados y discusión

Orden de Eventos

Cuando no se tiene o no se confía en un reloj global, se debe saber o indicar cuál de dos eventos ocurrió primero debido a que si no se procesan en orden pueden provocar un mal comportamiento del sistema. Los eventos a contemplar en un SD son los eventos internos del nodo, el evento de envío de un mensaje y el evento de recepción del mensaje. Lo que comúnmente se dice para ordenar los eventos está basado en la relación "paso antes", simbolizando como→, en donde se tienen dos categorías, el orden parcial y el orden total de eventos (Lamport, 1978). Lo que se indica es que, en el sistema se tiene un conjunto de eventos E y una relación ≤ entre los elementos del conjunto, se dice que la pareja (E, ≤) es un conjunto *parcialmente ordenado* o *poset* si cumplen las siguientes propiedades:

1) La relación ≤ es reflexiva, es decir: $\forall e_i, e_j \in E \;\; e_i \rightarrow e_j$.
2) La relación ≤ es antisimétrica, es decir: $\forall e_i, e_j \in E \;\; (e_i \rightarrow e_j) \wedge (e_j \rightarrow e_i) \Rightarrow e_i = e_j$.
3) La relación ≤ es transitiva, es decir: $\forall e_i, e_j, e_k \in E \;\; (e_i \rightarrow e_j) \wedge (e_j \rightarrow e_k) \Rightarrow e_i \rightarrow e_k$.

Además, para que exista un *orden total* entre los eventos se deben cumplir las propiedades anteriores más la siguiente propiedad: $\forall e_i, e_j \in E \;\; (e_i \rightarrow e_j) \vee (e_j \rightarrow e_i)$.

La teoría de orden proporciona una base formal para describir las afirmaciones de que evento sucedió antes que otro. En un orden causal (Birman & Joseph, 1987) los eventos se ven en el orden causa-efecto tal como ocurrieron, y en el orden Δ-causal (Baldoni et al., 1996) se ven en el orden causa-efecto sólo si la causa ha sido vista antes de que expire el tiempo de vida de dichos eventos. En el presente documento no se plasmará a fondo está rama algebraica de relación de eventos, sólo se muestra la aportación de Lamport de orden parcial de donde se desprendieron las dos variantes mencionadas anteriormente.

Propuesta de Sincronización de Lamport

El algoritmo de Condición de Reloj propuesto por Lamport (1978) es uno de los enfoques más conocidos para lograr la sincronización lógica de los eventos cuando se carece de un reloj global que se encargue de sincronizar a los relojes de todo el sistema. Lamport mostró que la sincronización de relojes es posible, señala que la sincronización de relojes no tiene que ser absoluta, si dos procesos no interactúan no es necesario que sus relojes estén sincronizados puesto que la carencia de sincronización no sería observable y por lo tanto no podría provocar problema, además lo que importa por lo general no es que todos los procesos concuerden de manera exacta en la hora sino que coincidan en el orden en que ocurren los eventos, asume que los eventos de un proceso forman una secuencia, donde el evento ***a*** ocurre antes que el evento ***b*** si ***a*** sucede antes que ***b***. Un sólo proceso se define como un conjunto de eventos con un orden total a priori, asumiendo que enviar o recibir un mensaje es un evento que puede definirse como la relación "*paso antes*" (o relación de causalidad), denotada por el símbolo →, bajo las siguientes condiciones:

- Si ***a*** y ***b*** son eventos en el mismo proceso y ***a*** llega antes que ***b*** entonces $a \rightarrow b$.
- Si ***a*** es el envío de un mensaje de un proceso y ***b*** es la recepción del mismo mensaje por otro proceso entonces $a \rightarrow b$.

- Si $a \rightarrow b$ y $b \rightarrow c$ entonces $a \rightarrow c$.

Otra manera de ver que $a \rightarrow b$ indica que, es posible que el evento ***a*** afecte causalmente al evento ***b***, sólo cuando no se cumplen las condiciones anteriores, se dice que los eventos ***a*** y ***b*** son concurrentes, por lo que, ni causalmente se afectan entre sí.

Al introducir un reloj C_i para cada proceso P_i como una función que asigna un número $C_i(\boldsymbol{a})$ a cualquier evento ***a*** en ese proceso, el sistema de relojes se representa como una función C la cual asigna a cualquier evento ***b*** un número $C(\boldsymbol{b})$ donde $C(\boldsymbol{b})=C_j(\boldsymbol{b})$ si ***b*** es un evento en el proceso P_j. Por lo que $\forall a, b \in E: Sia \rightarrow b\ entonces\ C(a) < C(b)$, a lo anterior se le conoce como **condición de reloj**, y el algoritmo relacionado es el siguiente.

Algoritmo de Condición de Reloj

1. Si **a** y **b** son eventos en el proceso $\boldsymbol{P_i}$, y el evento ***a*** sucede antes que el evento ***b*** entonces el proceso $\boldsymbol{P_i}$ incrementa el valor del reloj $\boldsymbol{C_i}$ para cumplir con $C_i(a) < C_i(b)$.
2. Si ***a*** y ***b*** son eventos en los procesos $\boldsymbol{P_i}$ y $\boldsymbol{P_j}$ respectivamente, donde el evento ***a*** es el que *envía* un mensaje ***m*** y el evento ***b*** es el *receptor* de ese mensaje, entonces
   - 2.1 el evento ***a*** debe colocar en el mensaje una etiqueta de tiempo $\mathbf{Tm} = \boldsymbol{C_i(a)}$, y
   - 2.2 el evento ***b*** al recibir el mensaje ***m*** fija su reloj $\boldsymbol{C_j}$ mayor que o igual a su valor actual pero mayor que **Tm** para cumplir con $C_i(a) < C_j(b)$.

Basado en la teoría anterior, se desprende el uso de un vector de relojes en el SD para controlar la causalidad de los eventos.

Propuesta de Fidge y Mattern sobre Vector de Relojes

En la investigación descrita por Baldoni y Raynal (2002) se pueden encontrar las investigaciones iniciales de la inclusión de un vector de relojes para seguir el orden de los eventos, allí se menciona que las primeras aportaciones parten de Fidge (1988) y Mattern (1988), escritas en forma simultánea, que formalizaron el uso de un sistema de reloj vectorial, el cual es un mecanismo de marcas de tiempo que permite rastrear la causalidad entre los eventos en un SD, de esta forma el algoritmo se puede ver cómo sigue.

Algoritmo de Vector de Relojes

1. El tiempo lógico está definido como un vector T_i de longitud N, donde N es el número de máquinas en el sistema.
2. El vector T_i de enteros se inicializa en 0.
3. El tiempo de un proceso i es $T_i[j]$ y su etiqueta de tiempo se coloca en un mensaje que contiene el vector $VT_i[j]$ con componente j-ésimo.
4. El tiempo de un proceso (sitio o nodo) será:
   - $T_i[i]=T_i[i]+1$ cuando sucede un evento interno o es un evento de envío.
   - Cada mensaje contiene el valor del reloj actual T_i o de su proceso emisor i.
   - Cuando un proceso i recibe un mensaje con el vector, todos los j's verifican:
     - Si j=i entonces $T_i[j]=T_i[j]+1$
       sino $T_i[j]=\max(T_i[j], VT_i[j])$

Como se observa, el componente j-ésimo del vector de tiempo de un proceso refleja el valor más alto del componente j-ésimo de todos los mensajes con marca de tiempo que ha recibido. Teniendo en consideración

que sólo el $T_i[i]$ refleja la marca de tiempo local del proceso i, y $T_i[j]$ refleja que el proceso i conoce sobre la marca de tiempo del proceso j, el vector de tiempo VT_i refleja que los procesos i's conocen el último estado de tiempo local de todos los procesos, y que cada componente del vector de tiempo evoluciona de forma independiente.

Tomando en consideración las aportaciones anteriores, Singhal y Kshemkalyani (1992) implementan una técnica para eficientar el uso de un sistema de vector de relojes indicando que si se cuenta con dos vectores, $LU_i[1..N]$ vector de última actualización, y $LS_i[1..N]$ vector de último envío, se tiene que:

1. $LU_i[j]=T_i[i]$ cuando el proceso i actualizó la entrada $T_i[j]$, considerando que $LU_i[i]=T_i[i]$ en todo proceso.
2. $LS_i[j]=T_i[i]$ cuando el proceso i envía por última vez un mensaje al proceso j.

Observando que, desde la última comunicación del proceso i al proceso j sólo han cambiado aquellos elementos $T_i[k]$ para los cuales $LS_i[j] < LU_i[k]$, por lo cual cuando un proceso i envía un mensaje a j sólo necesita enviar aquellas entradas $T_i[k]$ para las cuales $LS_i[j] < LU_i[k]$. Por lo que, el proceso i sólo necesita enviar al proceso j un conjunto de tuplas $\{(x,T_i[x])$ o $(LS_i[j]<LU_i[x])\}$ en lugar del vector de N entradas, reduciendo así la información de envío.

Las teorías anteriores sobre el uso de relojes lógicos se utilizan para establecer un orden parcial entre los eventos pero no representan el tiempo real, aunque son enfoques útiles para determinar la relación temporal entre ellos. Por otra parte, los relojes físicos se utilizan para representar el tiempo real, y su sincronización es crucial para aplicaciones en SD donde se requieren un alto grado de precisión. A continuación se describen los principales algoritmos para realizar sincronización.

Algoritmos Centralizados para Sincronización

Las siguientes dos propuestas presentan algoritmos asimétricos que utilizan un servidor para sincronizar las máquinas del sistema, en el primero un servidor maestro sincroniza las máquinas esclavas y en el segundo las máquinas esclavas preguntan al servidor maestro el tiempo para sincronizarse.

Propuesta de Sincronización de Gusella y Zatti

Gusella y Zatti (1987) plantearon contar con un servidor de tiempo activo que realice un muestreo periódico del tiempo de las máquinas, y a partir de ello calcular un tiempo promedio de desfazamiento e indicar a todas las máquinas que avancen o retrocedan su reloj para que estén sincronizadas. También, indican que las máquinas participantes deben estar en un límite de exactitud llamado ɤ para poder considerarlas en el cálculo del promedio e indicar con ello como ajustar el reloj. En su propuesta se debe contar con un conjunto de demonios de tiempo que se ejecutan cada uno en cada máquina, basados en una estructura de red maestro-esclavo usando un esquema asimétrico donde el maestro actúa como el coordinador. Para mostrar el funcionamiento del algoritmo, llamado comúnmente Algoritmo de Berkeley, suponga que el servidor maestro tiene en su reloj el tiempo de 12:10:00, las máquinas que forman el sistema tiene los siguientes tiempos, $máquina_1$ 12:05:00, $máquina_2$ 12:12:00, $máquina_3$ 12:00:00, y como límite en la exactitud un ɤ=00:05:00, por lo que a partir de ello, se realizan las siguientes acciones.

Algoritmo de Gusella y Zatti

1. El demonio en la máquina designada como maestro inicia la sincronización en cada periodo T.
2. El demonio en el maestro envía un mensaje con su tiempo actual a las demás máquinas, ellas calculan la diferencia del tiempo de su reloj con respecto al recibido, así como el maestro también mide su diferencia de tiempo:

- R_0=C(maestro)-C(maestro)= 12:10:00-12:10:00 = 00:00:00
- R_1=C(máquina$_1$)-C(maestro)= 12:05:00-12:10:00 = -00:05:00
- R_2=C(máquina$_2$)-C(maestro)= 12:12:00- 12:10:00 = 00:02:00
- R_3=C(máquina$_3$)-C(maestro)= 12:00:00- 12:10:00 = -00:10:00

3. Las máquinas participantes envían al maestro su diferencia, y el demonio maestro selecciona a los participantes para calcular el promedio, lo cual depende del valor de γ.

- el maestro participa: $\gamma \geq |R_0|$ (siempre se cumple)
- la máquina$_1$ y máquina$_2$ participan: $\gamma \geq |R_1|$, $\gamma \geq |R_2|$
- la máquina$_3$ no participa: $\gamma < |R_3|$

4. El demonio maestro calcula el promedio tomando en consideración a los participantes y envía este promedio a cada máquina para que realice su ajuste, así como ella hace su ajuste.
   - $Av = \frac{R_0+R_1+R_2}{3} = -00:01:00$
5. Al recibir el mensaje con el tiempo promedio cada máquina realiza el ajuste de su reloj

- Máquina coordinadora. Como su diferencia fue 00:00:00 entonces

$$C(maestro) = 12:10:00 + \big(00:00:00 + (-00:01:00)\big) = 12:09:00$$

- Máquinas del sistema.
  - máquina$_1$. Como su diferencia fue -00:05:00 (está atrasado debe adelantarse) se toma 00:05:00

$$C(máquina_1) = 12:05:00 + \big(00:05:00 + (-00:01:00)\big) = 12:09:00$$

  - máquina$_2$. Como su diferencia fue 00:02:00 (está adelantada debe retrasarse) se toma -00:02:00

$$C(máquina_2) = 12:12:00 + \big(-00:02:00 + (-00:01:00)\big) = 12:09:00$$

  - máquina$_3$. Como su diferencia fue -00:10:00 (está atrasado debe adelantarse) se toma 00:10:00

$$C(máquina_3) = 12:00:00 + \big(00:10:00 + (-00:01:00)\big) = 12:09:00$$

6. Al final del paso 5 las máquinas están sincronizadas, y el maestro espera un tiempo T para iniciar nuevamente el ajuste de relojes.

Gusella y Zatti plantean que a partir de un tiempo *t*, una función real derivable *C(t)* y el valor absoluto de la tasa de desviación máxima del reloj actual ρ, se tiene que el tiempo del reloj de una máquina podrá estar en el intervalo: $1 - \rho \leq \frac{dC(t)}{dt} \leq 1 + \rho$.

Por lo que, dos relojes se dice que están sincronizados en el tiempo t_0 si sus funciones asociadas tienen el mismo valor, es decir, si $C_A(t_0)=C_B(t_0)$, pero si no lo están se encontrarán en un rango de a lo más 2ρ: $\left|\frac{d\cdot(C_A(t)-C_B(t))}{dt}\right| \leq 2\rho$.

Así se puede decir que, si dos relojes están sincronizados en un tiempo t_0, en cualquier tiempo después t_1, su valor puede diferenciar a lo más por $\pm 2\rho(t_1-t_0)$.

Propuesta de Sincronización de Cristian Flaviu

Cristian (1989) indica que toda máquina en el sistema tiene una tasa máxima de alejamiento de su reloj en ρ con respecto al reloj del hardware H en un intervalo de tiempo $[t', t]$, es decir, $(1-\rho)(t-t') \leq H(t) -$

$H(t') \leq (1+\rho)(t-t')$. Partiendo de lo anterior, propone el siguiente algoritmo determinista para sincronizar cualquier máquina con respecto a la que se elija como la servidora de tiempo. El algoritmo es controlado por la máquina que desea sincronizarse con la máquina que será la referencia de tiempo.

Algoritmo de Sincronización de Cristian

1. La máquina P envía a la máquina Q, elegida como servidora de tiempo, un mensaje de solicitud de tiempo.
2. Cuando Q recibe el mensaje, contesta creando un mensaje donde coloca su tiempo T.

3. Si P no recibe respuesta entonces falló en la lectura del reloj Q y debe intentarlo en un tiempo de $(1\text{-}\rho)^{-1}$, en caso de recibir el mensaje de Q se debe contemplar la media del retardo de ida del mensaje de solicitud y la recepción del mensaje de respuesta, a esto se le llama D. Cuando P recibe la respuesta de Q, el tiempo T se encuentra en el intervalo: $[T + min(1-\rho), T + 2D(1+2\rho) - min(1+\rho)]$. Por lo que P puede estimar el valor del reloj de Q, tomando en consideración que 2D es el retardo de ida y vuelta con un error minimizado de $e = D(1+2\rho) - min(\rho)$, y a partir de ello ajustar su reloj.

Para cerrar los anteriores algoritmos centralizados, se puede decir que el algoritmo propuesto por Guzella y Zatti es más eficiente debido a que contempla factores como la propagación de la señal, el tiempo de procesamiento, descartar relojes defectuosos o poco fiables, en lo que se refiere a el algoritmo de Cristian, este es más simple y adecuado para sistemas donde la variación del retardo de red es baja y se necesita una sincronización básica (Kumar & Manjula, 2013). Así mismo, una ventaja que se observa en ambos algoritmos es el bajo número de intercambio de mensajes, pero también presentan desventajas en términos de escalabilidad, sensibilidad a variaciones y factores dependientes del sistema, además del punto de fallo que podría presentar el servidor maestro (Dalwadi & Padole, 2017), por lo que hay que sumar algoritmos de selección del maestro o la duplicidad del maestro, unas propuestas de mejora para la sincronización se presentan a continuación.

Algoritmos Distribuidos de Sincronización

Un SD está formado por un conjunto de procesadores que se comunican por medio de intercambio de mensajes por lo que el algoritmo encargado de la sincronización de los relojes se debe enfrentar al mal funcionamiento de la red y debe contar con tolerancia a fallos. A continuación se muestran algunas propuestas para poder garantizar la sincronización.

Propuestas de Sincronización de Marzullo y Owicki

Marzullo y Owicki (1983) proponen el Algoritmo de Consistencia y el Algoritmo de Intersección para resolver el problema de sincronización de relojes, parte del supuesto que, los relojes en el SD tienen diferentes precisiones, pero en forma general son estables y se encuentra sin averías, de modo que pueden ser ajustados hacia adelante o hacia atrás, además, suponen que se pueden recolectar datos por medio de mensajes broadcast.

Los autores indican que, un reloj estándar es aquel en el cual su función de tiempo $C(t) = t$, es correcto en un t_0 si $C(t_0) = t_0$, es preciso si su primera derivada es un segundo en cada segundo, es estable si su segunda derivada es cero, así un reloj perfecto es correcto, preciso y estable, lo cual es imposible en los relojes de un SD, lo que sí se puede garantizar por medio de un algoritmo es que los relojes sean consistentes entre sí, por lo que sólo estarán desfasados con un error máximo *E*, encontrado en el intervalo $[C_i(t) \pm E_i]$ y un servidor

de tiempo puede calcular un límite del error máximo con un retraso estimado que se mide a partir del tiempo del envío de una solicitud de tiempo y la recepción de la respuesta, llamado ξ, que es no determinista y está acotado, simbolizando como$\sigma_j$la cantidad de tiempo que transcurre desde que el servidor S_i envía una solicitud de tiempo al servidor S_j y cuando S_j recibe la solicitud, denotando como $\rho_j$el tiempo desde que S_j envía la respuesta a S_i y S_i la recibe. Suponiendo también un retraso mínimo de cero y con un límite superior de inexactitud con una tasa derivada mínima δ_i: $\left|1 - \frac{dC_i(t)}{dt} \leq \delta_i\right|$

Donde $C_i(t)$es continuo en $t_0 \leq t \leq t_0 + \Delta$, es decir $C_i(t_0) + \Delta - \delta_i\Delta \leq C_i(t_0 + \Delta) \leq C_i(t_0) + \Delta + \delta_i\Delta$. Tomando entonces la consideración que dos relojes son consistentes en un tiempo t_0 si: $\left|C_i(t_0) - C_j(t_0)\right| \leq E_i(t_0) + E_j(t_0)$. Teniendo las consideraciones anteriores plantean el siguiente algoritmo.

Algoritmo de Consistencia

El algoritmo supone un grafo donde los nodos son servidores de tiempo y las rutas de comunicación son los arcos, los servidores se sincronizan con su vecino cada cierto periodo de tiempo con un pequeño error máximo bajo los siguientes pasos:

1. Un servidor de tiempo S_i con reloj C_i que recibe una solicitud de tiempo responde con el par $\big(C_i(t), E_i(t)\big)$teniendo en consideración que:
   - Si el servidor S_i fue reiniciado, su reloj r_i y un error inherente ε_i deben contemplarse para formar $E_i(t) = \varepsilon_i + (C_i(t) - r_i)\delta_i$.
     Sino fue reiniciado, el tiempo se encuentra en el intervalo $t_0 \leq t \leq t_0 + \Delta$y $E_i(t_0 + \Delta) = E_i(t_0) + \delta_i\Delta$ debido a que ε_i y r_i son los mismo en t_0 y en $t_0+\Delta$.
2. Cada servidor de tiempo envía una solicitud de tiempo a sus vecinos al menos una vez en cada τ segundos. Donde *t* es el tiempo en que S_i envía una solicitud y ξ^i_j el tiempo medido por C_i desde que S_i envío la solicitud y recibió la respuesta de S_j.
3. Al recibir respuesta el servidor S_i verifica la consistente evaluando según la relación: $E_j + (1 + \delta_i)\xi^i_j \leq E_i$, lo cual es verdadero si en S_i sucede que: $\varepsilon_i \leftarrow E_j + (1 + \delta_i)\xi^i_j$, $C_i \leftarrow C_j \ \wedge \ r_i \leftarrow C_j$ en caso contrario se ignora la respuesta por inconsistencia.
4. El servidor S_i ajustan su reloj. Los servidores estarán sincronizados debido a que los relojes que poseen los errores más pequeños son consistentes, es decir: $\forall S_i, S_j \in S \ \left[\left|C_i(t) - C_j(t)\right| \leq E_i(t) + E_j(t)\right]$.

El error máximo de cualquier reloj en los servidores conectados y que ejecutan el algoritmo anterior es igual al error más pequeño en el sistema más cualquier error acumulado durante o después de su último reinicio. La desventaja que indican los autores es que, no es resistente cuando se presentan relojes que tienen un límite superior no válido en su tasa de derivación, y cuando se detecta la incoherencia no se tiene claro que hará el servidor, lo que se asume es que la probabilidad de que se presente es mínima. Otro problema es cuando se reinician los servidores y encuentran inconsistencia con sus vecinos debido al periodo de verificación de la sincronización. Para solucionar el problema, plantean particionar la red en diferentes grupos de servidores. En el algoritmo se tiene una precisión basada en el reloj más preciso del sistema, pero no se garantiza que cada servidor seleccione el mismo reloj para sincronizarse, por lo que todo se limita a la consistencia del sistema completo. Esto les da la pauta para indicar que el tiempo correcto se encuentra en la intersección de los intervalos de tiempo individuales. Formalmente, la intersección en los intervalos de tiempo de dos servidores de tiempo S_i y S_j se encuentra definido como: $\Big[max\big[C_i - E_i, C_j - E_j\big], min\big[C_i + E_i, C_j + E_j\big]\Big]$.

Además, si un intervalo no es subconjunto de otro, y $C_i - E_i \leq C_j - E_j$entonces se puede decir que $C_i + E_i \leq C_j + E_j$ obteniendo así que: $C_i \leq C_j \wedge |C_i - C_j| \leq |E_i - E_j|$. Teniendo las anteriores consideraciones plantean el siguiente algoritmo.

Algoritmo de Intersección

1. Este paso es igual al primer paso del Algoritmo de Consistencia. Un servidor de tiempo S_i con reloj C_i que recibe una solicitud de tiempo responde con el par $(C_i(t), E_i(t))$considerando que:
   - Si el servidor S_i fue reiniciado, su reloj r_i y un error inherente ε_i deben contemplarse para formar $E_i(t) = \varepsilon_i + (C_i(t) - r_i)\delta_i$.
   - Si no fue reiniciado, el tiempo se encuentra en el intervalo $t_0 \leq t \leq t_0 + \Delta$y $E_i(t_0 + \Delta) = E_i(t_0) + \delta_i \Delta$ debido a que ε_i y r_i son los mismo en t_0 y en $t_0+\Delta$.
2. El servidor S_j transforma cada respuesta $[C_i(t), E_i(t)]$dentro del intervalo $[a, b] = [T_j, L_j]$donde: $T_j \leftarrow C_j - E_j - C_i \wedge L_j \leftarrow C_j + E_j + (1 + \delta_i)\xi_j^i - C_i$.

El intervalo $[a, b]$ es construido como $a \leftarrow maxT_j$y $b \leftarrow minL_j$sobre todas las respuestas. Si $b > a$entonces el servicio de tiempo es consistente y S_i tiene $\varepsilon_i \leftarrow (b - a)/2,\ C_i \leftarrow (a + b)/2 + C_i \wedge r_i \leftarrow (a + b)/2 + C_i$.

3. El servidor S_j ajusta su reloj.

El algoritmo utiliza la información de cuánto se han desviado los servidores en comparación con una posible desviación para con ello construir un intervalo más preciso. Su mayor debilidad es la necesidad de que cada servidor tenga un límite superior correcto en la magnitud de su tasa de desviación, si los servidores no tienen límites válidos el sistema puede volverse inconsistente.

Las propuestas anteriores de sincronización no abordan la arquitectura de los protocolos para la comunicación en la red por lo que las aportaciones de Mills, desde sus primeros reportes técnicos sobre la sincronización en la Internet han servido para tener un sistema de sincronización durante los últimos años, formalmente desde 1985 hasta la fecha.

Propuestas de Sincronización de Mills

En forma general, Mills (1991) indica que el sistema NTP consta de una red de servidores de tiempo primarios y secundarios, clientes y rutas de transmisión, el servidor de tiempo principal se sincroniza directamente con una fuente de referencia principal UTC -Tiempo Universal Coordinado, y el servidor secundario deriva la sincronización, posiblemente a través de otros servidores secundarios mediante las rutas de red. En circunstancias normales, la sincronización del reloj se determina utilizando sólo los servidores y las rutas de transmisión más precisas y confiables, de modo que las rutas de sincronización reales generalmente asumen una configuración jerárquica con las fuentes de referencias primarias en la raíz y servidores de precisión decrecientes hacia las hojas. Siguiendo las convenciones establecidas por la industria telefónica, la referencia de precisión de cada servidor de tiempo se contempla por medio de un número llamado estrato, con el nivel asignado como uno para servidores primarios y cada nivel sucesivo para los servidores secundarios asignado como uno mayor que el nivel anterior hacia las hojas.

Las primeras aportaciones sobre sincronización de relojes, tomando como base cronológica lo descrito en (Mills, 2003), se remontan al RFC 778 (Mills, 1981) donde se describe un servicio de sincronización que

utiliza el Protocolo de Mensajes de Control de Internet, ICMP, tipo 13 código 0 solicitud de tiempo, y tipo 14 código 0 respuesta de tiempo y el Protocolo de Puerta de Enlace-Puerta de Enlace, GGP, para encapsular datagramas del Servicio de Reloj en Internet-*Internet Clock Service*, ICS, que transportan los campos de etiquetas tiempo de origen t_1, tiempo recibido t_2, y tiempo transmitido t_3, por lo que al obtener el tiempo de llegada del mensaje t_4 se calcula el retardo para el host (t_2-t_1), el tiempo de regreso (t_4-t_3), y el tiempo de ida y vuelta (t_2-t_1)+(t_4-t_3), y a partir de ellos se proceda a la sincronización. En el RFC 891 (Mills, 1983) introduce el software Fuzzball para la sincronización en una LAN de a lo más 256 hosts, y para obtener los tiempos de los hosts para la estimación y compensación de los relojes se usan mensajes Hello. En el RFC 958 (Mills, 1985a) se describe la primera especificación formal de NTP para la sincronización de los relojes de clientes y servidores distribuidos sobre Internet. Para el envío de solicitudes y respuestas utiliza los protocolos Time/UDP/IP, ICMP/IP, y con los datos recolectados de los tiempos realizaba la sincronización utilizando el Algoritmo de Subconjunto Mayoría -*Majority Subset* y el Algoritmo de Agrupamiento -C*lustering* que se describen en el RFC 956 (Mills, 1985b). En comparación con el Algoritmo de Intersección propuesto por Marzullo y Owicki (1983), Mills toma un enfoque de estimación estadístico con técnica de máxima verosimilitud bajo un método cualitativo de observación y experimentación, y no teoremas formales como lo realizado por ellos. A continuación se describen los dos algoritmos propuestos en el RFC 956:

- **Algoritmo de Subconjunto Mayoría.** El algoritmo sigue un modelo estocástico que consiste en una o más mediciones de diferencias de tiempo entre varios relojes de la red. Uno de los relojes es el reloj local del observador, que toma las mediciones de los otros relojes para formar un conjunto de estadísticas, medias y varianzas, con las que puede estimar el mejor tiempo para configurar el reloj local. El estimador de máxima verosimilitud es una estadística que maximiza la probabilidad de que un resultado particular de un experimento se deba a un conjunto supuesto sobre las restricciones del experimento. Se forman subconjuntos C(n,k) donde n es el número de relojes y k es el entero más grande de n/2 denotado como mayoría mínima, teniendo entonces que:
  1. Determinar la varianza para cada subconjunto de k muestras de las n observaciones.
  2. Seleccionar el subconjunto con la varianza más pequeña y usar su media como estimador de la media de distribución.
- **Algoritmo de Agrupamiento.** El algoritmo se utiliza como estimador de máxima verosimilitud:
  1. Obtiene un conjunto de muestras de n observaciones {x(1), x(2), … , x(n)}.
  2. Calcula la media de las n observaciones del conjunto de muestras y desecha la muestra x(i) con el valor más alejado de la media, dejando n-1 observaciones en el conjunto.
  3. Repite el paso 2 hasta que sólo quede una observación, la cuál será el valor del estimador de máxima verosimilitud.

En el RFC 1059 (Mills, 1988) se proporcionan los mecanismos para la sincronización y distribución de tiempo en la Internet por medio de una subred distribuida de servidores de tiempo que operan en una configuración maestro-esclavo jerárquica y autoorganizada. Los servidores principales toman el tiempo nacional a través de cable o radio, y lo distribuyen usando algoritmos de enrutamiento locales y demonios de tiempo. Para el diseño de NTP versión 1, se recurrió al protocolo Fuzzball/Hello plasmado en el RFC 891 (Mills, 1983) pero realizando algunos ajustes según los experimentos reportados en el RFC 957 (Mills, 1985c), incluyendo relojes sincronizados por radio en varios sitios, y aportando nuevos algoritmos de filtrado, como el Algoritmo de Filtrado de Reloj que selecciona las mejores muestras de compensación de un reloj, y el Algoritmo de

Selección de Reloj que se usa para seleccionar el mejor reloj entre un conjunto jerárquico de relojes. En esta versión 1 se introduce el formato del encabezado NTP de 48 bytes de longitud que se añade después del encabezado UDP usando el puerto 123, entre los campos que contiene está el campo de *stratum* que se utiliza para indicar el nivel del estrato del reloj, si es 1 es una referencia primaria y mayor que 1 indica una referencia secundaria.

En la versión 2 del NTP plasmado en el RFC 1119 (Mills, 1989a) se indica que las rutas de sincronización son determinadas por un árbol expansión de peso mínimo y la redistribución del tiempo es a través de algoritmos de enrutamiento locales y demonios de tiempo, usando un paquete NTP (por omisión de 48 bytes más 12 bytes opcionales para Autenticación) encapsulado en UDP/IP, adicionalmente se suma un mensaje de control auxiliar NTP que puede ser usado cuando no se cuenta con aplicación de monitoreo de red. Los algoritmos utilizan las mismas bases de los primeros algoritmos de las versiones anteriores, pero con las mejoras aplicadas en los experimentos a gran escala diseñadas para evaluar la disponibilidad, precisión y confiabilidad de la distribución de tiempo reportados en el RFC 1128 (Mills, 1989b).

En la versión 3 del NTP documentado en el RFC 1305 (Mills, 1992) se aplican los siguientes ajustes: refina los modelos de análisis e implementación de aplicaciones para trabajar a mayores velocidades aplicando mejoras en la estabilidad, exactitud y precisión en los tiempos, se proporciona un modelo para establecer límites de error para aumentar el rendimiento. También se incorporan dos nuevas características:

1) El Algoritmo de Combinación, para juntar las compensaciones de pares de servidores de tiempo con la finalidad de mejorar la precisión, para el proceso de perfeccionar la selección del reloj usa dos algoritmos, el A*lgoritmo de Selección*, basado en el Algoritmo de Intersección de Marzullo y Owicki, que construye una lista de tuplas <desplazamiento, retardo, dispersión> candidatas elegibles para convertirse en la fuente de sincronización, calcula un intervalo de confianza para cada uno y elimina las candidatas incorrectas de tiempo y, el A*lgoritmo de Agrupamiento* que ordena la lista de candidatas sobrevivientes en orden de estrato descartando repetidamente los valores atípicos hasta quedar sólo los más exactos, precisos y estables;

2) Algoritmos mejorados de reloj local que permiten que los intervalos de sondeo en todas las rutas de sincronización sean incrementados sustancialmente para reducir la sobrecarga de la red. El tamaño del encabezado principal sigue siendo de 48 bytes para los campos de operaciones pero se le suman dos encabezados de extensión de tamaño variable y finaliza con el Código de Autenticación de Mensaje, opcional, formado por el campo de identificación de la clave de 4 bytes y el campo de resumen de mensaje de 16 bytes, también se renombran el campo Distancia de Sincronización -*Synchronizing Distance* por Retardo a la Raíz -*Root Delay* y el campo Dispersión de Sincronización -*Synchronizing Dispersion* por Dispersión a la Raíz -*Root Dispersion*. Por otra parte, en el RFC 2030 (Mills, 1996) se describe el modelo de protocolo simple NTP versión 4, SNTPv4, compatible con NTP implementado para las pilas de protocolos IPv4, IPv6 y OSI, pero que excluye algoritmos innecesarios para operar en un servidor dedicado que cuente con reloj de radio integrado, así como también se pueda utilizar en servidores NTP independientes integrados con receptores GPS.

En la versión 4 de NTP, documentado en el RFC 5905 (Mills, 2010a), se modifica el encabezado del protocolo para poder usarse con IPv6, mejora algoritmos para tener mayor precisión a decenas de microsegundos, se amplían las marcas de tiempo por usar el tipo de dato doble flotante en las implementaciones, e incluye un esquema de descubrimiento de servidor de manera dinámica, servidor manycast / cliente manycast. El servidor primario está sincronizado con un reloj de referencia UTC, y el cliente se sincroniza con uno o más

servidores ascendentes, como un servidor secundario, que tiene uno o más servidores ascendentes y uno o más servidores o clientes descendentes. Existen tres modos para asociar los paquetes NTP: simétrico activo/pasivo, cliente/servidor, y broadcast servidor/cliente. El objetivo de sus algoritmos es minimizar tanto la diferencia horaria como la diferencia de frecuencia entre el UTC y el reloj del sistema, cuando las diferencias se reducen por debajo de las tolerancias nominales, se dice que el reloj del sistema está sincronizado con el UTC, además, se suma un nuevo esquema de autenticación de servidores a clientes por medio del modelo Autokey con clave pública descrito en el RFC 5906 (Mills, 2010b).

El modelo del sistema NTPv4, figura 1, está formado por los siguientes procesos:

- *Proceso Par.* Es un proceso dedicado en el servidor que se encarga de recibir paquetes de otro servidor o del reloj de referencia, opera sobre una estructura de datos común llamada "asociación". Dentro de este proceso se encuentra el *algoritmo de filtrado de reloj* que se encarga de preparar el flujo de datos en línea para seleccionar las muestras con mayor probabilidad de representar el tiempo preciso. El algoritmo produce las siguientes variables: *desplazamiento (θ), retardo (δ), dispersión (ε), jitter (ψ), filtro del reloj (filter) y filtro de tiempo (t_p).* A medida que todos los procesos pares producen cada muestra $<\theta, \delta, \varepsilon, \psi, t>$ son escaneadas por los *Algoritmos de Mitigación: Algoritmo de Selección, Algoritmo de Agrupamiento, Algoritmo de Combinación y el Algoritmo de Disciplina del Reloj,* para calcular tiempos. Para realizar lo anterior, el proceso ejecuta el *protocolo on-wire*, descrito más adelante.
- *Proceso de Sondeo.* Este proceso dedicado en el servidor se ejecuta a intervalos regulares para construir y enviar paquetes en asociaciones simétricas. Para gestionar el filtro del reloj y el registro de alcance se ejecuta de forma continua tanto si los servidores son accesibles como si no lo son. El proceso invoca a la rutina de sondeo para cada asociación y al proceso de transmisión para enviar el paquete.
- *Proceso del Sistema.* Este proceso incluyen *el Algoritmo de Selección, el Algoritmo de Agrupamiento* y *el Algoritmo de Combinación,* que se encargan de encontrar los servidores y relojes de referencias y a partir de ellos determinar los candidatos más precisos y confiables para sincronizar el reloj del sistema. Dentro de este proceso, figura 2, se ejecutan los siguientes algoritmos y procesos:
  - El A*lgoritmo de Selección* que escanea todas las tuplas $<\theta,\delta,\varepsilon,\psi,t_p>$ de asociación y aplica los principios bizantinos de detección de fallos para descartar los candidatos incorrectos llamados "falsetickers", relojes engañosos o inconsistentes, dejando sólo los candidatos "truechimers", los cuales son los relojes que mantiene una precisión preestablecida, y con ellos formar un *clan mayoría*.
  - El A*lgoritmo de Agrupamiento* que utiliza el clan mayoría, y aplica una serie de rondas para descartar las asociaciones estadísticamente más alejada, valores atípicos, dejando un número mínimo de supervivientes más precisos.
  - El A*lgoritmo de Combinación* que utiliza los sobrevivientes de los "truechimers" para calcular el desplazamiento final del reloj promediando sus estadísticas, y con ello produce las mejores estadísticas finales sobre una base de promedios ponderados, el desplazamiento final se pasa al A*lgoritmo de Disciplina del Reloj* para dirigir el reloj del sistema a la hora correcta, pero si el *Algoritmo de Selección* no produce un clan mayoría o no puede producir al menos sobrevivientes,

CMIN, el proceso del sistema sale sin disciplinar el reloj del sistema, pero si hay éxito el Algoritmo de Agrupamiento selecciona el mejor candidato como el "*par del sistema*" y la tupla $<\theta, \delta, \varepsilon, \psi, t\_p>$ que se utilizará para construir las variables del sistema heredadas para servidores y clientes dependientes, así como también se ponen a disposición de otras aplicaciones que se ejecutan en la misma máquina.

- P*roceso de disciplina del reloj*. El proceso controla el tiempo y la frecuencia del reloj del sistema, representado por un Oscilador de Frecuencia Variable, VFO. Las marcas de tiempo emitidas por el VFO cierran el ciclo de retroalimentación que mantiene la hora del reloj del sistema. Asociado al proceso de disciplina se encuentra el *proceso de ajuste del reloj* que se ejecuta una vez cada segundo para inyectar una compensación de tiempo calculada para mantener una frecuencia constante.

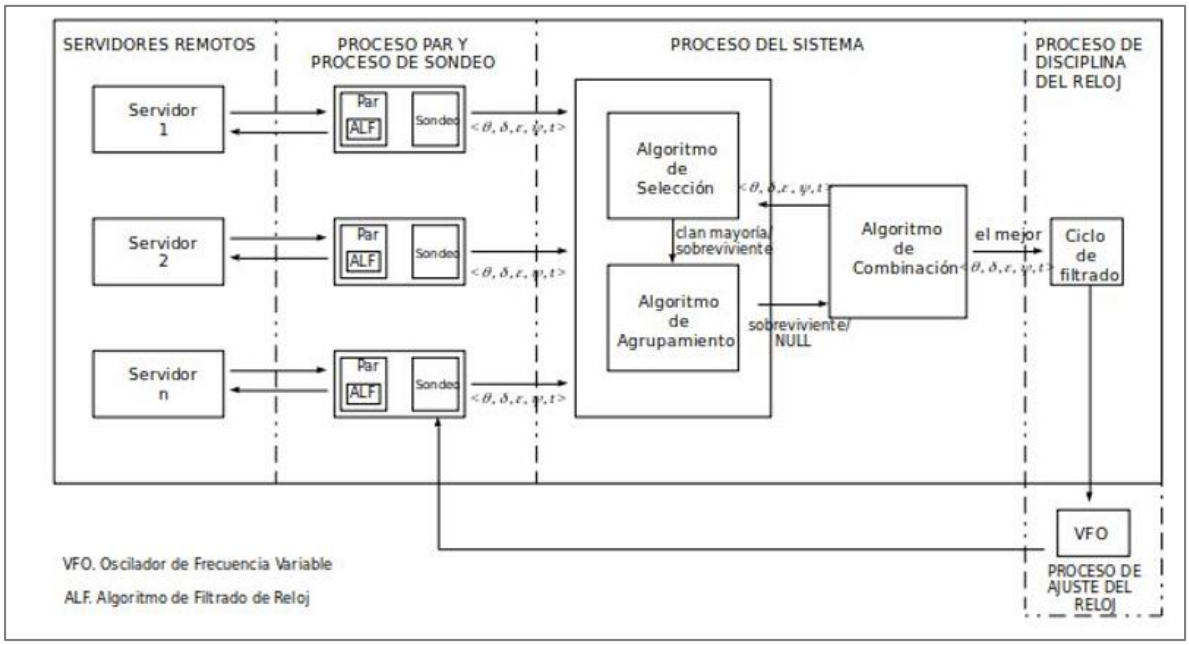

Figura 1. Interacción de los procesos principales del modelo NTPv4.

Nota. Adaptada de *Network Time Protocol Version 4: Protocol and Algorithms Specification, (p. 10), por* Mills, 2010, Internet Engineering Task Force Report RFC-5905. University of Delaware.

Como se mencionó anteriormente los *Algoritmos de Mitigación* se apoyan en el *protocolo on-wire* utilizado en cada nodo, este usa cuatro etiquetas de tiempo $\{t_1, t_2, t_3, t_4\}$ y tres variables de estado {*origen*, *recibido*, *transmitido*} = {*org*, *rec*, *xmt*} asociadas a las variables T_1, T_2, y T_3, con ellas se mide el tiempo de desplazamiento y el retardo relativo entre el par de nodos A y B. Para iniciar el protocolo se tiene que $T_1=T_2=T_3= 0$ tanto en el nodo A como en el nodo B, y se realizan los siguientes pasos:

1. El nodo A envía un paquete que contiene solamente la etiqueta de tiempo t_1 de cuando se origina el paquete, y se copia T_1 en la variable de estado local *xmt*.
2. El nodo B recibe el paquete en un tiempo t_2 y copia t_1 en su variable T_1, afectando *org*, y la etiqueta de tiempo de recibido t_2 en la variable T_2 afectando *rec*.

3. En un tiempo t_3, de forma inmediata en modo cliente/servidor o en un tiempo posterior en modo simétrico, el nodo B responde transmitiendo al nodo A un paquete que contiene t_1, t_2, y la etiqueta de tiempo de transmisión t_3. Estas tres etiquetas de tiempo son copiadas a las variables T_1, T_2, T_3, afectando los estados *org*, *rec*, *xmt*, respectivamente.
4. Cuando el nodo A recibe el paquete en un tiempo t_4 actualiza las variables de estado, se verifica que $org \neq 0$ y que $t_1 = T_1$, y pasa los elementos $\{t_1, t_2, t_3, t_4\}$ para calcular el tiempo de desplazamiento y el retardo relativo de B:

$$desplazamiento = \theta = T(B) - T(A) = 1/2 * [(T_2 - T_1) + (T_3 - T_4)].$$
$$retardo = \delta = T(ABA) = (T_4 - T_1) - (T_3 - T_2).$$

Como la precisión de sincronización de tiempo en milisegundos de NTP (Peng et al., 2009) es insuficiente para muchas aplicaciones actuales, tales como transacciones financieras, sistemas de control y medición industrial, entre otras, la IEEE realizó el proyecto del estándar IEEE 1588 para la sincronización por medio del PTP que ha permitido trabajar a niveles de precisión en el rango de nanosegundos para la sincronización de los dispositivos en un SD empleando etiquetado de tiempo de paquetes a nivel de hardware (Wu & Peloquin, 2009).

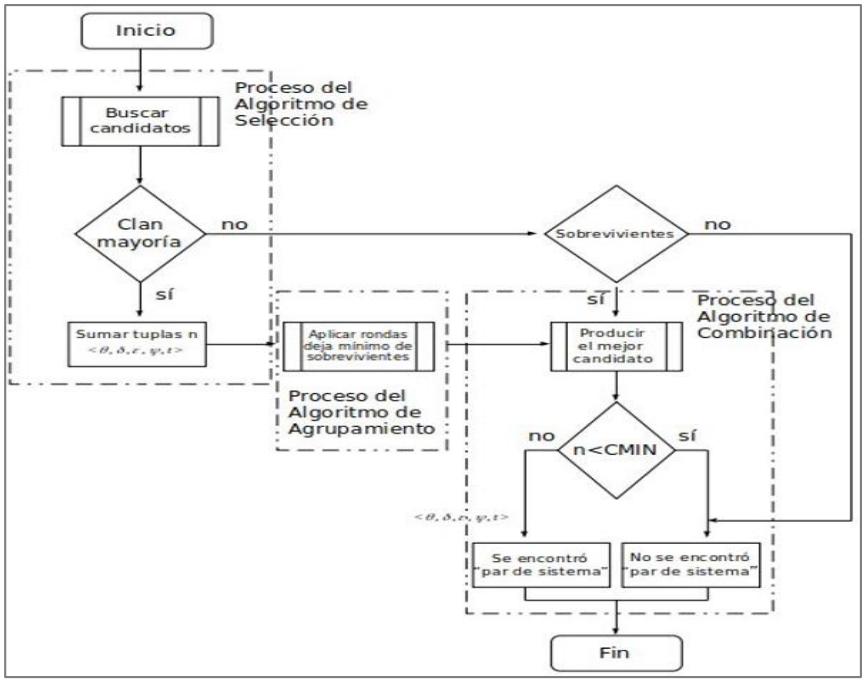

Figura 2. Resumen de operaciones del Proceso del Sistema.
Nota. Elaborada a partir de la información de *Network Time Protocol Version 4: Protocol and Algorithms Specification, por* Mills, 2010, Internet Engineering Task Force Report RFC-5905. University of Delaware.

Propuesta del IEEE 1588-Protocolo de Tiempo de Precisión

La primera versión del IEEE 1588 o PTP (IEEE Standar 1588-2002) se publicó en 2002, revisada posteriormente para dar paso a la versión 2 (IEEE Standar 1588-2008), después se le sumó adecuaciones para

trabajar tanto en IPv4 como en IPv6 así como funcionar en aplicaciones de velocidad a rango de sub-nanosegundos dando como resultado el estándar versión 2.1 (IEEE Standar 1588-2019) que fue adoptado por la Comisión Electrotécnica Internacional, IEC, y es conocido actualmente como el estándar IEC 61588:2021 (IEEE/IEC 61588-2021). Pandey et al. (2019) realizaron un análisis del PTP que se puede complementar con base en el estándar IEEE/IEC 61588 con las siguientes características:

1. Un SD PTP está formado por dispositivos habilitados con PTP y sin PTP.
2. Los dispositivos habilitados con PTP pueden ser un nodo administrador, o un dispositivo con un reloj interno que intercambia mensajes para conocer los valores de tiempo con otros dispositivos PTP. Los dispositivos con reloj interno se clasifican en: ordinario, borde, transparente de extremo a extremo, o transparente punto a punto.
3. Los mensajes PTP viajan encapsulados en UDP/IPv4, UDP/IPv6, IEEE 802.3/Ethernet, DeviceNET, o ControlNET. Estos paquetes son recibidos en entidades llamadas puertos que se modelan para admitir en una interfaz mensajes de la clase evento y en otros mensajes de la clase general.
4. Los mensajes PTP están formado por un encabezado, un cuerpo y un sufijo. El encabezado, de 34 bytes, tiene el campo *tipo de mensaje*, de 4 bits, que puede ser usado para especificar la clase evento, cuyo valor hexadecimal está entre 0 y 3: *Sync*, *Delay_Req*, *Pdelay_Req*, *Pdelay_Resp*; o la clase general, cuyo valor hexadecimal está entre 8 y D: *Follow_Up*, *Delay_Resp*, *Pdelay_Resp_Follow_Up*, *Announce*, *Signaling*, *Management*. Los mensajes eventos son mensajes cronometrados en la forma que se crean etiquetas de tiempo precisas tanto en el emisor como en el receptor, y los mensajes generales se encargan de relacionar dicho mensaje de evento.
5. Todos los mensajes que entran a los dispositivos PTP tipo reloj se procesan como entrada a una máquina de estado propuesta por el estándar.
6. La máquina de estado está formada por nueve estados: *inicialización*, *fallo*, *desactivado*, *escuchando*, *pre-maestro*, *maestro*, *pasivo*, *sin calibrar* y *esclavo*. Los estados por los que transita cada mensaje depende de la configuración del dispositivo. Las siguientes acciones se siguen cuando se configura el dispositivo para competir por ser maestro: el primero estado es el de *inicialización*, aquí entran los mensajes a ser procesados si se acaba de encender el dispositivo, se ha levantado de un fallo, se ha levantado de una desactivación o se manda a inicializar después de cualquier estado, como segundo estado y siendo esté el centro de la máquina de estado se encuentra el estado *escuchando*, aquí el dispositivo espera determinado tiempo para recibir un mensaje *Announce* de un dispositivo maestro, si ocurre algún error pasará al estado *fallo*, pero si pasa dicho tiempo y no recibe el mensaje transita al estado *pre-maestro*, y si recibe en ese estado un mensaje *Announce* de otra red verifica por medio del *Algoritmo del Mejor Reloj Maestro -Best Master Clock Algorithm*, BMCA, quién asumirá el rol de Gran Maestro y el otro pasará a estado *esclavo*. El estado *sin calibrar* se utiliza después de transitar el estado *escuchando* y saber quién es el dispositivo maestro.
7. El dispositivo reloj ordinario mantiene un conjunto de datos de su reloj y un conjunto de datos de su puerto único, y puede ser configurado para ser sólo esclavo o competir para ser maestro; el dispositivo reloj borde es designado para los dispositivos, conmutador/puente/enrutador, entre el maestro y los esclavos, normalmente tiene varios puertos físicos y cada puerto se comunica con la red por medio de dos interfaces lógicas para los mensajes eventos y los mensajes generales, y puede ser configurado para ser maestro y esclavo. El dispositivo reloj transparente extremo a extremo o E2E funciona como un sistema multipuertos que reenvía todos los mensajes, se usa como puente entre el maestro y el esclavo, con los mensajes eventos

se encarga de avanzar y ajustar los tiempos por la residencia del mensaje en el puente, colocando los nuevos tiempos en el campo de corrección del encabezado PTP. El dispositivo reloj transparente punto a punto o P2P funciona similar al E2E excepto en la forma de corregir y manejar los tiempos en los mensajes PTP, esto es, se encarga de avanzar y ajustar únicamente los mensajes *Sync* y *Follow_Up*, se corrigen de acuerdo a los tiempos de residencia en el puente y el tiempo de retardo se coloca en el campo de corrección del encabezado PTP.

8. El PTP se ejecuta en dos fases, *1) Establecer la jerárquica maestro-esclavo*: en un grupo lógico de relojes que se sincronizan por medio de PTP, llamado dominio, los relojes ordinarios y bordes ejecutan la máquina de estado y para transitar entre los estados examina el contenido del mensaje *Announce* que contiene la prioridad y la calidad del reloj, y aplican el BMCA para ser maestro o pasar a ser esclavo, con esto se autoorganizan en una jerarquía en la que el reloj con mayor prioridad y calidad estará en la parte superior, será el Gran Maestro, y los relojes que están en la parte inferior son los esclavos, para unirlos se encuentran los relojes de borde y los relojes transparentes (Watt et al., 2014); *2) Sincronizar los relojes con el Gran Maestro*: cada esclavo, reloj ordinario y borde, se sincroniza con su maestro basado en los siguientes pasos:

1. El Gran Maestro en un tiempo T1 registra y transmite:

1.1. un mensaje evento *Sync* donde coloca la etiqueta de tiempo T1, y

1.2. un mensaje general *Follow_Up* donde coloca T1.

2. El esclavo al recibir el mensaje registra el tiempo de su arribo T2, y transmite el mensaje evento *Delay_Req* al Gran Maestro donde coloca la etiqueta de tiempo T3 y la registra.

3. Al recibir el mensaje el Gran Maestro envía al esclavo el mensaje evento *Delay_Req* donde coloca la etiqueta de tiempo T4.

4. El esclavo al recibir el mensaje tiene cuatro etiquetas de tiempo para poder calcular

$$Retardo = [(T2 - T1) + (T4 - T3)]/2$$
$$Offset = [(T2 - T1) - (T4 - T3)]/2$$

Resumiendo, la propuesta el IEEE 1588 sigue la arquitectura maestro-esclavo que puede tener más de un segmento de red con múltiples relojes, generalmente, el reloj llamado el Gran Maestro tiene el mejor oscilador con la hora estándar y está asociado a un GPS externo o reloj atómico, y cada segmento o dominio cuenta con esclavos sincronizados con un maestro y este maestro está sincronizado con el Gran Maestro (Zeba et al., 2020). Las desventajas u oportunidades de mejoras a la sincronización planteada por la IEEE se enfrenta a tres grandes problemas, el primero implícito en la red, como los problemas de precisión debido al retardo que puede haber en la ruta, el jitter, y la dependencia de la ubicación de la toma de la etiqueta de tiempo en la pila del modelo de comunicación, el segundo es el hardware, como la resolución interna del reloj, y el tercero es la seguridad, como los ataques mencionados por Itkin & Wool (2020): suplantación de identidad en los mensajes *Delay_Response*, *Sync*, creación de mensajes *Announce* para proponerse como maestro, ataque a esclavos en el dominio por medio de mensajes *Sync* aplicando una clave simétrica legítima, y mensajes de administración falsos para cambiar la información del conjunto de datos de un nodo.

Discusión

En esta sección se confrontan y argumentan los resultados del estudio realizado con los resultados reportados en la literatura académica por otros investigadores, que abarcan la temática de análisis.

Los resultados obtenidos en nuestra investigación muestran una alta precisión en la sincronización de relojes mediante el uso de algoritmos descentralizados, lo cual está en consonancia con estudios previos. Por ejemplo, la revisión de algoritmos de sincronización por Gusella y Zatti (1989) destaca la efectividad de los enfoques descentralizados en reducir la carga de comunicación y mejorar la precisión temporal en sistemas distribuidos, algo que también se ha confirmado en nuestro análisis.

En cuanto a los algoritmos centralizados, los estudios de Cristian Flaviu (1989) han demostrado que su simplicidad y efectividad son ventajosas en ciertos contextos, aunque con limitaciones en la escalabilidad y tolerancia a fallos. Nuestros resultados concuerdan con estas observaciones, señalando que aunque los algoritmos centralizados pueden proporcionar una sincronización precisa en redes pequeñas y controladas, su aplicabilidad disminuye en sistemas distribuidos a gran escala debido a problemas de carga y puntos únicos de fallo.

Por otro lado, Lamport (1978) propuso la sincronización lógica, la cual se ha convertido en una base teórica sólida para muchos algoritmos modernos de sincronización. Nuestra investigación confirma que la sincronización lógica sigue siendo un componente esencial para la coordinación temporal, aunque su implementación práctica requiere complementarse con algoritmos que consideren la variabilidad de retrasos en la red, algo que ha sido abordado en estudios posteriores como los de Mills (1991) con el NTP.

El Protocolo de Tiempo de Precisión (PTP), estandarizado por el IEEE, ha mostrado una mayor precisión en la sincronización en entornos industriales y telecomunicaciones. Los estudios de Schwab y Schmidt (2008) destacan la importancia del PTP en aplicaciones que requieren sincronización en el rango de sub-microsegundos, una observación que nuestros resultados también corroboran, señalando su superioridad sobre el NTP en términos de precisión, aunque con mayores requerimientos de configuración y mantenimiento.

Finalmente, nuestros resultados también indican que no existe un algoritmo único que sea óptimo para todas las situaciones. La elección del algoritmo de sincronización debe basarse en las necesidades específicas del sistema distribuido, considerando factores como la escala, la tolerancia a fallos, y los requisitos de precisión temporal. Esto coincide con las conclusiones de investigaciones recientes, como las de Sommer y Wattenhofer (2009), que sugieren un enfoque adaptativo en la selección de algoritmos de sincronización.

Conclusiones

Dentro del conjunto de importancias a considerar que no debe faltar en un SD se encuentra la sincronización del tiempo de los relojes, esto es fundamental para poder compartir adecuadamente recursos y preservar sus estados, en el caso de no tener en funcionamiento un buen algoritmo de sincronización se puede caer en el problema de inconsistencia. Los algoritmos presentados, que utilizan un reloj físico para la sincronización, se pueden clasificar como algoritmos centralizados, asimétrico y pasivos, como el propuesto por Cristian, algoritmos centralizados, asimétrico y activos, como el propuesto por Gusella y Zatti, algoritmos distribuidos sin tolerancia a fallos, como el algoritmo de consistencia propuesto por Marzullo y Owichi, y algoritmos distribuidos con tolerancia a fallos, como el algoritmo de intersección propuesto por Marzullo y Owichi, las propuestas de Mills usadas en el desarrollo de NTP y el PTP de la IEEE.

Como se mostró en el documento, un elemento esencial en un SD es tener una secuencia correcta de eventos, ya sea en orden lógico, como lo indicado por Lamport, o en orden cronológico, como lo indica el resto de los algoritmos mencionados. Los algoritmos expuestos son incrustados en software indispensables para que los

dispositivos compartan un tiempo común y preciso, y pueden ser utilizados dentro de sistemas de transacciones financieras, control de operaciones, telecomunicaciones y sistemas de navegación, entre otros. Actualmente los algoritmos propuestos por Mills y usados en el NTPv4 hacen al protocolo robusto para la sincronización con tolerancia a fallo que puede automáticamente seleccionar la mejor de varias fuentes de tiempo en la Internet, pero su baja velocidad para aplicaciones industriales, hace que sea desplazado por el PTP para las aplicaciones en la Internet de las Cosas, IoT.

La investigación y desarrollo en el área de sincronización de relojes continua evolucionando con la expansión de las tecnologías de la IoT debido a que los requisitos varían según los escenarios en la industria (Dang et al., 2023) y se presentan nuevos desafíos para mejorar la precisión, rapidez y eficiencia de los algoritmos así como el desarrollo de nuevos protocolos. Para complementar este documento puede pensarse en realizar un estudio sobre las propuestas de algoritmos en la IoT que brindan sincronización en la Internet Industrial.

Referencias

Baldoni, R., Mostefaoui, A., & Raynal, M. (1996). Causal delivery of messages with real-time data in unrealiable networks. Journal of Real-Time Systems, 10, 245-262.

Baldoni, R. & Raynal, M. (2002). Fundamentals of Distributed Computing: A Practical Tour of Vector Clock Systems. IEEE Distributed Systems Online, 3(2).

Birman, K., & Joseph, T. (1987). Reliable Communication in the Presence of Failures. ACM Transactions on Computer Systems, 5(1), 47-76.

Chandy, K. & Lamport, L. (1985). Distributed Snapshots: Determining Global States of Distributed Systems. ACM Transactions on Computer Systems, 3(1), 63-75.

Cristian, F. (1989). Probabilistic clock synchronization. Distributed Computing, 3(3), 146-158.

Dalwadi, N. & Padole, M. (2017). Comparative Study of Clock Synchronization Algorithms in Distributed Systems. Advances in Computational Sciences and Technology, 10(6), 1941-1952.

Dang, F., Sun, X., Liu, K., Xu, Y., & Liu, Y. (2023). A Survey on Clock Synchronization in the Industrial Internet. Journal of Computer Science and Technology, 38(1), 146-165, doi: 10.1007/ s11390-023-2908-4.

Fidge, C. (1988). Timestamp in Message Passing Systems that Preserves Partial Ordering. Proc. 11Th Australian Computing Science Communication, 10(1), 56-66.

Gusella, R. & Zatti, S. (1987). The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3 BSD. Report No. UCB/CSD 87/33. Computer Science Division (EECS). University of California. Berkeley California.

IEEE Standar 1588-2002. (Noviembre 2002). IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. http://standards.ieee.org

IEEE Standar 1588-2008. (Julio 2008). IEEE Standar for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. http://standards.ieee.org

IEEE Standar 1588-2019. (Noviembre 2019). IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. http://standards.ieee.org

IEEE/IEC 61588-2021. (Junio 2021). IEC/IEEE International Standar- Precision Clock Synchronization Protocol for NetWorked Measurement and Control Systems. http://standards.ieee.org

Itkin, E., & Wool, A. (2020). A Security Analysis and Revised Security Extension for the Precision Time Protocol. IEEE Transactions on Dependable and Secure Computing, 17(1), 22-34, doi: 10.1109/TDSC.2017.2748583

Kumar, M. & Manjula, R. (2013). Performance Comparison of Physical Clock Synchronization Algorithms. International Journal of Engineering Sciences & Research, 2(3), 601-607.

Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, 21(7), 558-565.

Marzullo, K. & Owicki, S. (1983). Maintaining the Time in a Distributed System. Report No. 83-247. Computer Systems Laboratory. Departments of Electrical Engineering and Computer Science. Stanford University.

Mattern, F. (1988). Virtual Time and Global States of Distributed systems [Conference]. En Cosnard, Quinton, Raynal, & Robert (Eds). Proc. Workshop on Parallel and Distributed Algorithms, 215–226.

Mills, D. (1981). DCNET Internet Clock Service. DARPA Network Working Group Report RFC-778. COMSAT Laboratories.

Mills, D. (1983). DCN Local-Network Protocols. DARPA Network Working Group Report RFC-891. M/A-COM Linkabit.

Mills, D. (1985a). Network Time Protocol (NTP). DARPA Network Working Group Report RFC-958. M/A-COM Linkabit.

Mills, D. (1985b). Algorithms for Synchronizing Network Clocks. DARPA Network Working Group Report RFC-956. M/A-COM Linkabit.

Mills, D. (1985c). Experiments in Network Clocks Synchronization. DARPA Network Working Group Report RFC-957. M/A-COM Linkabit.

Mills, D. (1988). Network Time Protocol (Version 1) Specification and Implementation. DARPA Network Working Group Report RFC-1059. University of Delaware.

Mills, D. (1989a). Network Time Protocol (Version 2) Specification and Implementation. DARPA Network Working Group Report RFC-1119. University of Delaware.

Mills, D. (1989b). Measured Performance of the Network Time Protocol in the Internet System. DARPA Network Working Group Report RFC-1128. University of Delaware.

Mills, D. (1991). Internet Time Synchronization: The Network Time Protocol. IEEE Transactions on Commmunications, 39(10), 1482-1493. doi: 10.1109/26.103043

Mills, D. (1992). Network Time Protocol (Version 3) Specification, Implementation and Analysis. DARPA Network Working Group Report RFC-1305. University of Delaware.

Mills, D. (1996). Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI. DARPA Network Working Group Report RFC-2030. University of Delaware.

Mills, D. (2003). A Brief History of NTP time: Memoirs of an Internet timekeeper. Computer Communication Review, 33, 9-21.

Mills, D. (2010a). Network Time Protocol Version 4: Protocol and Algorithms Specification. Internet Engineering Task Force Report RFC-5905. University of Delaware.

Mills, D. (2010b). Network Time Protocol Version 4: Autokey Specification. Internet Engineering Task Force, RFC-5906. University of Delaware.

Pandey, P., Pratap, B. & Pandey, R. (2019). Analysis and Design of Precision Time Protocol System Based on IEEE1588 Standars. Proceedings of the Fourth International Conference on Communication and Electronics Systems. 1963-1967.

Peng, Y., Luo, Q., & Liu, Z. (2009). An automatic evaluation system for IEEE 1588 synchronization clock unit. Proceeding of the Ninth International Conference on Electronic Measurement and Instruments, 408-413.

Singhal, M. & Kshemkalyani, A. (1992). An efficient implementation of vector clocks. Information Processing Letters, 43(1), 47-52.

Watt, S., Achanta, S., Abubakari, H., & Sagen, E. (2014). Understanding and Applying Precision Time Protocol. Power and Energy Automation Conference. Schweitzer Engineering Laboratories, Inc.

Wu, J. & Peloquin, R. (2009). Synchronizing Device Clocks Using IEEE 1588 and Blackfin Embedded Processors. Analog Dialogue, 43(11), 1-5.

Zeba, I., Granados, J., Sun, Y., Latif, S., Gong, L., Zou, Z., & Zheng, L. (2020). IEEE 1588 for Clock Synchronization in Industrial IoT and Related Applications: A Review on Contributing Technologies, Protocols and Enhancement Methodologies. IEEE Access, 8, 155660-155678. doi: 10.1109/ACCESS.2020.3013669