文章编号: 1006-9348(2020)11-0216-06

基于 NTP 的跨平台网络时间同步模块设计

何承恩[1],王 忠[1],岳佳欣[2]

(1. 四川大学电气信息学院,四川 成都 610065;2. 四川电影电视学院,四川 成都,610036)

摘要: 为了解决在操作系统多样、设备部署分散或运行独立性高的环境下设备的时间同步问题,设计一个跨平台的网络时间同步模块。模块基于第 4 版网络时间协议(Network Time Protocol Version 4, NTPv4),采用 C++面向对象的思想,利用条件编译分别在 Windows 和 Linux 系统下将时间同步过程中各个步骤单独封装,构造出 Windows 和 Linux 环境的派生类,最终对外提供时间同步接口函数和库文件,以供跨平台软件开发时调用。实验结果表明,双系统中其它程序调用该库文件和接口函数在广域网中能够实现亚毫秒级的时间同步功能。

关键词: 时间同步模块; 跨平台; 网络时间协议

中图分类号: TP311 **文献标识码:** B

Design of Cross Platform Network Time Synchronization Module Based on NTP

HE Cheng-en[1], WANG Zhong[1], YUE Jia-xin[2]

(1. College of Electrical Engineering and Information Technology, Sichuan University, Chengdu Sichuan 610065, China;
2. Sichuan Film and Television University, Chengdu Sichuan 610036, China)

ABSTRACT: In order to solve the problem of time synchronization in the environment of diverse operating systems, decentralized device deployment or high operational independence, we designed cross-platform network time synchronization module. Based on the Network Time Protocol Version 4 (NTPv4), this module used the object-oriented idea of C++. Using conditional compilation, each step of time synchronization process was encapsulated separately in Windows and Linux systems, and then the derived classes of Windows and Linux were constructed. Finally, a time synchronization interface function and library file were provided for cross-platform software development. The experimental results show that other programs in the dual system can call the library file and interface function to achieve sub-millisecond time synchronization in WAN.

KEYWORDS: Time synchronization module; Cross platform; NTP

1 引言

在交通、金融、电信等领域,为各个设备配置统一时间对整个系统运行具有重要意义。1972 年诞生的协调世界时(Coordinated Universal Time, UTC)是全球民用标准频率和时间信号发波的基础[1]。目前常用的授时发播系统有星基授时系统,例如 GPS、北斗导航卫星[2];陆基无线电授时系统,例如长波授时系统、短波授时系统及低频时码授时系统;有线授时系统,例如 NTP 互联网授时系统和 PSTN 公网电话授时系统[3]。

传统的对时方法或是组建专用网络为各系统提供时间基准,或是购买专门的时间接收装置,或是采用人工方式进行时间调整,存在效率低、成本高、误差大的问题。且分布式系统无法为彼此间相互独立的模块提供一个统一的全局时钟,通常由各个进程或模块维护各自的本地时钟。目前最常用的两个时间同步技术是 GPS 全球定位系统和网络时间协议 NTP[4]。网络授时是以互联网为载体,为互联网中其它终端设备提供标准时间信息的方式[5]。常见的网络授时协议有 Daytime 协议、Time 协议、NTP 协议和 PTP(Precise Time Protocol)协议,以及发展中的 White Rabbit(WR)和 SAE AS6802 协议[6]。前两个协议相对简单,精度只能达到秒级,已停止使用。PTP 协议用所定义的边界时钟和透明时钟来代替网络中的普通交换设备,去除了报文传输过程中的不确定性因素带来的抖动影响,具有亚微秒级时间同步功能[7]。主要应用于网络环境与拓扑结构比较简单的网络,需要专用的硬件[8]。WR 技术应用于精度要求极高的大型物理实验

基金项目: 四川省科技厅科技支撑项目(2015FZ061)
收稿日期: 2019-03-06 修回日期: 2019-06-03

装置和宇宙射线观测装置中[9]，同步精度能达到皮秒级别，但需要利用光纤链路单独构建网络，成本高昂。SAE AS6802协议主要用于航空电子系统内各个电子模块之间的时间同步，同步精度在微秒级别，网络规模很小。

NTP是目前应用最普遍的时间同步协议，只要支持网络套接字的计算机系统均能使用NTP进行时间同步[10]。其高效性、便利性和精准性使其成为互联网设备广泛使用的时间同步方式[11]。但NTP的实现方式因系统而异，导致在一个跨平台软件开发中需要根据不同的系统分别实现NTP功能。故面向最常用的Windows和Linux系统，使用C++语言设计一个跨平台网络时间同步模块，使用者只需调用生成的静态库和动态库就可以实现Windows和Linux系统下的时间同步功能，提高了跨平台软件的开发效率和多系统分布式设备保持时间同步的便利性。

2 NTP简介

NTP协议通常描述为一种客户端-服务器模式，发送和接收时间戳使用互联网协议(Internet Protocol，IP)和用户数据报协议(User Datagram Protocol，UDP)，通信端口默认使用123。使用NTP进行时间同步的设备及其传输链路组成时间同步子网络，该网络中将设备依据其精度和重要性编码为0～15的级别，编码数越小的设备，时钟精度和重要性越高。第0级设备是高精度计时设备，如GPS时钟，被称为参考时钟。往下依次是时间服务器，以及各种具备较高时钟精度的设备，各级别中的设备即可作为上一级设备的客户端，又可作为下一级设备的服务器。NTP时间同步系统示意图如图1所示。NTP报文的数据格式如图2所示。

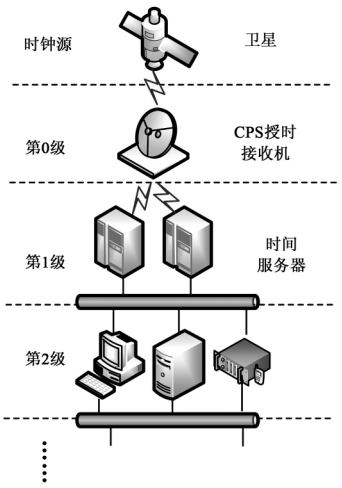

图1 NTP时间同步系统示意图

*NTP*的工作流程如下。首先客户端发送一条*NTP*报文，并打上发送时刻客户端的时间戳T_1；服务器收到此报文后打上接收时刻服务器的时间戳T_2；服务器经过相应处理后再将此报文发回客户端，并打上发送时刻服务器的时间戳

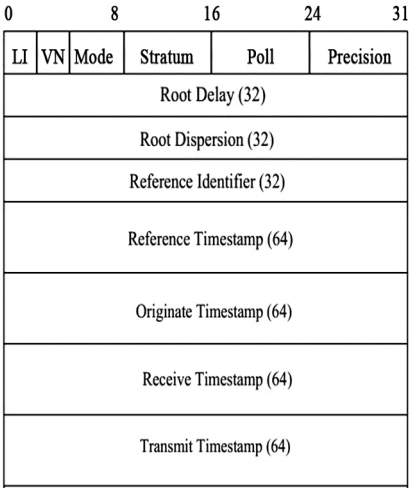

图2 NTP报文的数据格式

T_3；客户端收到此报文后再打上接收时刻客户端的时间戳T_4。*NTP*报文传输示意图如图3所示。

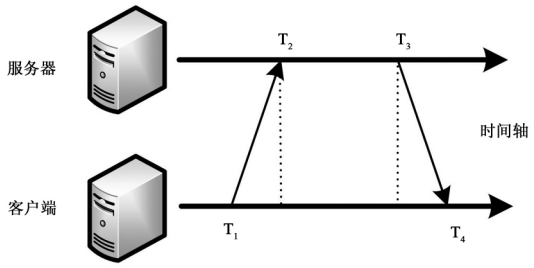

图3 *NTP*报文传输示意图

设客户端和服务器之间的时间偏移量为θ，报文从客户端发送到服务器的时延为δ_1，从服务器到客户端的时延为δ_2，总的链路时延为δ，得到如下三个方程

$$T_2 - T_1 = \theta + \delta_1 \tag{1}$$

$$T_4 - T_3 = \delta_2 - \theta \tag{2}$$

$$\delta_1 + \delta_2 = \delta \tag{3}$$

可假设$\delta_1 = \delta_2 = \frac{\delta}{2}$，式(1)－(3)变换为

$$T_2 - T_1 = \theta + \frac{\delta}{2} \tag{4}$$

$$T_4 - T_3 = \frac{\delta}{2} - \theta \tag{5}$$

推出理想状态下的链路时延δ和时间偏移量θ为

$$\delta = (T_4 - T_1) - (T_3 - T_2) \tag{6}$$

$$\theta = \frac{(T_2 - T_1) + (T_3 - T_4)}{2} \tag{7}$$

将客户端的本地时间加上时间偏移量θ即为校正后的时间T_{new}

$$T_{new} = T_4 + \theta \tag{8}$$

考虑到客户端和服务器的本地时钟误差和测量误差，真实的时间偏差值θ_0应在θ的某领域内波动。因为$\theta_0 + \delta_1 = T_2 - T_1$，且客户端到服务器的网络时延$\delta_1 = T_2 - T_1 - \theta_0 \geq 0$，故$T_2 - T_1 \geq \theta_0$. 同样的，有$\delta_2 - \theta_0 = T_4 - T_3$，故$\theta_0 \geq T_3 - T_4$，记为

$$T_3 - T_4 \leqslant \theta_0 \leqslant T_2 - T_1 \tag{9}$$

结合式(4)和(5)，式(9)等价于

$$\theta - \frac{\delta}{2} \leqslant \theta_0 \leqslant \theta + \frac{\delta}{2} \tag{10}$$

设随机变量f表示本地晶振的频率误差，$f \in [-\varphi, \varphi]$，$\varphi$表示晶振频率的最大容错度，故一段时间$t$内频率误差可表示为$f \times t \in [-\varphi t, \varphi t]$，$\varphi t$可视为一定时间内晶振频率的最大误差。

设随机变量r表示计算机读取当前时刻的读取误差，$r \sim U[-\rho, 0]$，ρ为时钟的最大读取误差。随机变量f和r相互独立。用ρ_c、ρ_s、f_c、f_s表示两地时钟的最大读取误差和晶振频率误差，下标s表示服务器端的相关参数，下标c表示客户端的相关参数，时间同步过程的最大可能误差为ε，由此可得四个时间戳$T_1 \sim T_4$的误差分别为

$$\varepsilon_1 = [-\rho_c, 0] \tag{11}$$

$$\varepsilon_2 = [-\rho_s, 0] \tag{12}$$

$$\varepsilon_3 = [-\rho_s, 0] + f_s(T_3 - T_2) \tag{13}$$

$$\varepsilon_4 = [-\rho_c, 0] + f_c(T_4 - T_1) \tag{14}$$

由式(1)可推得δ的误差区间为

$$[-\rho_c - \rho_s, \rho_c + \rho_s] + f_c(T_4 - T_1) - f_s(T_3 - T_2) \tag{15}$$

令ε_θ表示θ测量值的最大可能误差，ε_δ表示δ测量值的最大可能误差，φ_c和φ_s分别表示两地时钟晶振的频率最大容错度，φ_s的值可从时间服务器返回的NTP报文中的Root Dispersion项得到，而φ_c的值常为$\frac{1}{86400}$。因为$f_c \leqslant |\varphi_c|$，$f_s \leqslant |\varphi_s|$，记$\rho = \max(\rho_c, \rho_s)$，故链路延迟$\delta$的最大可能误差为

$$\varepsilon_\delta = \rho + \varphi_c(T_4 - T_1) + \varphi_s(T_3 - T_2) \tag{16}$$

同样的，根据式(6)，可得真实的时间偏移量θ_0的最大可能误差为

$$\varepsilon_\theta = \frac{\rho + \varphi_s(T_4 - T_1) + \varphi_c(T_3 - T_2)}{2} \tag{17}$$

即真实的时间偏移量θ_0的范围是

$$\theta - \varepsilon_\theta - \frac{\delta + \varepsilon_\delta}{2} \leqslant \theta_0 \leqslant \theta + \varepsilon_\theta + \frac{\delta + \varepsilon_\delta}{2} \tag{18}$$

3 模块设计

3.1 跨平台模块设计概念

跨平台软件开发，既不依赖于操作系统，也不依赖硬件环境，致力于使应用程序可以几乎不做任何的修改就能运行在不同的平台上[12]。Linux和Windows系统的底层差异和应用编程接口的差异导致使用C++语言进行跨平台编译出现困难，需要在代码中利用条件编译或工厂模式的方法实现代码的一次编写，多次编译[13]。在Windows平台需要将头文件和函数实现编译生成.lib格式的静态库文件或.dll格式的动态库文件，Linux平台则对应.a格式的静态库文件或.so格式的动态链接库，再将所有用到的库文件链接起来，形成可执行文件。源程序、目标文件和可执行文件的关系如图4所示。

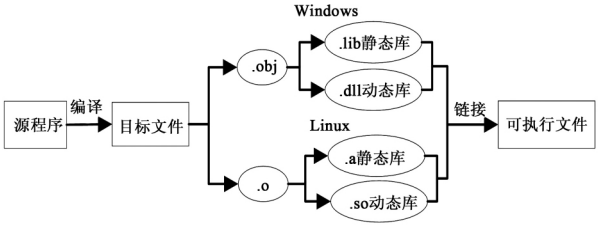

图4 源程序、目标文件和可执行文件的形成关系

3.2 关键代码设计

将NTP时间戳命名为NtpTimeStamp，整数部分和小数部分均为32位的无符号整型。

```
struct NtpTimeStamp
{
    uint32_t integer; //整数部分
    uint32_t fraction; //小数部分
};
```

NTP请求报文命名为NtpPacket，构造如下。

```
struct NtpPacket
{
    uint8_t leapVerMode; //跳变、版本号、模式
    uint8_t stratum; //层级
    uint8_t poll; //轮询间隔
    int8_t precision; //本地时钟精度
    int rootDelay; //往返延迟
    int rootDispersion; //最大误差
    uint32_t referenceIdentifier; //参考源标识
    NtpTimeStamp referenceTimestamp;  //T1
    NtpTimeStamp originateTimestamp;  //T2
    NtpTimeStamp receiveTimestamp; //T3
    NtpTimeStamp transmitTimestamp; //T4
};
```

NTP时间同步代码的核心部分由基类CntpBase及其派生类Win_ntp和Lnx_ntp构成，分别适用于Windows和Linux系统。

```
class CntpBase
{
    public:
    CntpBase();
    virtual ~CntpBase();
    int constructPacket(char *  packet);
    virtual void createSocket() = 0;
```

```
    virtual void bindServAddr( struct sockaddr_in &h, const char s1 [] ) = 0;    //绑定套接字
    virtual int getNtpTime( int sk, struct sockaddr_in *  addr, struct NtpPacket *  ret_time) = 0;
     //获取时间服务器的时间
    virtual int setLocalTime( struct NtpPacket *  pnew_time_packet) = 0;
    //校正本地时间
    int setServerIP( const char s1 [], const char s2 []);
  //填写时间服务器 IP 地址
    virtual void showTime( ) = 0; //显示系统时间
    virtual int Ntpdate( ) = 0;    //接口函数
  };
```

NTP 时间同步程序的流程图如图 5 所示。

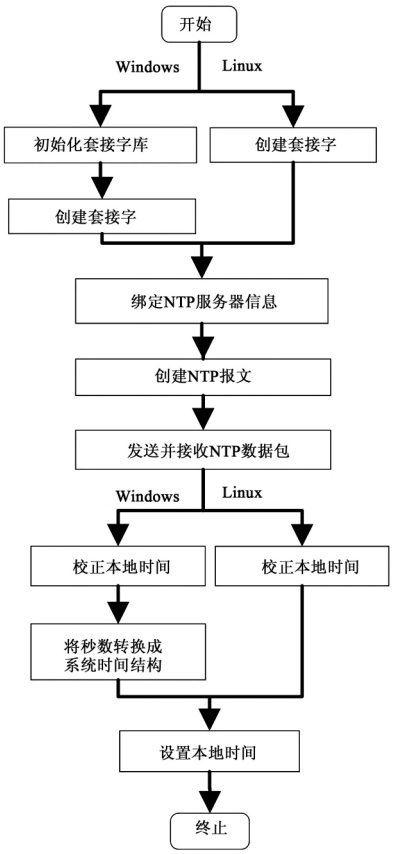

图 5　NTP 时间同步程序流程图

Windows 系统的套接字类型是 SOCKET，需要先初始化套接字库并调用静态库 Ws2_32. lib 才能创建套接字。Linux 系统的套接字为 int 类型，可直接创建，然后调用 bindServAddr( ) 函数将已知的时间服务器 IP 地址与套接字绑定，再使用 sendto( ) 函数向时间服务器发送 NTP 请求。

constructPacket( ) 函数用于创建 NTP 报文、初始化报文头部并打上时间戳。

```
int CntpBase::constructPacket( char *  packet)
{
    struct NtpPacket *  ntp;
    ntp = ( struct NtpPacket *  ) packet;
    //赋值数据报的报头部分
    ntp->leapVerMode = 35;
    ntp->stratum = STRATUM;
    ntp->poll = POLL;
    ntp->precision = PRECISION;
    //打上第一个时间戳
    struct timeval tv;
    gettimeofday( &tv, NULL) ;
    ntp->referenceTimestamp. integer=htonl( tv. tv_sec+JAN_1970) ;
    ntp->referenceTimestamp. fraction=htonl( USEC2FRAC( tv. tv_usec) ) ;
    return 0;
}
```

其中，gettimeofday(&tv, NULL) 函数用于获取当前系统的精确时间并保存在结构体 tv 中，其值为从 1970 年 1 月 1 日 0 时到当前时刻的秒数，精确到微秒。而从 GPS 上获取的 NTP 时间戳是从 1900 年 1 月 1 日 0 时到当前时刻的秒数，故需要将系统返回的秒数加上 70 年的秒数。gettimeofday( ) 函数所在的头文件 sys/time. h 只存在于 Unix 系统，Windows 下则需要在源代码中增加带条件编译的 gettimeofday(struct timeval * tp, void * tzp) 函数。其中* tp 指向 timeval 结构的时间，* tzp 指向时区，默认使用 NULL，根据系统本身的时区而定。

```
#ifdef WIN32
int gettimeofday( struct timeval * tp, void * tzp)
{
    time_t clock;    //秒数
    struct tm tm;    //年月日时分秒的结构
    SYSTEMTIME wtm;    //系统时间的结构
    GetLocalTime( &wtm) ; //获取系统时间
    //将系统时间转换成 NTP 报文需要的时间
    tm. tm_year = wtm. wYear - 1900;
    tm. tm_mon = wtm. wMonth - 1;
    tm. tm_mday = wtm. wDay;
    tm. tm_hour = wtm. wHour;
    tm. tm_min = wtm. wMinute;
    tm. tm_sec = wtm. wSecond;
    tm. tm_isdst = -1;
    clock = mktime( &tm) ;
    tp->tv_sec = ( long) clock;
    tp->tv_usec = wtm. wMilliseconds *  1000;
    return 0;
}
```

```
#endif
```

为了保证程序在主时间服务器故障时不出现阻塞状态，使用 select( ) 函数来确定多个套接口的状态。该函数可以同时监视多个描述符的读写状态，从而实现 I/O 多路复用[14]。一旦发往某个时间服务器的套接字在超时时间之内没有回复，select( ) 函数会返回 0，表示超时，并使用另一个套接字向备用时间服务器发送 NTP 请求，避免程序被阻塞。

客户端收到时间服务器返回的 NTP 报文后，根据式(6)和(7)计算出链路延迟 δ 和时间偏移量 θ，计算时间偏移量 θ 的函数 GetTheta( ) 如下。

```
double GetTheta( const struct NtpPacket * ntp, const struct timeval * recvtv)
{
    double t1, t2, t3, t4;
    t1 = NTPSTAMP2SEC( &ntp->originateTimestamp) ;
    t2 = NTPSTAMP2SEC( &ntp->receiveTimestamp) ;
    t3 = NTPSTAMP2SEC( &ntp->transmitTimestamp) ;
    t4 = recvtv->tv_sec + recvtv->tv_usec / 1000000.0;
    return ( ( t2 - t1) + ( t3 - t4) ) / 2;
}
```

其中* ntp 指向存储 NTP 报文的结构，* recvtv 是客户端接收到 NTP 报文后的本地时间，NTPSTAMP2SEC 是将 NTP 时间戳转换成秒数的宏定义。

```
#define NTPSTAMP2SEC ( x)  ( ( double) ( ntohl ( ( ( struct NtpTimeStamp * ) ( x) ) ->intpart) - JAN_1970 + FRAC2USEC( ntohl( ( ( struct NtpTimeStamp * ) ( x) ) ->frac-part) ) / 1000000.0 ) )
```

根据式(8)编写如下代码得到校正后的时间，存储在 struct timeval newtv 中。

```
newtv. tv_sec += ( int) theta;
newtv. tv_usec += theta - ( int) theta;
```

Linux 系统有现成的修改系统时间的函数 settimeofday( )。Windows 环境则需要搭配系统时间结构体 SYSTEMTIME 变量来辅助修改系统时间。

```
__time64_t timep; //存储校正后的秒数
timep = newtv. tv_sec;
struct tm tv1;
_gmtime64_s( &tv1, &timep) ; //将秒数转换成时间结构
SYSTEMTIME sysTime;
//将 NTP 报文格式的秒数转换成系统时间
sysTime. wYear = 1900 + tv1. tm_year;
sysTime. wMonth = 1 + tv1. tm_mon;
sysTime. wDay = tv1. tm_mday;
sysTime. wHour = tv1. tm_hour;
sysTime. wMinute = tv1. tm_min;
sysTime. wSecond = tv1. tm_sec;
sysTime. wDayOfWeek = tv1. tm_wday;
sysTime. wMilliseconds = ( WORD) newtv. tv_usec/1000;
```

至此，系统的时间已经校正为准确的 UTC 时间。对外提供一个接口函数 Ntpdate( )，实现整个网络时间同步功能。配置好主备服务器的 IP 地址和时间同步间隔后，即可调用 Ntpdate( ) 函数对该程序所在的系统校时。

4 实验结果

选用公网上的两台时间服务器，IP 地址分别为 203. 107. 6. 88 和 182. 92. 12. 11，测试时间同步模块在广域网中的性能，选取链路延迟和时间偏移量作为性能指标。在实验之前先将系统的时间调早 1 个小时，然后每隔 3 秒钟调用一次时间同步接口函数 Ntpdate( ) 进行时间同步，分别记录 NTP 报文中的 4 个时间戳数值，截取部分实验结果如图 6 所示。

```
root@hce:/home/hce/桌面# date
2018年 11月 14日 星期三 20:39:52 CST
root@hce:/home/hce/桌面# date -s 19:40:00
2018年 11月 14日 星期三 19:40:00 CST
root@hce:/home/hce/桌面# ./ntpdate.out
t1 = 1542195602.007667,
t2 = 1542199200.823073,
t3 = 1542199200.823095,
t4 = 1542195602.088282
delta = 0.080593, theta = 3598.775109
Wed Nov 14 20:40:00 2018

t1 = 1542199203.864252,
t2 = 1542199203.903074,
t3 = 1542199203.903095,
t4 = 1542199203.942365
delta = 0.078092, theta = -0.000224
Wed Nov 14 20:40:03 2018

t1 = 1542199206.942745,
t2 = 1542199206.983270,
t3 = 1542199206.983288,
t4 = 1542199207.022039
delta = 0.079276, theta = 0.000887
Wed Nov 14 20:40:07 2018
```

图 6　前三次时间同步模块测试

图 6 中 t1 ~ t4 是 NTP 报文中的四个时间戳转换而来的秒数。delta 和 theta 分别给出了链路延迟 δ 和时间偏移量 θ。记录前 50 个样本的实验结果，绘制参数 δ 和 θ 如图 7 所示。当链路延迟 δ 维持稳定时，时间偏移量 θ 均小于 1 毫秒。当网络情况出现波动时，链路延迟 δ 增大，时间偏移量 θ 相应得出现小幅增加，波动范围在十几毫秒以内。δ 和 θ 的统计信息记录在表 1 中。

表 1　前 50 个样本的 δ 和 θ 值的统计结果

	同步周期/s	样本数	最大值/ms	最小值/ms	中位数/ms	标准差
δ	3	50	14.987	0.224	0.855	2.719
θ	3	50	110	58.3	70.2	14.4

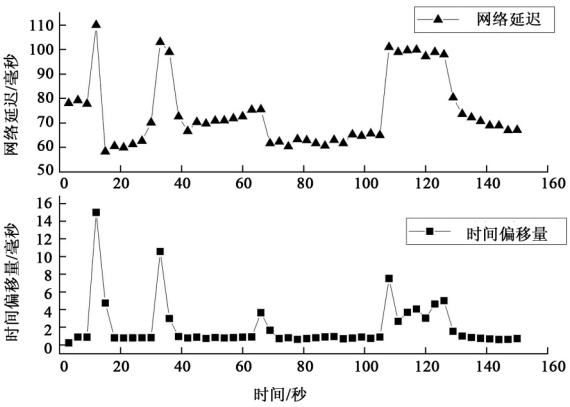

图7　前50个样本的δ和θ值的趋势

由图7可知，链路延迟和时间同步精度成正相关。链路延迟主要由传输链路延迟、网络时钟处理延迟和网络设备延迟三部分组成[15]。传输链路延迟是指报文在网络电缆中的传输时间，其特点是相对固定，可通过增加NTP时钟滤波算法改善；网络时钟处理延迟是指报文从应用层到物理层所需的时间，因为NTP打时间戳的步骤是在应用层完成的，该延迟跟协议栈处理时间直接相关；网络设备延迟是指报文在网络设备中停留的时间，跟网络拥堵程度直接相关，具有不确定性[16]。

在Windows环境下，将第一个时间服务器的IP地址故意改错，模拟主时间服务器故障，程序应能自动向备用服务器发送NTP请求并校正系统时间，因为select()机制的超时时间为3秒，故现在时间同步间隔为6秒。Windows系统下的主备切换测试结果如图8所示。

图8　Windows系统下模拟主服务器故障

5　结束语

本文提出的基于NTP时间同步程序可以通过生成静态库或动态库的方式，被Windows和Linux环境下的其它程序所调用，可提高有时间同步需求的跨平台软件开发效率，时间同步误差为亚毫秒级。当主服务器故障时，程序可自动向备用服务器进行时间同步，同步时间间隔可根据系统对时间的敏感性以及网络带宽情况灵活调整。该模块适用于时间同步精度在毫秒级、无法额外增加硬件辅助的分布式跨平台系统。

参考文献：

[1] 董绍武，王燕平，武文俊，等. 国际原子时及NTSC守时工作进展[J]. 时间频率学报，2018，41(2)：73-79.

[2] Yuan-Xi Yang，Yang-Yi Xu，Jinlong LI et al. Progress and performance evaluation of BeiDou global navigation satellite system：Data analysis based on BDS-3 demonstration system[J]. Science China Earth Sciences，2018，61：614.

[3] 李培基，李卫，朱祥维，等. 网络时间同步协议综述[J]. 计算机工程与应用，2019，55(3)：30-38.

[4] Jie Xu，Liang Xu，Lian Dong et al. Research on network timing system based on NTP[C]. 2017 13th IEEE International Conference on Electronic Measurement & Instruments，2017：356-360.

[5] Jing-Wen Tian，Mei-Juan Gao，Yue Zhou. Wireless sensor network for multi-channel 3D data synchronizing acquisition system and visual simulation research[C]. Wireless Pers Commun，2017，95：1981-2001.

[6] Wen-Lun Yang，Min-Yue Fu. A filter - based clock synchronization protocol for wireless sensor networks[J]. Asian Journal of Control，2019，21(3).

[7] 李德骏，汪港，杨灿军，等. 基于NTP和IEEE1588海底观测网时间同步系统[J]. 浙江大学学报(工学版)，2014，48(01)：1-7.

[8] 袁炳南，张国旺. 遥测网络时间同步技术实现及检测方法研究[J]. 计算机测量与控制，2018，26(04)：15-18.

[9] Guang-Hua Gong，Hong-Ming Li. High- precision time distribution based on optical ethernet[J]. Navigation Positioning & Timing，2017，4(6).

[10] Ilir Capuni，Nertil Zhuri，Rejvi Dardha. Timestream：exploiting video streams for clock synchronization[J]. Ad Hoc Networks，2019，91.

[11] Mitchell W S. Overdick，Joseph E. Canfield，Andrew G. Klein. A software-defined radio implementation of timestamp-free network synchronization[C]. IEEE International Conference on Acoustics，Speech and Signal Processing (ICASSP)，2017：1193-1197.

[12] 张驰. 基于C++语言的跨平台软件开发的设计与实现[D]. 北京交通大学，2010.

[13] Colombet B，Woodman M，Badier J M. AnyWave：A cross-platform and modular software for visualizing and processing electrophysiological signals[J]. Journal of Neuroscience Methods，2015，242：118-126.

(下转第333页)

4.4 大数据挖掘准确率

为了进一步验证本文方法的有效性，对本文提出的基于蚁群算法的非结构化大数据深度挖掘方法、基于中文分词的电子病历数据挖掘方法和用电信息采集系统非结构化数据挖掘方法的非结构化大数据深度挖掘结果与实际测试结果进行对比分析，对比结果如图6所示。

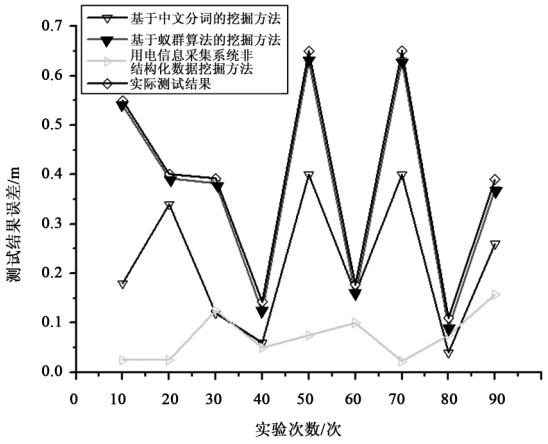

图6 三种方法的挖掘结果误差

根据图6可知，本文方法的非结构化大数据深度挖掘结果与实际测试的挖掘结果一致，而基于中文分词的电子病历数据挖掘方法和用电信息采集系统非结构化数据挖掘方法的非结构化大数据深度挖掘结果与实际测试的挖掘结果误差相差较大。说明本文方法的非结构化大数据深度挖掘的准确率比基于中文分词的电子病历数据挖掘方法和用电信息采集系统非结构化数据挖掘方法的数据挖掘准确率高。

5 结束语

本文提出基于蚁群算法的非结构化大数据深度挖掘方法。在蚁群算法的支持下，非结构化大数据深度挖掘方法联合OpenFlow控制器，限定大数据批量处理模块的执行权限，再借助开源型框架，定义PPC-Tree树状体及大数据K均值。而随着这种新型挖掘手段的应用，节点容纳承载量、匹配路径传输速率均出现不同程度的上升趋势，且非结构化大数据深度挖掘结果与实际测试的挖掘结果相一致，提高了数据挖掘的准确率。

参考文献：

[1] 刘晓阳，刘恩福，靳江艳. 基于蚁群算法的异步并行装配序列规划方法[J]. 机械工程学报，2019，55(9)：107-119.

[2] 谢剑，周小茜，童凌，等. 基于中文分词的电子病历数据挖掘技术[J]. 湖南科技学院学报，2016，37(10)：54-59.

[3] 祝恩国，刘宣，葛磊蛟. 用电信息采集系统非结构化数据管理设计[J]. 电力系统及其自动化学报，2016，28(10)：123-128.

[4] 郑英姿，张福泉，李立杰. 基于强化学习的大数据频繁项集挖掘算法[J]. 计算机工程与设计，2019，23(8)：2270-2277.

[5] 黄彦璐，张震，张喆，等. 基于非侵入式负荷辨识和关联规则挖掘的用户柔性负荷区间预测[J]. 南方电网技术，2019，18(4)：60-66.

[6] 杨胜利，余亮，李超. 时空嵌入式网络用户轨迹序列模式挖掘仿真[J]. 计算机仿真，2019，36(4)：315-318，344.

[7] 邵哲平，周田瑞，潘家财，等. 基于AIS数据挖掘的受限水域船舶动态领域研究[J]. 地球信息科学学报，2018，20(5)：564-570.

[8] 王芳，饶德坤，游静，等. 基于改进蚁群算法的带硬时间窗的接送机场服务路径优化研究[J]. 系统科学与数学，2019，39(1)：76-89.

[9] 张述仁，徐雅，谢代梁，等. 基于蚁群算法的超声波水体悬移质浓度测量研究[J]. 传感技术学报，2019，(8)：1163-1168.

[10] 陆金钰，鲁梦. 蚁群算法在自适应索穹顶结构内力控制中的应用[J]. 东南大学学报(自然科学版)，2017，47(6)：1161-1166.

[作者简介]

金 欣(1982-)，男(汉族)，江西赣州人，讲师，主要研究方向：数据挖掘。

(上接第221页)

[14] Benjamin Hesmans，Olivier Bonaventure. An enhanced socket API for multipath TCP[C]. Proceedings of the 2016 Applied Networking Research Workshop，2016：1-6.

[15] Rakesh Kumar，Monowar Hasan，Smruti Padhy et al. End-to-end network delay guarantees for real-time systems using SDN[C]. 2017 IEEE Real-Time Systems Symposium (RTSS)，2017：231-242.

[16] Martin Langer，Kristof Teichel，Dieter Sibold et al. Time synchronization performance using the network time security protocol[C]. 2018 European Frequency and Time Forum(EFTF)，2018：138-144.

[作者简介]

何承恩(1994-)，男(汉族)，四川成都人，硕士研究生，主要研究领域为VoIP语音通信、计算机视觉；

王 忠(1964-)，男(汉族)，四川人，副教授，硕士研究生导师，主要研究领域为无线电导航与室内定位、无线与移动通信、网络通信、大数据与云计算、深度学习与人工智能；

岳佳欣(1982-)，女(汉族)，四川南充人，硕士研究生，主要研究方向为电子与无线通信，大数据、人工智能。