



# UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

## APLICACIONES DISTRIBUIDAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

---

# Algoritmos de Sincronización

---

*Estudiantes:*

Sebastián Arias, Christopher Bazurto, Stephen Drouet, Josué Merino

*Docente:*

Ing. Darío Morales

# Índice general

<b>1. Objetivos</b>	<b>3</b>
1.1. Objetivo Principal . . . . .	3
1.2. Objetivos Específicos . . . . .	3
<b>2. Desarrollo</b>	<b>4</b>
2.1. NTP . . . . .	4
2.1.1. Código . . . . .	4
2.1.2. Explicación . . . . .	5
2.1.3. Capturas y Evidencias . . . . .	6
2.2. Cristian's Algorithm . . . . .	6
2.2.1. ¿Cómo funciona el Algoritmo de Cristian? . . . . .	6
2.2.2. Código . . . . .	7
2.2.3. Explicación . . . . .	10
2.2.4. Capturas y Evidencias . . . . .	10
2.3. Algoritmo de Berkeley . . . . .	11
2.3.1. Código . . . . .	12
2.3.2. Explicación . . . . .	16
2.3.3. Capturas y Evidencias . . . . .	16
2.4. Enlaces . . . . .	16
<b>3. Conclusiones</b>	<b>17</b>

# Índice de figuras

2.1. NTP . . . . .	6
2.2. Ejecución del Servidor . . . . .	11
2.3. Ejecución del Cliente . . . . .	11
2.4. Hora enviada al cliente desde el servidor . . . . .	11
2.5. Algoritmo de Berkeley . . . . .	16

# 1. Objetivos

## 1.1. Objetivo Principal

Implementar la sincronización de relojes en un sistema distribuido mediante tres métodos: NTP, el algoritmo de Cristian y el algoritmo de Berkeley, con el fin de lograr una sincronización precisa entre múltiples nodos de un sistema.

## 1.2. Objetivos Específicos

- Implementar el cliente NTP: Desarrollar una aplicación que consulte un servidor de tiempo (como "time.google.com") y ajuste el reloj local en base a la hora proporcionada por dicho servidor.
- Implementar el Algoritmo de Cristian: Crear una aplicación cliente-servidor donde el cliente consulte a un servidor de tiempo para estimar la hora exacta, teniendo en cuenta la latencia de la red.
- Simular el Algoritmo de Berkeley: Desarrollar una simulación en la que varios nodos (hilos o instancias) ajusten sus relojes mediante la colaboración para llegar a un tiempo promedio.
- Generar el Informe: Documentar el proceso de implementación y funcionamiento de cada algoritmo, explicando las fases de sincronización y los resultados obtenidos.
- Proveer Evidencias: Incluir capturas de pantalla o videos demostrando los resultados de la sincronización de los relojes.

## 2. Desarrollo

### 2.1. NTP

Es un protocolo de Internet para sincronizar los relojes de los sistemas informáticos a través del enrutamiento de paquetes en redes con latencia variable. NTP utiliza UDP como su capa de transporte, usando el puerto 123. Está diseñado para resistir los efectos de la latencia variable. [1]

#### 2.1.1. Código

Al crear el proyecto se agregan las dependencias en pom.xml

```
<dependency>
  <groupId>commons-net</groupId>
  <artifactId>commons-net</artifactId>
  <version>3.7</version>
</dependency>
```

Se crea el archivo NTPClient:

```
import org.apache.commons.net.ntp.NTPUDPClient;
import org.apache.commons.net.ntp.TimeInfo;
import java.net.InetAddress;
import java.util.Date;

class NTPClient {
  public static void main(String[] args) {
    try {
```

```

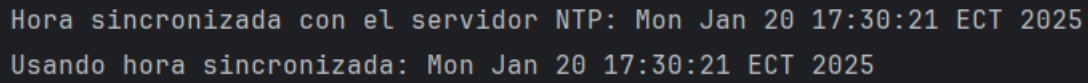
        String ntpServer = "time.google.com";
        NTPUDPClient client = new NTPUDPClient();
        client.setDefaultTimeout(10000);
        InetAddress inetAddress = InetAddress.getByName(ntpServer);
        TimeInfo timeInfo = client.getTime(inetAddress);
        long returnTime = timeInfo.getMessage().getTransmitTimeStamp().getTime();
        Date currentDate = new Date(returnTime);
        System.out.println("Hora sincronizada con el servidor NTP: "
            + currentDate);
    } catch (Exception e) {
        System.err.println("Error al sincronizar con el servidor NTP: "
            + e.getMessage());
    }
}
}

```

### 2.1.2. Explicación

- Librería NTP: Se usa la librería commons-net para interactuar con el servidor NTP.
- NTPUDPClient: Es la clase que proporciona la funcionalidad para consultar un servidor NTP a través de UDP.
- TimeInfo: Representa la respuesta del servidor, que incluye la hora en el servidor.
- Conexión al servidor: El cliente se conecta a time.google.com y consulta el tiempo.
- Tiempo: La respuesta contiene el tiempo de transmisión, que se convierte en un objeto Date para mostrar la hora.
- Manejo de excepciones: Si ocurre algún error, se captura y muestra un mensaje.

### 2.1.3. Capturas y Evidencias



```
Hora sincronizada con el servidor NTP: Mon Jan 20 17:30:21 ECT 2025
Usando hora sincronizada: Mon Jan 20 17:30:21 ECT 2025
```

Figura 2.1: NTP

## 2.2. Cristian's Algorithm

Es un método, dentro de la computación distribuida, para la sincronización de relojes. Cristian describe el método como probabilístico debido a que se consigue sincronización solo si el tiempo de respuesta es suficientemente corto comparado con la precisión requerida. Consiste en un servidor conectado a una fuente de UTC y unos clientes que se sincronizan con dicho servidor. [2]

### 2.2.1. ¿Cómo funciona el Algoritmo de Cristian?

El algoritmo de Cristian se basa en los siguientes pasos:

1. El cliente envía una solicitud de tiempo al servidor: El cliente le pide al servidor su hora actual.
  2. El servidor responde con su hora: El servidor devuelve su hora actual (típicamente utilizando NTP o un mecanismo similar).
  3. El cliente recibe la hora del servidor: El cliente recibe la respuesta del servidor junto con el tiempo que ha pasado desde que hizo la solicitud.
  4. El cliente estima la hora del servidor: Usando la hora que le devolvió el servidor y la latencia de la red, el cliente calcula una estimación de la hora exacta del servidor.
- $T_1$  = Tiempo de solicitud enviado por el cliente
  - $T_2$  = Tiempo de respuesta recibido por el servidor
  - $T_3$  = Tiempo de respuesta recibido por el cliente

- $T_4$  = Tiempo de solicitud recibido por el servidor

La estimación de la hora exacta del servidor es:

$$T_s = T_2 + \frac{T_3 - T_1}{2}$$

### 2.2.2. Código

Se crea el servidor:

```
package com.espe.cristian;

import java.io.*;
import java.net.*;
import java.util.Date;

class TimeServer {
    public static void main(String[] args) {
        try {
            // Puerto en el que el servidor escuchará
            int port = 12345;
            DatagramSocket socket = new DatagramSocket(port);
            System.out.println("Servidor esperando conexiones...");

            // Bucle para recibir solicitudes de los clientes
            while (true) {
                byte[] receiveData = new byte[1024];

                // Recibir la solicitud del cliente
                DatagramPacket receivePacket =
                    new DatagramPacket(receiveData, receiveData.length);
                socket.receive(receivePacket);

                // Obtener la hora actual
```



```

        long serverTime = System.currentTimeMillis();
        String serverTimeString = Long.toString(serverTime);

        // Enviar la respuesta con la hora del servidor
        InetAddress clientAddress = receivePacket.getAddress();
        int clientPort = receivePacket.getPort();
        DatagramPacket sendPacket =
            new DatagramPacket(serverTimeString.getBytes(),
                serverTimeString.length(), clientAddress, clientPort);
        socket.send(sendPacket);
        System.out.println("Hora enviada al cliente: " + serverTime);
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

Paso siguiente se crea el cliente:

```

package com.espe.cristian;

import java.net.*;
import java.util.Date;

class TimeClient {
    public static void main(String[] args) {
        try {
            // Dirección y puerto del servidor
            String serverAddress = "localhost";
            int serverPort = 12345;

            // Crear el socket
            DatagramSocket socket = new DatagramSocket();

```

```

// Establecer el tiempo de solicitud (T1)
long T1 = System.currentTimeMillis();
byte[] sendData = new byte[1024];
String requestMessage = "Hora solicitada";
sendData = requestMessage.getBytes();

// Enviar la solicitud al servidor
InetAddress serverInetAddress = InetAddress.getByName(serverAddress);
DatagramPacket sendPacket =
new DatagramPacket(sendData, sendData.length,
serverInetAddress, serverPort);
socket.send(sendPacket);
System.out.println("Solicitud enviada al servidor...");

// Recibir la respuesta del servidor
byte[] receiveData = new byte[1024];
DatagramPacket receivePacket =
new DatagramPacket(receiveData, receiveData.length);
socket.receive(receivePacket);
long T3 = System.currentTimeMillis();
// Hora cuando se recibe la respuesta

// Obtener la hora del servidor (T2)
String serverTimeString =
new String(receivePacket.getData(), 0,
receivePacket.getLength());
long T2 = Long.parseLong(serverTimeString);

// Calcular la hora estimada del servidor (T_s)
long T_s = T2 + ((T3 - T1) / 2);

```

```

        // Mostrar los resultados
        System.out.println("Hora solicitada por el cliente (T1): "
+ new Date(T1));
        System.out.println("Hora recibida del servidor (T2): "
+ new Date(T2));
        System.out.println("Hora recibida por el cliente (T3): "
+ new Date(T3));
        System.out.println("Hora estimada del servidor (T_s): "
+ new Date(T_s));

        socket.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

### 2.2.3. Explicación

El servidor escucha en el puerto 12345. Recibe la solicitud del cliente y obtiene la hora actual del sistema. Envía esa hora de vuelta al cliente. El cliente envía una solicitud al servidor. Mide el tiempo T1 (cuando se envía la solicitud) y T3 (cuando se recibe la respuesta). Usa la hora T2 proporcionada por el servidor para estimar la hora exacta del servidor utilizando la fórmula:

$$T_s = T_2 + \frac{T_3 - T_1}{2}$$

### 2.2.4. Capturas y Evidencias

Se ejecuta primero el servidor:

A terminal window with a dark background and light-colored text. The text reads "Servidor esperando conexiones..." in a monospaced font.

Figura 2.2: Ejecución del Servidor

Después se ejecuta el cliente:

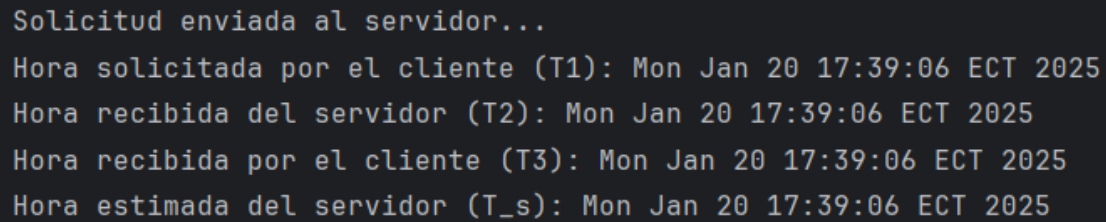
A terminal window with a dark background and light-colored text. The text shows the client's execution details: "Solicitud enviada al servidor...", "Hora solicitada por el cliente (T1): Mon Jan 20 17:39:06 ECT 2025", "Hora recibida del servidor (T2): Mon Jan 20 17:39:06 ECT 2025", "Hora recibida por el cliente (T3): Mon Jan 20 17:39:06 ECT 2025", and "Hora estimada del servidor (T\_s): Mon Jan 20 17:39:06 ECT 2025".

Figura 2.3: Ejecución del Cliente

El servidor muestra la hora enviada al cliente:

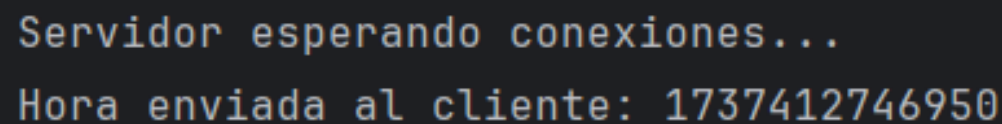
A terminal window with a dark background and light-colored text. The text shows the server's output: "Servidor esperando conexiones..." and "Hora enviada al cliente: 1737412746950".

Figura 2.4: Hora enviada al cliente desde el servidor

1737412746950 es el número de milisegundos desde el 1 de enero de 1970 a la medianoche (hora UTC). En términos prácticos, esto es el equivalente a una fecha y hora específica.

## 2.3. Algoritmo de Berkeley

Se trata de un algoritmo de sincronización de relojes diseñado por Gusella y Zatti en 1989. Dicho algoritmo se creó para entornos en los cuales no se tienen receptores de tiempo UTC, de forma que, gracias a este algoritmo se pueden mantener los relojes del entorno sincronizados con la misma hora. Este algoritmo sincroniza procesos en un sistema distribuido y garantiza que estos mismos se ejecuten de manera cronológica y secuencial (respetando el orden de los eventos en el sistema). Este algoritmo se categoriza como un algoritmo de sincronización de relojes físicos e internos.

La base de todos estos tipos de algoritmos es la comunicación del tiempo de reloj de cada nodo. Cada uno calcula alguna función de tipo promedio o la mediana de todos los valores, si la diferencia con su reloj actual es mayor que la desviación máxima permitida se actualiza el reloj con el nuevo valor. Posteriormente el algoritmo se ejecutará de nuevo hasta lograr una convergencia de todos los nodos. [3]

### 2.3.1. Código

Se crea un BerkeleyServer.java para manejar la lógica del servidor, que recibe los tiempos de los nodos, calcula el tiempo promedio y ajusta los relojes de los nodos.

```
package com.espe.berkeley;

import java.text.SimpleDateFormat;
import java.util.*;

public class BerkeleyServer {

    private static final int NUM_NODES = 5; // Número de nodos en el sistema
    private List<ClockNode> nodes; // Lista de nodos
    private long averageTime; // Tiempo promedio calculado

    public BerkeleyServer() {
        nodes = new ArrayList<>();
        // Crear y añadir nodos al servidor
        for (int i = 1; i <= NUM_NODES; i++) {
            nodes.add(new ClockNode(i, this));
        }
    }

    // Método para recibir los tiempos de los nodos y calcular el promedio
    public void receiveTimesAndCalculateAverage() {
        long total = 0;
        for (ClockNode node : nodes) {
            total += node.getLocalTime();
        }
    }
}
```

```

        System.out.println("Servidor recibe tiempo del nodo "
            + node.getNodeId() + ": "
            + formatTime(node.getLocalTime()));
    }

    averageTime = total / nodes.size();
    System.out.println("Servidor calcula el tiempo promedio: "
        + formatTime(averageTime));
}

// Método para enviar el tiempo promedio a los nodos
public void adjustNodeTimes() {
    for (ClockNode node : nodes) {
        node.adjustTime(averageTime);
    }
}

// Formato para mostrar el tiempo legible
private String formatTime(long timeInMillis) {
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
    Date date = new Date(timeInMillis);
    return sdf.format(date);
}

// Método principal para ejecutar la simulación
public static void main(String[] args) {
    BerkeleyServer server = new BerkeleyServer();

    // Recibir tiempos de los nodos y calcular el promedio
    server.receiveTimesAndCalculateAverage();

    // Ajustar los tiempos de los nodos según el tiempo promedio

```

```

        server.adjustNodeTimes();
    }
}

```

Se crea una clase `ClockNode.java` que representa a cada nodo en el sistema con su propio reloj. Cada nodo tiene un desfase aleatorio simulado para mostrar cómo los relojes pueden desincronizarse antes de que el servidor los sincronice.

```

package com.espe.berkeley;

import java.text.SimpleDateFormat;
import java.util.*;

public class ClockNode {
    private int nodeId;
    private long localTime; // Tiempo local del nodo
    private BerkeleyServer server; // Referencia al servidor

    public ClockNode(int nodeId, BerkeleyServer server) {
        this.nodeId = nodeId;
        this.server = server;

        // Simular un desfase aleatorio de -500 a 500 ms
        Random random = new Random();
        this.localTime = System.currentTimeMillis() + random.nextInt(1000) - 500;
    }

    // Obtener el tiempo local del nodo
    public long getLocalTime() {
        return localTime;
    }

    // Obtener el ID del nodo
    public int getNodeId() {

```

```

        return nodeId;
    }

    // Ajustar el reloj del nodo según el tiempo promedio
    public void adjustTime(long averageTime) {
        localTime = averageTime;
        System.out.println("Nodo " + nodeId + " ajusta su tiempo a: "
            + formatTime(localTime));
    }

    // Formato para mostrar el tiempo legible
    private String formatTime(long timeInMillis) {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
        Date date = new Date(timeInMillis);
        return sdf.format(date);
    }
}

```

Main.java: Este archivo contiene el método main donde se crea el servidor y se ejecuta el proceso de sincronización de los nodos.

```

package com.espe.berkeley;

public class Main {
    public static void main(String[] args) {
        BerkeleyServer server = new BerkeleyServer();

        // Recibir tiempos de los nodos y calcular el promedio
        server.receiveTimesAndCalculateAverage();

        // Ajustar los tiempos de los nodos según el tiempo promedio
        server.adjustNodeTimes();
    }
}

```



```
Servidor recibe tiempo del nodo 1: 2025-01-20 17:48:20.810
Servidor recibe tiempo del nodo 2: 2025-01-20 17:48:20.330
Servidor recibe tiempo del nodo 3: 2025-01-20 17:48:20.438
Servidor recibe tiempo del nodo 4: 2025-01-20 17:48:20.221
Servidor recibe tiempo del nodo 5: 2025-01-20 17:48:20.467
Servidor calcula el tiempo promedio: 2025-01-20 17:48:20.453
Nodo 1 ajusta su tiempo a: 2025-01-20 17:48:20.453
Nodo 2 ajusta su tiempo a: 2025-01-20 17:48:20.453
Nodo 3 ajusta su tiempo a: 2025-01-20 17:48:20.453
Nodo 4 ajusta su tiempo a: 2025-01-20 17:48:20.453
Nodo 5 ajusta su tiempo a: 2025-01-20 17:48:20.453
```

Figura 2.5: Algoritmo de Berkeley

### 2.3.2. Explicación

- Simula un sistema con 5 nodos.
- El servidor recibe los tiempos de los nodos, calcula el tiempo promedio y luego ajusta los relojes de los nodos a ese tiempo promedio.
- Los tiempos de los nodos son inicialmente desincronizados debido a un desfase aleatorio.

### 2.3.3. Capturas y Evidencias

## 2.4. Enlaces

Repositorio: <https://github.com/ejmerino/algoritmos-de-sincronizacion>

### 3. Conclusiones

- La implementación del cliente NTP demostró ser efectiva para sincronizar el reloj local con precisión al consultar servidores de tiempo confiables, como "time.google.com". El protocolo NTP es robusto y se adapta a diferentes latencias de red sin comprometer la precisión.
- Al considerar la latencia de red, fue exitoso para estimar un tiempo preciso, aunque la precisión depende de la estabilidad y la latencia de la conexión. Este método permite obtener una sincronización más exacta que simplemente consultar un servidor de tiempo sin considerar la red.
- La simulación del algoritmo de Berkeley en un sistema distribuido con varios nodos fue exitosa. El ajuste de los relojes de los nodos para alcanzar un promedio proporcionó un buen resultado en términos de precisión en sistemas distribuidos. Este algoritmo demuestra la importancia de la colaboración entre nodos para mantener la sincronización a través de redes no confiables.
- Todos los métodos implementados demostraron que, a pesar de las variaciones en la latencia de red o el número de nodos, es posible lograr una sincronización precisa de relojes en sistemas distribuidos, lo cual es fundamental en aplicaciones que requieren alta precisión temporal.
- Las capturas de pantalla y videos mostraron claramente la sincronización exitosa de los relojes entre el cliente y los servidores, y entre los nodos en el caso del algoritmo de Berkeley, lo que valida la correcta implementación de los algoritmos.

# Bibliografía

- [1] Tecnozero. (2023, 25 abril). Servidor NTP o Servidor de hora. Tecnozero Soluciones Informaticas. <https://www.tecnozero.com/servidor/ntp/>
- [2] colaboradores de Wikipedia. (2023, 31 enero). Algoritmo de Cristian. Wikipedia, la Enciclopedia Libre. [https://es.wikipedia.org/wiki/Algoritmo\\_de\\_Cristian#:~:text=El%20algoritmo%20de%20Cristian%20\(1989,comparado%20con%20la%20precisi%C3%B3n%20requerida.](https://es.wikipedia.org/wiki/Algoritmo_de_Cristian#:~:text=El%20algoritmo%20de%20Cristian%20(1989,comparado%20con%20la%20precisi%C3%B3n%20requerida.)
- [3] colaboradores de Wikipedia. (2022, 2 julio). Algoritmo de Berkeley. Wikipedia, la Enciclopedia Libre. [https://es.wikipedia.org/wiki/Algoritmo\\_de\\_Berkeley](https://es.wikipedia.org/wiki/Algoritmo_de_Berkeley)