NTP网络时间协议实验报告

1 实验目的

本次实验的目的是通过深入理解网络时间协议（NTP）及其相关的时间服务算法原理，掌握其设计和实现方法。NTP协议是分布式计算机系统中一种常见的时间同步协议，用于通过网络确保计算机时钟的准确性和一致性。实验中将通过Java实现一个NTP客户端来模拟客户端和NTP服务器之间的时间同步过程，学习如何发送时间请求、接收响应，并计算网络延迟和调整本地时钟。

2 实验内容

2.1 技术原理

2.1.1 概述

网络时间协议（NTP）位于OSI模型的应用层，其在数据网络潜伏时间可变的计算机系统之间，通过分组交换实现客户端和服务器之间的时间同步，NTP服务器从权威时钟源（例如原子钟、GPS）接收精确的协调世界时UTC，客户端再从服务器请求和接收时间。

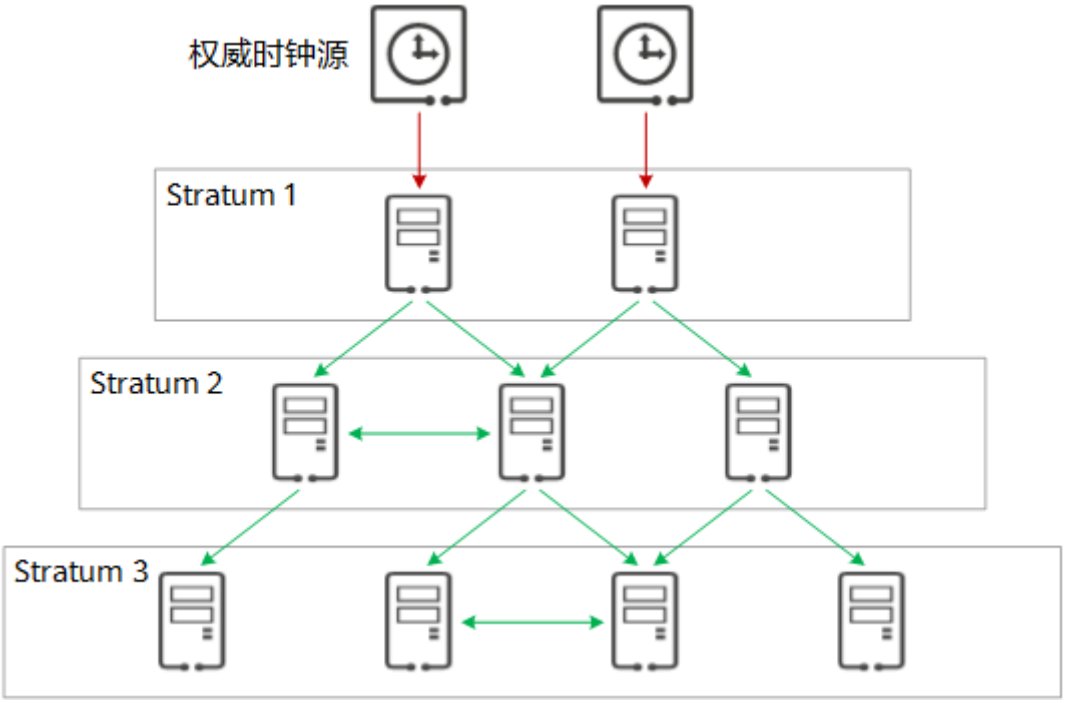
NTP基于UDP报文进行传输，使用的UDP端口号为123。

- 目的：将参与计算机的协调时间时（UTC）时间同步到几毫秒误差内；
- 设计：使用Marzullo算法的修改版，选择准确的时间服务器，减轻可变网络延迟造成的影响；
- 效果：在公共互联网保持几十毫秒的误差，在理想的局域网环境中实现超过1毫秒的精度；
- 版本：NTPv4，是RFC 5905文档中的建议标准。

2.1.2 时钟层

NTP使用一个分层、半分层的时间源系统。该层次的每个级别被称为“stratum”，顶层分配为数字0。一个通过阶层n同步的服务器将运行在阶层n + 1。

通常将从权威时钟获得时钟同步的NTP服务器的层数设置为Stratum 1，并将其作为主时间服务器，为网络中其他的设备提供时钟同步。而Stratum 2则从Stratum 1获取时间，Stratum 3从Stratum 2获取时间，以此类推。时钟层数的取值范围为1～16，取值越小，时钟准确度越高。层数为1～15的时钟处于同步状态；层数为16的时钟被认为是未同步的，不能使用的。



- **Stratum 0**：高精度计时设备，例如原子钟（如铯、铷）、GPS时钟或其他无线电时钟。它们生成精确的脉冲秒信号，触发所连接计算机上的中断和时间戳。阶层0设备也称为参考（基准）时钟。
- **Stratum 1**：与Stratum 0设备相连、在几微秒误差内同步系统时钟的计算机。Stratum 1服务器可能与其他Stratum 1服务器对等相连，以进行完整性检查和备份。它们也被称为主要（primary）时间服务器。
- **Stratum 2及更高**：网络中的普通计算机，它们从更高级别的Stratum服务器同步时间，并为其他设备提供时间信息。这些服务器可能分布在全球各地，构成了分布式的时间同步体系。

Stratum的上限为15；Stratum 16被用于标识设备未同步。每台计算机上的NTP算法相互构造一个贝尔曼-福特算法最短路径生成树，以最小化所有客户端到阶层1服务器的累积往返延迟。

2.1.3 时间戳

NTP使用64比特的时间戳，其中32位表示秒，32位表示秒的小数，给出一个每232秒（136年）才会翻转的时间尺度，理论分辨率2−32秒（233皮秒）。NTP以1900年1月1日作为开始时间，因此第一次翻转将在2036年2月7日发生。

NTP的未来版本可能将时间表示扩展到128位：其中64位表示秒，64位表示秒的小数。当前的NTPv4格式支持“时代数字”（Era Number）和“时代偏移”（Era Offset），正确使用它们应该有助于解决日期翻转问题。

2.1.4 时间同步算法

典型的NTP客户端将定期轮询不同网络上的三个或更多服务器。步骤如下：

1. 客户端向服务端发送一个NTP请求报文，其中包含了该报文离开客户端的时间戳t_1；
2. NTP请求报文到达NTP服务器，此时NTP服务器的时刻为t_2；
3. 当服务端接收到该报文时，NTP服务器处理之后，于t_3时刻发出NTP应答报文。该应答报文中携带报文离开NTP客户端时的时间戳t_1、到达NTP服务器时的时间戳t_2、离开NTP服务器时的时间戳t_3；
4. 客户端在接收到响应报文时，记录报文返回的时间戳t_4。

为同步其时钟，客户端必须计算其时间偏移量和来回通信延迟。有：

- 往返延迟“δ”：$\delta = (t_4 - t_1) - (t_3 - t_2)$
- 时间偏移“θ”：$\theta = \frac{(t_2 - t_1) + (t_3 - t_4)}{2}$

解得时间差：$offset = \frac{(t_2-t_1)+(t_3-t_4)}{2}$

“θ”和“δ”的值通过过滤器并进行统计分析。异常值被剔除，并从最好的三个剩余候选中导出估算的时间偏移。然后调整时钟频率以逐渐减小偏移，创建一个反馈回路。

当客户端和服务器之间的输入和输出路由都具有对称的标称延迟时，同步是正确的。如果路由没有共同的标称延迟，则将差异取半作为测量误差。

2.2 技术方案与实现

2.2.1 NTP客户端程序

2.2.1.1 类和常量

```
private static int NTP_Port = 123;          // 默认端口号，通常是 123
private static int NTP_PACKET_SIZE = 48;     // NTP 数据包的大小，固定为 48 字节
private static long SeventyYears = 2208988800L;  // 1900 年到 1970 年的秒数
```

2.2.1.2 类中的成员变量

```
DatagramSocket m_TimeService_Socket;     // 用于与 NTP 服务器通信的 DatagramSocket 对象
InetAddress m_TimeService_IPAddress;     // 保存 NTP 服务器的 IP 地址
Boolean m_bNTP_Client_Started;           // 指示 NTP 客户端是否已启动的标志
private Boolean m_bTimeServiceAddressSet;     // 标志，用于指示是否已成功设置 NTP 服务器的地址
```

2.2.1.3 枚举类型

```
public enum NTP_Client_ResultCode {NTP_Success, NTP_ServerAddressNotSet, NTP_SendFailed, NTP_ReceiveFailed};
```

- `NTP_Success`：成功获取时间戳。
- `NTP_ServerAddressNotSet`：未设置 NTP 服务器地址。
- `NTP_SendFailed`：发送请求失败。
- `NTP_ReceiveFailed`：接收响应失败。

2.2.1.4 内部类

- `NTP_Timestamp_Data`类：存储 NTP 时间戳的数据

```
public final class NTP_Timestamp_Data{
    public NTP_Client_ResultCode eResultCode;     // 操作结果的状态码
    public long lUnixTime;     // Unix 时间戳（自 1970 年以来的秒数）
    public long lHour;         // UTC时间的小时
    public long lMinute;         // UTC时间的分钟
    public long lSecond;         // UTC时间的秒

    NTP_Timestamp_Data()
    {
        eResultCode = NTP_Client_ResultCode.NTP_ServerAddressNotSet;
        lHour = 0;
        lMinute = 0;
        lSecond = 0;
        lUnixTime = 0;
    }
```

```
};
```

2.2.1.5 构造函数

```
public NTP_Client(){
    m_bTimeServiceAddressSet = false;      // 是否已经设置了时间服务器地址：没有
    m_bNTP_Client_Started = false;         // 客户端是否已经启动：没有
}
```

2.2.1.6 主要方法

1. 创建`DatagramSocket`发送和接收UDP数据包：`CreateSocket()`方法

```
public Boolean CreateSocket(){
    try
    {
        m_TimeService_Socket = new DatagramSocket();
        m_TimeService_Socket.setSoTimeout(500); // 设置超时时间为500毫秒
    }
    catch (SocketException Ex)
    {
        return false;
    }
    return true;
}
```

2. 解析IP地址：`SetUp_TimeService_AddressStruct(String sURL)`方法

```
public InetAddress SetUp_TimeService_AddressStruct(String sURL){
    String sFullURL = "http://" +sURL;
    try
    {
        m_TimeService_IPAddress = InetAddress.getByName(new
URL(sFullURL).getHost());
        m_bTimeServiceAddressSet = true;
    }
    catch (Exception Ex)
    {
        return null;
    }
    return m_TimeService_IPAddress;
}
```

3. 返回端口号：`GetPort()`方法

```
public int GetPort(){
    return NTP_Port;
}
```

4. 发送请求、接收响应：`Get_NTP_Timestamp()`方法

```
public NTP_Timestamp_Data Get_NTP_Timestamp(){
    // 1. 创建一个新的 NTP_Timestamp_Data 对象：存储时间戳数据及操作结果
    NTP_Timestamp_Data NTP_Timestamp = new NTP_Timestamp_Data();
```

```
    // 2. 检查 NTP 服务器的地址是否已设置
    if(true == m_bTimeServiceAddressSet)
    {
        // 3. 调用 Send_TimeService_Request() 方法向 NTP 服务器发送请求：
        // 如果请求成功，返回 true，否则返回 false
        if(true == Send_TimeService_Request())
        {
            // 4. 发送请求成功，执行以下操作：
            // 调用 Receive() 方法接收来自 NTP 服务器的响应，并将响应的数据填充到
NTP_Timestamp 对象中
            NTP_Timestamp = Receive(NTP_Timestamp);
            // 5. 检查接收到的时间戳是否有效：
            // 若有效，返回包含时间戳数据和结果代码的 NTP_Timestamp 对象
            if(0 != NTP_Timestamp.lUnixTime)
            {
                NTP_Timestamp.eResultCode = NTP_Client_ResultCode.NTP_Success;
                return NTP_Timestamp;
            }
            // 若无效，返回包含失败结果的 NTP_Timestamp 对象
            NTP_Timestamp.eResultCode = NTP_Client_ResultCode.NTP_ReceiveFailed;
            return NTP_Timestamp;
        }
        NTP_Timestamp.eResultCode = NTP_Client_ResultCode.NTP_SendFailed; //
Signal that the send operation failed (Time server was not contacted)
        return NTP_Timestamp;
    }
    NTP_Timestamp.eResultCode = NTP_Client_ResultCode.NTP_ServerAddressNotSet;
// Signal that Time server address has not been set, cannot get NTP timestamp
    return NTP_Timestamp;
}
```

5. 构造并发送一个NTP请求数据包：`Send_TimeService_Request()`方法

```
Boolean Send_TimeService_Request(){
    byte[] bSendBuf = new byte [NTP_PACKET_SIZE]; // 48字节的UDP 数据包
    bSendBuf[0] = (byte) 0xE3;    // 0b11100011，0xE3 表示版本号为 4，模式为客户端请求
    // LI bits（第 7 和第 6 位）：11 表示时钟未同步
    // Version bits（第 5、4、3 位）：100 表示使用的是 NTP 协议的版本 4
    // Mode bits（第 2、1、0 位）：011 表示这是由客户端发送的请求（即客户端发起的请求包）

    try
    {
        DatagramPacket SendPacket = new DatagramPacket(bSendBuf,
bSendBuf.length,
                                                       m_TimeService_IPAddress
/*The address to send to*/, NTP_Port);
        m_TimeService_Socket.send(SendPacket);
    }
    catch(SocketTimeoutException Ex)
    {
        return false;
    }
    catch(Exception Ex)
    {
        System.out.printf("Send failed: %s\n", Ex.toString());
        return false;
```

```
    }
    return true;
}
```

6. 从 NTP 服务器接收响应数据包，解析时间戳：`Receive(NTP_Timestamp_Data NTP_Timestamp)` 方法

```
private NTP_Timestamp_Data Receive(NTP_Timestamp_Data NTP_Timestamp){
    byte[] bRecvBuf = new byte [NTP_PACKET_SIZE];
    DatagramPacket RecvPacket = new DatagramPacket(bRecvBuf, NTP_PACKET_SIZE);
    try
    {
        m_TimeService_Socket.receive(RecvPacket);
    }
    catch(Exception ex)
    {
        NTP_Timestamp.lUnixTime = 0;
        return NTP_Timestamp;
    }

    if (0 < RecvPacket.getLength())
    {   // NTP 响应中时间戳存储在数据包的第 40 到 43 字节
        long l1 = (long) bRecvBuf[40] & 0xFF;
        long l2 = (long) bRecvBuf[41] & 0xFF;
        long l3 = (long) bRecvBuf[42] & 0xFF;
        long l4 = (long) bRecvBuf[43] & 0xFF;
        long secsSince1900 = (l1 << 24) + (l2 << 16) + (l3 << 8) + l4;
        NTP_Timestamp.lUnixTime = secsSince1900 - SeventyYears;
        NTP_Timestamp.lHour = (long) (NTP_Timestamp.lUnixTime  % 86400L) / 3600;
        NTP_Timestamp.lMinute = (long) (NTP_Timestamp.lUnixTime % 3600) / 60;
        NTP_Timestamp.lSecond = (long) NTP_Timestamp.lUnixTime % 60;
    }
    else
    {
        NTP_Timestamp.lUnixTime = 0;
    }
    return NTP_Timestamp;
}
```

7. 获取和设置`m_bNTP_Client_Started`标志

```
public Boolean Get_ClientStarted_Flag(){
    return m_bNTP_Client_Started;
}

public void Set_ClientStarted_Flag(Boolean bClient_Started)
{
    m_bNTP_Client_Started = bClient_Started;
}
```

8. 关闭`DatagramSocket`连接：`CloseSocket()`方法

```
public void CloseSocket(){
    try
    {
        m_TimeService_Socket.close();
    }
    catch (Exception Ex)
    {
    }
}
```

2.2.2 NTP客户端处理逻辑

2.2.2.1 成员变量定义

```
NTP_Client m_NTP_Client;      // 存储NTP客户端的实例
int m_iNumRequestsSent;     // 发送的NTP请求数量
int m_iNumResponsesReceived;     // 接收到的NTP响应数量
Timer m_Timer_SendNTPRequests;     // 定时发送NTP请求
DefaultListModel m_listModel_NTPServerList;      // 管理NTP服务器地址的列表
DefaultListModel m_listModel_LocationList;      // 管理NTP服务器地址对应的地理位置列表
ListSelectionListener m_SelectionListener_NTPServerURLs;     // 监听 JList 中服务器地址列表项的选择变化
```

2.2.2.2 初始化

1. 构造方法`TimeServiceClient_GUI_uses_library()`:

```
public TimeServiceClient_GUI_uses_library(){
    // 1. 设置NTP服务器列表选择事件监听器：监听用户在 jList_NTPServerURLs 组件（显示NTP服务器地址列表）中选择的项目
    m_SelectionListener_NTPServerURLs = new ListSelectionListener()
    {
        public void valueChanged(ListSelectionEvent listSelectionEvent)
        {
            int iSelectionIndex = jList_NTPServerURLs.getSelectedIndex();
            jList_NTPServerLocations.setSelectedIndex(iSelectionIndex);
            Get_ServerURL_listBox_Selection();
            jTextField_UNIX_Time.setText("");
            jTextField_UTC_Time.setText("");
            m_iNumRequestsSent = 0;
            m_iNumResponsesReceived = 0;
            UpdateStatisticsDisplay();
        }
    };
    // 2. 初始化NTP服务器列表模型
    m_listModel_NTPServerList = new DefaultListModel(); // NTP服务器地址列表
    m_listModel_LocationList = new DefaultListModel(); // NTP服务器的地理位置列表

    // 3. 初始化组件，填充NTP服务器列表
    initComponents();
    Populate_NTP_Server_List();

    // 4. 创建NTP客户端，打开Socket连接
    m_NTP_Client = new NTP_Client();
    Boolean bSocketOpenSuccess = m_NTP_Client.CreateSocket();
    if (false == bSocketOpenSuccess)
```

```
    {
        JOptionPane.showMessageDialog(null, "Error creating socket", "NTP
client", JOptionPane.PLAIN_MESSAGE);
        CloseSocketAndExit();
    }

    // 5. 初始化请求和响应计数器，更新统计信息
    m_iNumRequestsSent = 0;
    m_iNumResponsesReceived = 0;
    UpdateStatisticsDisplay();

    // 6. 初始化控制项
    InitialiseControls();
}
```

2. 初始化控件：`InitialiseControls()`方法

- 初始化GUI中各个控件的状态（启用或禁用）。

```
private void InitialiseControls()    {
    // 1. 禁用一些组件，包括：显示服务器地址、端口、IP地址、时间信息和请求/响应计数器的文本框
    jPanel_NTPServerAddressDetails.setEnabled(false);
    jTextField_URL.setEnabled(false);
    jTextField_Port.setEnabled(false);
    jTextField_ServerIPAddress.setEnabled(false);
    jTextField_UNIX_Time.setEnabled(false);
    jTextField_UTC_Time.setEnabled(false);
    jTextField_NumRequestsSent.setEnabled(false);
    jTextField_NumResponsesReceived.setEnabled(false);

    // 2. 启用与NTP服务器相关的控件：
    // 启用 jList_NTPServerURLs 组件（NTP服务器地址列表）和其滚动面板，允许用户选择NTP服务
器
    jList_NTPServerURLs.setEnabled(true);
    JScrollPane_NTPServerURLs.setEnabled(true);

    // 3. 禁用与服务器位置相关的控件：
    // 禁用 jList_NTPServerLocations（服务器地理位置列表）和其滚动面板
    jList_NTPServerLocations.setEnabled(false);
    JScrollPane_NTPServerLocations.setEnabled(false);

    // 4. 启用“开始”按钮和“完成”按钮，允许用户启动NTP客户端和结束操作
    jButton_StartNTPClient.setEnabled(true);
    jButton_Done.setEnabled(true);

    // 5. 初始化服务器URL列表框：默认选择列表框中的第一个项
    Initialise_ServerURL_listBox(); // Selects first item in list boxes, by
default
}
```

3. 初始化列表框：`Initialise_ServerURL_listBox()`方法

- 初始化NTP服务器地址列表框（`jList_NTPServerURLs`）和地理位置列表框（`jList_NTPServerLocations`）的选中项，并调用`Get_ServerURL_listBox_Selection()`方法处理选中的服务器地址。

```
private void Initialise_ServerURL_listBox(){
    // 1. 设置默认选中的服务器地址和位置：第一个项
    jList_NTPServerURLs.setSelectedIndex(0);
    jList_NTPServerLocations.setSelectedIndex(0);
    // 2. 处理选中的服务器地址
    Get_ServerURL_listBox_Selection();
}
```

2.2.3 发送和处理NTP请求

1. 定时发送NTP请求：`Start_Timer_SendNTPRequests()`方法

- 启动一个定时任务，用于定时发送NTP请求。

```
void Start_Timer_SendNTPRequests(){
    m_Timer_SendNTPRequests = new Timer();      // 创建定时器
    m_Timer_SendNTPRequests.scheduleAtFixedRate(
        new Get_NTP_Timestamp(), 100, 10000);
    // 1. 第一个参数new Get_NTP_Timestamp()： 定时任务，即每次定时触发时要执行的操作
    // 2. 第二个参数 100 ：第一次任务延迟100毫秒后执行
    // 3. 第三个参数 10000： 之后每10秒（10000毫秒）执行一次该任务
}
```

定时器会在启动后的100毫秒发送第一次NTP请求，然后每10秒发送一次NTP请求。

其中包含`Get_NTP_Timestamp`类：继承自`TimerTask`，负责获取NTP时间戳数据并根据不同的结果更新GUI显示：

```
class Get_NTP_Timestamp extends TimerTask{
    public void run()
    {
        // 1. 获取NTP时间戳
        NTP_Client.NTP_Timestamp_Data NTP_Timestamp =
m_NTP_Client.Get_NTP_Timestamp();

        // 2. 根据不同的返回结果处理
        switch (NTP_Timestamp.eResultCode)
        {
                // 成功获取时间戳：更新请求发送和响应接收计数；将UNIX时间和UTC时间显示在GUI
上
            case NTP_Success:
                m_iNumRequestsSent++;
                m_iNumResponsesReceived++;

 jTextField_UNIX_Time.setText(Long.toString(NTP_Timestamp.lUnixTime));
                String sUTC_Time = String.format("%02d:%02d:%02d",
NTP_Timestamp.lHour, NTP_Timestamp.lMinute, NTP_Timestamp.lSecond);
                jTextField_UTC_Time.setText(sUTC_Time);
                UpdateStatisticsDisplay();
                break;
                // 服务器地址未设置：不操作
            case NTP_ServerAddressNotSet:
                break;
                // 发送请求失败：不操作
            case NTP_SendFailed:
                break;
```

```
                // 接收响应失败：更新请求发送计数，更新显示的统计信息
            case NTP_ReceiveFailed:
                m_iNumRequestsSent++;
                UpdateStatisticsDisplay();
                break;
        }
    }
}
```

2. 停止定时发送NTP请求：`StopTimer()`方法

```
void StopTimer(){
    if (null != m_Timer_SendNTPRequests)
    {
        m_Timer_SendNTPRequests.cancel();
    }
}
```

3. 关闭连接，退出程序：`CloseSocketAndExit()`方法

```
void CloseSocketAndExit(){
    StopTimer();     //1. 停止定时器
    m_NTP_Client.CloseSocket();     //2. 关闭Socket连接
    System.exit(0);    // 3. 退出程序
}
```

2.2.4 处理NTP服务器地址

1. 填充NTP服务器列表及其对应的地理位置：`Populate_NTP_Server_List()`方法

```
void Populate_NTP_Server_List(){
    m_listModel_NTPServerList.addElement("time.nist.gov");
    m_listModel_LocationList.addElement("NIST round robin load equalisation");
    m_listModel_NTPServerList.addElement("time.windows.com");
    m_listModel_LocationList.addElement("Windows Time service");
    m_listModel_NTPServerList.addElement("nist1-atl.ustiming.org");
    m_listModel_LocationList.addElement("Atlanta, Georgia");
    m_listModel_NTPServerList.addElement("wolfnisttime.com");
    m_listModel_LocationList.addElement("Birmingham, Alabama");
    m_listModel_NTPServerList.addElement("nist1-chi.ustiming.org");
    m_listModel_LocationList.addElement("Chicago, Illinois");
    m_listModel_NTPServerList.addElement("nist1-lnk.binary.net");
    m_listModel_LocationList.addElement("Lincoln, Nebraska");
    m_listModel_NTPServerList.addElement("time-a.timefreq.bldrdoc.gov");
    m_listModel_LocationList.addElement("NIST, Boulder, Colorado");
    m_listModel_NTPServerList.addElement("ntp-nist.ldsbc.edu");
    m_listModel_LocationList.addElement("LDSBC, Salt Lake City, Utah");
    m_listModel_NTPServerList.addElement("nist1-lv.ustiming.org");
    m_listModel_LocationList.addElement("Las Vegas, Nevada");
    m_listModel_NTPServerList.addElement("nist1-la.ustiming.org");
    m_listModel_LocationList.addElement("Los Angeles, California");
    m_listModel_NTPServerList.addElement("nist1-ny.ustiming.org");
    m_listModel_LocationList.addElement("New York City, NY");
    m_listModel_NTPServerList.addElement("nist1-nj.ustiming.org");
    m_listModel_LocationList.addElement("Bridgewater, NJ");
}
```

2. 据选中的服务器地址，设置IP和端口：方法`Get_ServerURL_listBox_Selection()`

```
private void Get_ServerURL_listBox_Selection(){
    // 1. 获取选中的服务器地址：获取 jList_NTPServerURLs 列表框中当前选中的值
    String sSelectedURL = jList_NTPServerURLs.getSelectedValue().toString();

    // 2. 显示选中的URL
    jTextField_URL.setText(sSelectedURL);

    // 3.设置IP和端口：调用 SetUp_TimeService_AddressStruct() 方法
    SetUp_TimeService_AddressStruct(sSelectedURL);
}


void SetUp_TimeService_AddressStruct(String sURL)
{
    // 1. 获取NTP服务器的IP地址
    InetAddress TimeService_IPAddress =
m_NTP_Client.SetUp_TimeService_AddressStruct(sURL);

    // 2. 检查IP地址是否有效
    if(null != TimeService_IPAddress)
    {
        // 3（1）. 若IP有效，显示IP地址和端口

 jTextField_ServerIPAddress.setText(TimeService_IPAddress.getHostAddress());
        jTextField_Port.setText(Integer.toString(m_NTP_Client.GetPort()));
    }
    else
    {
        // 3（2）. 若IP无效，在文本框中显示 "Not found"，并清空端口框
        jTextField_ServerIPAddress.setText("Not found");
        jTextField_Port.setText("");
    }
}
```

2.2.3 GUI界面设计

2.2.3.1 初始化组件、设置布局：`initComponents()`方法

1. 组件的创建和布局

标签和文本框：

- 创建了多个 `JLabel` 和 `JTextField`，用于显示和输入不同的信息，如 URL、端口、IP 地址、UNIX 时间、UTC 时间、请求发送次数和响应接收次数等；每个标签都有相应的文本框，用于显示信息或输入数据。

2. 面板布局

NTP 服务器地址面板（`jPanel_NTPServerAddressDetails`）：

- 显示与 NTP 服务器相关的信息，包括 URL、端口和 IP 地址。使用了 `GroupLayout` 布局管理器对组件进行了排布。

时间和状态面板（`jPanel_Time_Status`）：

- 显示 NTP 客户端的状态信息，包括 UNIX 时间、UTC 时间、请求发送次数和响应接收次数。也使用了 `GroupLayout` 布局进行排列。

NTP 服务器选择面板（`jPanel_NTPServerSelection`）：

- 显示 NTP 服务器 URL 列表和对应的服务器位置或描述。`JList` 组件用于显示服务器 URL 和描述，并通过 `JScrollPane` 添加滚动条功能。

控制面板（`jPanel_Controls`）：

- 包含两个按钮：一个用于启动/停止 NTP 请求（`jButton_StartNTPClient`），另一个用于退出程序（`jButton_Done`）。按钮点击事件由 `actionPerformed` 方法处理。

3. 布局管理

使用了 `GroupLayout` 来管理不同面板中的组件布局。`GroupLayout` 是一种灵活的布局管理器，它允许开发者定义组件的水平和垂直排列方式，同时还支持容器内组件的对齐和间距。

- 组件按照特定的顺序和排列方式被放置在各个面板中，最终将面板添加到主窗口（`getContentPane()`）中。
- `setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE)` 确保关闭窗口时退出应用程序。

```
private void initComponents(){
    jLabel_URL = new JLabel();
    jLabel_URL.setText("URL");
    jTextField_URL = new JTextField();
    jTextField_URL.setMaximumSize(new Dimension(250, 4));
    jTextField_URL.setHorizontalAlignment(JTextField.CENTER);
    jLabel_Port = new JLabel();
    jLabel_Port.setText("Port");
    jTextField_Port = new JTextField();
    jTextField_Port.setMaximumSize(new Dimension(120, 4));
    jTextField_Port.setHorizontalAlignment(JTextField.CENTER);
    jLabel_ServerIPAddress = new JLabel();
    jLabel_ServerIPAddress.setText("Server IP address");
    jTextField_ServerIPAddress = new JTextField();
    jTextField_ServerIPAddress.setMaximumSize(new Dimension(120, 4));
    jTextField_ServerIPAddress.setHorizontalAlignment(JTextField.CENTER);

    jPanel_NTPServerAddressDetails = new JPanel();
    jPanel_NTPServerAddressDetails.setPreferredSize(new Dimension(400, 300));

 jPanel_NTPServerAddressDetails.setBorder(BorderFactory.createTitledBorder("Sele
cted NTP Server Address"));
    org.jdesktop.layout.GroupLayout jPanel1Layout = new
org.jdesktop.layout.GroupLayout(jPanel_NTPServerAddressDetails);
    jPanel_NTPServerAddressDetails.setLayout(jPanel1Layout);
    jPanel1Layout.setHorizontalGroup(

 jPanel1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.CENTER)
        .add(jPanel1Layout.createSequentialGroup()
             .add(jLabel_URL)
             .addContainerGap(10, 10)
             .add(jTextField_URL))
        .add(jPanel1Layout.createSequentialGroup()
```

```
                .add(jLabel_Port)
                .addContainerGap(10, 10)
                .add(jTextField_Port))
            .add(jPanel1Layout.createSequentialGroup()
                .add(jLabel_ServerIPAddress)
                .addContainerGap(10, 10)
                .add(jTextField_ServerIPAddress)));
        jPanel1Layout.setVerticalGroup(

    jPanel1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.CENTER)
            .add(jPanel1Layout.createSequentialGroup()
                .add(jPanel1Layout.createParallelGroup()
                    .add(jLabel_URL)
                    .add(jTextField_URL))
                .addContainerGap(20, 20)
                .add(jPanel1Layout.createParallelGroup()
                    .add(jLabel_Port)
                    .add(jTextField_Port))
                .addContainerGap(20, 20)
                .add(jPanel1Layout.createParallelGroup()
                    .add(jLabel_ServerIPAddress)
                    .add(jTextField_ServerIPAddress))));

        jLabel_UNIX_Time = new JLabel();
        jLabel_UNIX_Time.setText("UNIX time");
        jTextField_UNIX_Time = new JTextField();
        jTextField_UNIX_Time.setMaximumSize(new Dimension(120, 4));
        jTextField_UNIX_Time.setHorizontalAlignment(JTextField.CENTER);
        jLabel_UTC_Time = new JLabel();
        jLabel_UTC_Time.setText("UTC time");
        jTextField_UTC_Time = new JTextField();
        jTextField_UTC_Time.setMaximumSize(new Dimension(120, 4));
        jTextField_UTC_Time.setHorizontalAlignment(JTextField.CENTER);
        jLabel_NumRequestsSent = new JLabel();
        jLabel_NumRequestsSent.setText("Number of NTP time requests sent");
        jTextField_NumRequestsSent = new JTextField();
        jTextField_NumRequestsSent.setMaximumSize(new Dimension(60, 4));
        jTextField_NumRequestsSent.setHorizontalAlignment(JTextField.CENTER);
        jLabel_NumResponsesReceived = new JLabel();
        jLabel_NumResponsesReceived.setText("Number of NTP time responses
received");
        jTextField_NumResponsesReceived = new JTextField();
        jTextField_NumResponsesReceived.setMaximumSize(new Dimension(60, 4));
        jTextField_NumResponsesReceived.setHorizontalAlignment(JTextField.CENTER);

        jPanel_Time_Status = new JPanel();
        jPanel_Time_Status.setPreferredSize(new Dimension(400, 300));
        jPanel_Time_Status.setBorder(BorderFactory.createTitledBorder("Time and
Status"));
        org.jdesktop.layout.GroupLayout jPanel2Layout = new
org.jdesktop.layout.GroupLayout(jPanel_Time_Status);
        jPanel_Time_Status.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(

    jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.CENTER)
            .add(jPanel2Layout.createSequentialGroup()
                .add(jLabel_UNIX_Time)
                .addContainerGap(10, 10)
```

```
                .add(jTextField_UNIX_Time))
            .add(jPanel2Layout.createSequentialGroup()
                .add(jLabel_UTC_Time)
                .addContainerGap(10, 10)
                .add(jTextField_UTC_Time))
            .add(jLabel_NumRequestsSent)
            .add(jTextField_NumRequestsSent)
            .add(jLabel_NumResponsesReceived)
            .add(jTextField_NumResponsesReceived));
        jPanel2Layout.setVerticalGroup(

     jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.CENTER)
            .add(jPanel2Layout.createSequentialGroup()
                .add(jPanel2Layout.createParallelGroup()
                    .add(jLabel_UNIX_Time)
                    .add(jTextField_UNIX_Time))
                .addContainerGap(20, 20)
                .add(jPanel2Layout.createParallelGroup()
                    .add(jLabel_UTC_Time)
                    .add(jTextField_UTC_Time))
                .addContainerGap(20, 20)
                .add(jLabel_NumRequestsSent)
                .add(jTextField_NumRequestsSent)
                .addContainerGap(20, 20)
                .add(jLabel_NumResponsesReceived)
                .add(jTextField_NumResponsesReceived)));

        jLabel_NIST_Servers = new JLabel();
        jLabel_NIST_Servers.setText("A selection of NIST servers are provided
(availability may change over time)");
        jLabel_NTPServerURLs = new JLabel();
        jLabel_NTPServerURLs.setText("NTP Server URLs");
        jLabel_NTPServerLocations = new JLabel();
        jLabel_NTPServerLocations.setText("Location / description");
        jList_NTPServerURLs = new JList(m_listModel_NTPServerList);
        jList_NTPServerURLs.setMaximumSize(new Dimension(300, 250));
        jList_NTPServerURLs.setSelectedIndex(0);
        jList_NTPServerURLs.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

     jList_NTPServerURLs.addListSelectionListener(m_SelectionListener_NTPServerURLs)
;
        JScrollPane_NTPServerURLs = new javax.swing.JScrollPane(jList_NTPServerURLs,

     ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,

     ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        jList_NTPServerLocations = new JList(m_listModel_LocationList);
        jList_NTPServerLocations.setMaximumSize(new Dimension(300, 250));
        jList_NTPServerLocations.setSelectedIndex(0);

     jList_NTPServerLocations.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        JScrollPane_NTPServerLocations = new
javax.swing.JScrollPane(jList_NTPServerLocations,

ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,

ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
```

```
    jPanel_NTPServerSelection = new JPanel();
    jPanel_NTPServerSelection.setPreferredSize(new Dimension(500, 300));
    jPanel_NTPServerSelection.setBorder(BorderFactory.createTitledBorder("NIST /
NTP Server Selection"));
    org.jdesktop.layout.GroupLayout jPanel3Layout = new
org.jdesktop.layout.GroupLayout(jPanel_NTPServerSelection);
    jPanel_NTPServerSelection.setLayout(jPanel3Layout);
    jPanel3Layout.setHorizontalGroup(

 jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.CENTER)
        .add(jPanel3Layout.createSequentialGroup()
             .add(jLabel_NIST_Servers))
        .add(jPanel3Layout.createSequentialGroup()

.add(jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.CENTER)
                  .add(jLabel_NTPServerURLs)
                  .add(JScrollPane_NTPServerURLs))

.add(jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.CENTER)
                  .add(jLabel_NTPServerLocations)
                  .add(JScrollPane_NTPServerLocations))));
    jPanel3Layout.setVerticalGroup(

 jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.CENTER)
        .add(jPanel3Layout.createSequentialGroup()
             .add(jLabel_NIST_Servers)
             .addContainerGap(20, 20)
             .add(jPanel3Layout.createParallelGroup()
                  .add(jPanel3Layout.createSequentialGroup()
                       .add(jLabel_NTPServerURLs)
                       .add(JScrollPane_NTPServerURLs))
                  .add(jPanel3Layout.createSequentialGroup()
                       .add(jLabel_NTPServerLocations)
                       .add(JScrollPane_NTPServerLocations)))));

    jButton_StartNTPClient = new JButton();
    jButton_StartNTPClient.setText("START NTP requests");
    jButton_StartNTPClient.setMaximumSize(new Dimension(100, 4));
    jButton_StartNTPClient.addActionListener(this);
    jButton_Done = new JButton();
    jButton_Done.setText("Done");
    jButton_Done.setMaximumSize(new Dimension(100, 4));
    jButton_Done.addActionListener(this);

    jPanel_Controls = new JPanel();
    jPanel_Controls.setPreferredSize(new Dimension(500, 100));
    jPanel_Controls.setBorder(BorderFactory.createTitledBorder("Controls"));
    org.jdesktop.layout.GroupLayout jPanel4Layout = new
org.jdesktop.layout.GroupLayout(jPanel_Controls);
    jPanel_Controls.setLayout(jPanel4Layout);
    jPanel4Layout.setHorizontalGroup(

 jPanel4Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.CENTER)
        .add(jPanel4Layout.createSequentialGroup()
             .addContainerGap(200, 200)
             .add(jButton_StartNTPClient)
             .addContainerGap(200, 200)
             .add(jButton_Done)
```

```
                .addContainerGap(200, 200)));
        jPanel4Layout.setVerticalGroup(

     jPanel4Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.CENTER)
            .add(jPanel4Layout.createParallelGroup()
                 .add(jButton_StartNTPClient)
                 .add(jButton_Done)));

        org.jdesktop.layout.GroupLayout layout = new
    org.jdesktop.layout.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        getContentPane().setPreferredSize(new Dimension(700, 450));

        layout.setHorizontalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.CENTER)
            .add(layout.createSequentialGroup()
                 .add(layout.createParallelGroup()
                      .add(jPanel_NTPServerAddressDetails)
                      .add(jPanel_Time_Status))
                 .add(layout.createParallelGroup()
                      .add(jPanel_NTPServerSelection)
                      .add(jPanel_Controls))));
        layout.setVerticalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.CENTER)
            .add(layout.createSequentialGroup()
                 .add(jPanel_NTPServerAddressDetails)
                 .addContainerGap(20, 20)
                 .add(jPanel_Time_Status))
            .add(layout.createSequentialGroup()
                 .add(jPanel_NTPServerSelection)
                 .addContainerGap(20, 20)
                 .add(jPanel_Controls)));

        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setTitle("Network Time Protocol client");
        pack();
    }
```

2.2.3.2 实现交互：`actionPerformed(ActionEvent e)` 方法

- **判断按钮类型**：
  - 如果点击的是 `jButton_Done` 按钮，则调用 `CloseSocketAndExit()` 方法关闭套接字并退出程序。
  - 如果点击的是 `jButton_StartNTPClient` 按钮：
    - 如果客户端未启动（`m_NTP_Client.Get_ClientStarted_Flag()` 返回 `false`），则：
      - 更改按钮文本为 "STOP NTP requests"。
      - 禁用 `jList_NTPServerURLs`，防止在请求开始时修改服务器列表。
      - 重置请求发送次数和响应接收次数，并更新显示的统计信息。
      - 调用 `Start_Timer_SendNTPRequests()` 启动定时器，定时发送 NTP 请求。
      - 设置客户端为已启动状态（通过 `m_NTP_Client.Set_ClientStarted_Flag(true)`）。
    - 如果客户端已经启动，则：
      - 更改按钮文本为 "START NTP requests"。

- 启用 `jList_NTPServerURLs`，允许用户选择服务器。
- 设置客户端为未启动状态，并停止定时器（调用 `StopTimer()`）。

```
public void actionPerformed(ActionEvent e){
    if(jButton_Done == e.getSource())
    {
        CloseSocketAndExit();
    }
    if(jButton_StartNTPClient == e.getSource())
    {
        if(false == m_NTP_Client.Get_ClientStarted_Flag())
        {
            jButton_StartNTPClient.setText("STOP NTP requests");
            jList_NTPServerURLs.setEnabled(false);
            m_iNumRequestsSent = 0;
            m_iNumResponsesReceived = 0;
            UpdateStatisticsDisplay();
            Start_Timer_SendNTPRequests();
            m_NTP_Client.Set_ClientStarted_Flag(true);
        }
        else
        {
            jButton_StartNTPClient.setText("START NTP requests");
            jList_NTPServerURLs.setEnabled(true);
            m_NTP_Client.Set_ClientStarted_Flag(false);
            StopTimer();
        }
    }
}
```

2.2.3.3 其他组件和事件

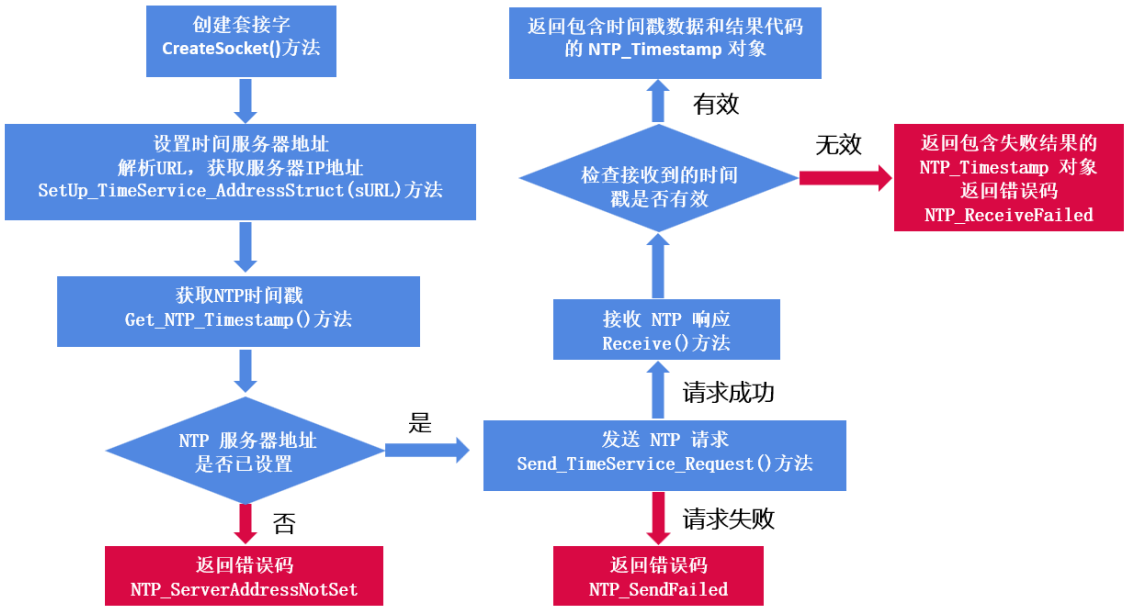
- **按钮（`jButton_StartNTPClient` 和 `jButton_Done`）**：
  - `jButton_StartNTPClient` 按钮用来启动或停止 NTP 客户端的请求，点击时会切换按钮文本并启动/停止定时任务。
  - `jButton_Done` 按钮用来关闭套接字并退出程序。
  - 两个按钮都通过 `addActionListener(this)` 将事件监听器添加到按钮上。
- **`JList` 组件**：
  - `jList_NTPServerURLs` 用于显示 NTP 服务器的 URL 列表，用户可以选择一个服务器进行连接。
  - `jList_NTPServerLocations` 用于显示每个 NTP 服务器的描述信息。

2.4 主程序

- 入口点`main`方法如下：

```
public static void main(String args[]) throws Exception{
    // 1. 设置 Look and Feel
        try
        {
            javax.swing.UIManager.LookAndFeelInfo[]
installedLookAndFeels=javax.swing.UIManager.getInstalledLookAndFeels();
            for (int idx=0; idx<installedLookAndFeels.length; idx++)
            {
                if ("Nimbus".equals(installedLookAndFeels[idx].getName())) {
```

```
 javax.swing.UIManager.setLookAndFeel(installedLookAndFeels[idx].getClassName())
;
                    break;
                }
            }
        }
        catch (Exception Ex)
        {
            System.exit(0);
        }

    // 2. 将启动 GUI 窗口的任务添加到事件队列中；确保该任务在事件调度线程中执行
        java.awt.EventQueue.invokeAndWait(new Runnable() // Create and display
the GUI form
        {
            public void run()
            {
            // 创建 TimeServiceClient_GUI_uses_library 类的一个新实例，并将其显示在屏幕
上
                new TimeServiceClient_GUI_uses_library().setVisible(true);
            }
        } );
    }
```



3 实验总体思路

3.1 改进方案

3.1.1 提升时间精度

原有的代码仅处理了 NTP 数据包中的秒数部分，而分数部分则被用来表示秒的小数部分。为了提升精度，我们进一步处理了分数部分，将其转换为毫秒值。

具体实现中，通过将 NTP 数据包中的分数部分（由 4 个字节组成）解析为一个 `long` 类型值，接着将其转化为秒的小数部分，并乘以 1000 得到毫秒数。通过 `Math.round()` 方法对结果进行四舍五入，将其赋值给 `lMillisecond` 属性。同时，将计算出的毫秒数转换为秒并加回到 `lUnixTime` 中，确保 Unix 时间戳精度达到了毫秒级。

3.1.2 增加向北京时间校准

为了更好的判断同步是否准备，将UTC转换为CST（北京时间）。

3.1.3 减小网络延迟

通过前文所说NTP原理，计算出offset来降低网络延迟。

4 实验过程

4.1 提升时间精度

在原始代码中，UNIX和UTC的精度为秒，如如下代码所示：

```
private NTP_Timestamp_Data Receive(NTP_Timestamp_Data NTP_Timestamp){
    byte[] bRecvBuf = new byte [NTP_PACKET_SIZE];
    DatagramPacket RecvPacket = new DatagramPacket(bRecvBuf, NTP_PACKET_SIZE);
    try{
        m_TimeService_Socket.receive(RecvPacket);
    }
    catch(Exception ex){
        NTP_Timestamp.lUnixTime = 0; // Signal that an error occurred
        return NTP_Timestamp;
    }

    if (0 < RecvPacket.getLength()){
        long l1 = (long) bRecvBuf[40] & 0xFF;
        long l2 = (long) bRecvBuf[41] & 0xFF;
        long l3 = (long) bRecvBuf[42] & 0xFF;
        long l4 = (long) bRecvBuf[43] & 0xFF;
        long secsSince1900 = (l1 << 24) + (l2 << 16) + (l3 << 8) + l4;
        NTP_Timestamp.lUnixTime = secsSince1900 - SeventyYears;
        NTP_Timestamp.lHour = (long) (NTP_Timestamp.lUnixTime  % 86400L) / 3600;
        NTP_Timestamp.lMinute = (long) (NTP_Timestamp.lUnixTime % 3600) / 60;
        NTP_Timestamp.lSecond = (long) NTP_Timestamp.lUnixTime % 60 + 100*frac;
    }
    else
    {
        NTP_Timestamp.lUnixTime = 0; // Signal that an error occurred
    }
    return NTP_Timestamp;
}
```

原有的代码仅处理了 NTP 数据包中的秒数部分，而分数部分则被用来表示秒的小数部分。为了提升精度，我们进一步处理了分数部分，将其转换为毫秒值。

具体实现中，通过将 NTP 数据包中的分数部分（由 4 个字节组成）解析为一个 `long` 类型值，接着将其转化为秒的小数部分，并乘以 1000 得到毫秒数。通过 `Math.round()` 方法对结果进行四舍五入，将其赋值给 `lMillisecond` 属性。同时，将计算出的毫秒数转换为秒并加回到 `lUnixTime` 中，确保 Unix 时间戳精度达到了毫秒级。

这一改进显著提高了时间同步的精度，使得从 NTP 服务器获取的时间可以精确到毫秒，为精确的时间校准和同步提供了更可靠的数据支持。改进后的代码如下：

```
private NTP_Timestamp_Data Receive(NTP_Timestamp_Data NTP_Timestamp) {
    byte[] bRecvBuf = new byte [NTP_PACKET_SIZE];
```

```
    DatagramPacket RecvPacket = new DatagramPacket(bRecvBuf, NTP_PACKET_SIZE);
    try {
        m_TimeService_Socket.receive(RecvPacket);
    }
    catch(Exception ex) {
        NTP_Timestamp.lUnixTime = 0;
        return NTP_Timestamp;
    }
    if (0 < RecvPacket.getLength()) {
        long l1 = (long) bRecvBuf[40] & 0xFF;
        long l2 = (long) bRecvBuf[41] & 0xFF;
        long l3 = (long) bRecvBuf[42] & 0xFF;
        long l4 = (long) bRecvBuf[43] & 0xFF;
        long secsSince1900 = (l1 << 24) + (l2 << 16) + (l3 << 8) + l4;
        NTP_Timestamp.lUnixTime = secsSince1900 - SeventyYears;
        long fraction = ((long) bRecvBuf[44] & 0xFF) << 24 |
            ((long) bRecvBuf[45] & 0xFF) << 16 |
            ((long) bRecvBuf[46] & 0xFF) << 8 |
            ((long) bRecvBuf[47] & 0xFF);

        double milliseconds = (fraction / (double) (1L << 32)) * 1000.0;
        NTP_Timestamp.lMillisecond = Math.round(milliseconds);
        NTP_Timestamp.lUnixTime+= (double) NTP_Timestamp.lMillisecond /1000;

        NTP_Timestamp.lHour = (long) (NTP_Timestamp.lUnixTime  % 86400L) / 3600;
        NTP_Timestamp.lMinute = (long) (NTP_Timestamp.lUnixTime % 3600) / 60;
        NTP_Timestamp.lSecond = (long) NTP_Timestamp.lUnixTime % 60;
    } else {
        NTP_Timestamp.lUnixTime = 0;
    }
    return NTP_Timestamp;
}
```

4.2 增加向北京时间校准

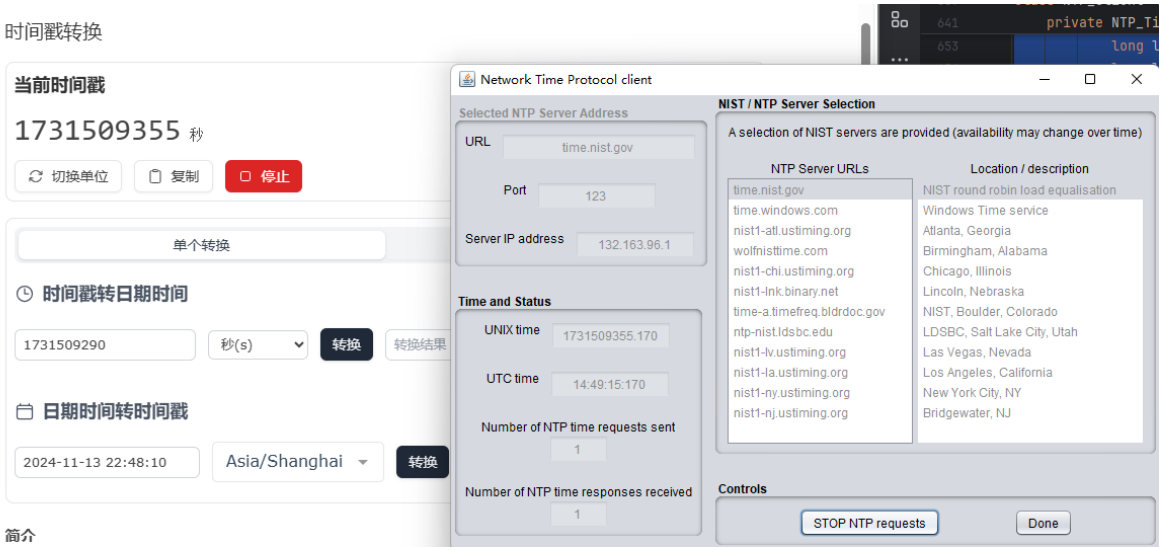
```
private NTP_Timestamp_Data Receive(NTP_Timestamp_Data NTP_Timestamp) {
    byte[] bRecvBuf = new byte[NTP_PACKET_SIZE];
    DatagramPacket RecvPacket = new DatagramPacket(bRecvBuf, NTP_PACKET_SIZE);
    try {
        m_TimeService_Socket.receive(RecvPacket);
    } catch (Exception ex) {
        NTP_Timestamp.lUnixTime = 0;
        return NTP_Timestamp;
    }

    if (0 < RecvPacket.getLength()) {
        long l1 = (long) bRecvBuf[40] & 0xFF;
        long l2 = (long) bRecvBuf[41] & 0xFF;
        long l3 = (long) bRecvBuf[42] & 0xFF;
        long l4 = (long) bRecvBuf[43] & 0xFF;
        long secsSince1900 = (l1 << 24) + (l2 << 16) + (l3 << 8) + l4;
        NTP_Timestamp.lUnixTime = secsSince1900 - SeventyYears;

        long fraction = ((long) bRecvBuf[44] & 0xFF) << 24 |
            ((long) bRecvBuf[45] & 0xFF) << 16 |
            ((long) bRecvBuf[46] & 0xFF) << 8 |
            ((long) bRecvBuf[47] & 0xFF);
```

```
        double milliseconds = (fraction / (double) (1L << 32)) * 1000.0;
        NTP_Timestamp.lMillisecond = Math.round(milliseconds);
        NTP_Timestamp.lUnixTime += (double) NTP_Timestamp.lMillisecond / 1000;

        NTP_Timestamp.lHour = (long) (NTP_Timestamp.lUnixTime % 86400L) / 3600;
        NTP_Timestamp.lMinute = (long) (NTP_Timestamp.lUnixTime % 3600) / 60;
        NTP_Timestamp.lSecond = (long) NTP_Timestamp.lUnixTime % 60;

        NTP_Timestamp.lHour += 8;
        if (NTP_Timestamp.lHour >= 24) {
            NTP_Timestamp.lHour -= 24;
        }
    } else {
        NTP_Timestamp.lUnixTime = 0;
    }

    return NTP_Timestamp;
}
```

5 实验结果

5.1 提升时间精度

如下图所示，实验精度提升到了毫秒级。



5.2 增加向北京时间校准

5.3 减小网络延迟

向NTP服务器发送时记录T1：

```
Boolean Send_TimeService_Request()  1 usage
{
    byte[] bSendBuf = new byte [NTP_PACKET_SIZE]; // Zero-out entire 48-byte buffer
    // Initialize values needed to form NTP request
    bSendBuf[0] = (byte) 0xE3;// 0b11100011;
    // LI bits 7,6      = 3 (Clock not synchronised),
    // Version bits 5,4,3   = 4 (The current version of NTP)
    // Mode bits 2,1,0  = 3 (Sent by client)

    try
    {
        DatagramPacket SendPacket = new DatagramPacket(bSendBuf, bSendBuf.length,
                m_TimeService_IPAddress /*The address to send to*/, NTP_Port);
        m_TimeService_Socket.send(SendPacket);
        sendTime= (double) System.currentTimeMillis() /1000;
    }
    catch(SocketTimeoutException Ex)
    {
        return false;
    }
    catch(Exception Ex)
    {
        System.out.printf("Send failed: %s\n", Ex.toString());
        return false;
    }
    return true;
}
```

客户端接收时间T4：

```
private NTP_Timestamp_Data Receive(NTP_Timestamp_Data NTP_Timestamp) {  1 usage
    byte[] bRecvBuf = new byte[NTP_PACKET_SIZE];
    DatagramPacket RecvPacket = new DatagramPacket(bRecvBuf, NTP_PACKET_SIZE);
    try {
        m_TimeService_Socket.receive(RecvPacket);
        receiveTime= (double) System.currentTimeMillis() /1000;
    } catch (Exception ex) {
        NTP_Timestamp.lUnixTime = 0;
        return NTP_Timestamp;
    }
```

由于T3-T2<<offset，所以UnixTime直接加上offset：

```
    long secsSince1900 = (l1 << 24) + (l2 << 16) + (l3 << 8) + l4;
    NTP_Timestamp.lUnixTime = secsSince1900 - SeventyYears;
    NTP_Timestamp.lUnixTime+=(receiveTime-sendTime)/2;
```

5.4 调研北斗三号 PPP-B2b 全球时间同步方案

PPP-B2b 时间同步方法将 PPP-B2b 钟差解算与时钟控制相结合，实时调节接收机外接 10 MHz铷钟频率，并驯服输出同步到时间基准的 1PPS 信号。具体原理如下图所示。

5.3.1 PPP观测值

PPP-B2b时间同步方法使用PPP算法进行时间传递。在本文中，PPP使用双频无电离层组合模型消除电离层低阶延迟项，并使用伪距和载波相位观测值进行未知参数估计。双频无电离层组合的伪距IF_P和载波相位IF_L的观测方程可表示为：

$$IF_P = \rho + c(\delta t_r - \delta t_s) + d_r + d_s + M_w Z_w + \lambda_I F N_{IF} + \epsilon_{IF}$$
$$\lambda_I F IF_L = \rho + c(\delta t_r - \delta t_s) + D_r + D_s + M_w Z_w + \lambda_I F N_{IF} + e_{IF}$$

其中：

- ρ为接收机天线相位中心与卫星天线相位中心之间的几何距离。
- c为光速。
- δt_r 和δt_s分别为接收机和卫星时钟相对于对应卫星系统的时间偏差。
- d_r和 d_s分别为双频无电离层组合的接收机伪距硬件延迟和相应卫星的伪距硬件延迟。
- D_r和 D_s分别为双频无电离层组合的接收机相位硬件延迟和相应卫星的相位硬件延迟。
- M_w为湿投影函数。
- Z_w为测站天顶方向对流层延迟湿分量。
- λ_{IF}为双频无电离层组合波长。
- N_{IF}为双频无电离层组合整周模糊度。
- ϵ_{IF}和 e_{IF} 为双频无电离层伪距和载波相位的误差噪声。

5.3.2 B2b精密轨道与钟差恢复

通过使用PPP-B2b轨道改正参数结合广播星历计算出的位置向量，可以计算出校正后的卫星精密位置：

$$X_O = X_{brdc} + \delta X$$

其中：

- X_O为由轨道改正参数校正后的卫星精密位置。
- X_{brdc}为广播星历计算得到的卫星位置。
- δX为通过B2b轨道改正参数计算出的卫星位置修正矢量。

卫星钟差改正数用于校正广播星历计算出的卫星钟差：

$$\delta t_s = t_{brdc} + \Delta t$$

其中：

- δt_s为校正后的卫星精密钟差。
- t_{brdc}为通过广播星历计算出的卫星钟差。
- Δt为PPP-B2b信息类型4中获得的钟差改正数。

5.3.3 PPP-B2b时间同步方法

PPP-B2b时间同步方法将PPP-B2b钟差解算与时钟控制相结合，实时调节接收机外接10 MHz铷钟频率，并驯服输出同步到时间基准的1PPS信号。具体原理如下：

1. GNSS接收机输出原始卫星观测值、广播星历、PPP-B2b改正数。
2. 将广播星历与PPP-B2b轨道、钟差改正数结合恢复出B2b精密星历。
3. 将恢复出的卫星精密位置、接收机观测值输入PPP-B2b解算模块。
4. PPP-B2b解算模块进行数据预处理，剔除高度角和信噪比不符合要求的卫星。
5. 通过接收机实时观测数据和B2b精密星历解算PPP观测方程，输出接收机实时钟差。
6. GNSS接收机将自身晶振锁定在外置10 MHz铷钟上，与铷钟具有相同的频率特性和固定的相位差。
7. 通过测量10 MHz铷钟输出1PPS与接收机输出1PPS的相位差，结合接收机钟差，生成10 MHz铷钟的频率控制量和1PPS相位调整量，调节铷钟频率和输出1PPS相位。

通过实验验证，该方法在短基线实测环境的时间同步精度优于0.95 ns，模拟器仿短基线时间同步精度优于0.87 ns，模拟器仿长基线时间同步精度优于0.9 ns，显示出良好的时间同步性能和长基线时间同步精度不变的特点。这种方法不仅提高了时间同步的精度，而且由于不依赖互联网，具有更强的单机作业能力，适用于分布式雷达、无人机协同作战等不能连接互联网的应用场景。

6 效果和问题分析

在实现基于 NTP 协议的时间同步实验中，成功地将客户端时间与不同 NTP 服务器的时间进行校准，达到了毫秒级的同步精度。实验过程中，通过解析 NTP 数据包并考虑网络延迟对时间进行校正，能够有效解决时钟同步问题。然而，在实践中也遇到了一些挑战，比如网络延迟的变化和时钟漂移对同步精度的影响。虽然通过简单的时间戳获取能够实现初步的时间同步，但对于高精度的需求，单纯的时间戳获取无法满足要求。实际开发中，还需要考虑服务器故障和网络异常的情况，因此实现重试机制、容错处理以及多服务器加权平均等措施，能够进一步提升时间同步的可靠性和精度。总的来说，虽然时间同步在实验中得到了有效实现，但如何处理异常情况、提高同步精度以及容错能力，仍然是未来完善该系统时需要重点考虑的因素。

7 体会和建议

通过实现基于 Java 的 NTP 协议来向不同的 NTP 服务器进行时间校准，体会到了时间同步在分布式系统中的重要性和复杂性。首先，理解 NTP 协议本身是一个必要的基础，它不仅涉及到精确的时间计算，还需要考虑网络延迟、时钟漂移等多种因素。在实验过程中，通过解析 NTP 数据包并根据网络延迟进行校正，使得客户端能够准确同步服务器时间，精度达到了毫秒级。这一过程增强了对 NTP 协议的理解，也更好地掌握了如何在网络通信中处理时间同步的问题。

在实际开发中，除了考虑基本的时间同步，还需要考虑容错机制，例如服务器故障、时钟漂移等因素。因此，在实际使用中，建议客户端程序不仅要能从多个服务器获取时间，还要实现重试机制，以确保服务器不可达时能够进行有效的故障恢复。此外，网络延迟补偿和时间精度校正是不可忽视的细节，简单的时间戳获取无法提供足够精度，因此在程序中对时间的精细化处理显得尤为重要。最后，针对不同的 NTP 服务器，可能存在时间误差，采用多个服务器来进行对比和加权平均，能进一步提高时间同步的可靠性。

综上所述，成功实现一个高精度的 NTP 时间同步系统，不仅需要理解协议本身，还需要在实际编程中灵活地处理各种异常情况，并对时间精度进行充分的考虑。