

实验 3-1：基于 UDP 服务设计可靠传输协议并编程实现

姓 名： 杨科迪 学 号： 1813828
专 业： 计算机科学与技术 指导老师： 徐敬东
实验时间： 2020 年 12 月 10 日 实验地点： 实验楼 A 区 204

目录

1 实验目的	2
2 实验原理	2
3 实验步骤	4
3.1 预备工作	4
3.1.1 定时器	4
3.1.2 重传超时时间计算	5
3.2 协议设计	6
3.3 建立连接	6
3.3.1 发送端	6
3.3.2 接收端	7
3.4 差错检测	8
3.5 确认重传	8
3.5.1 发送端	8
3.5.2 接收端	10
3.6 连接关闭	11
3.6.1 发送端	11
3.6.2 接收端	11
3.7 发送端	12
3.8 接收端	13
4 实验结果	14

1 实验目的

1. 用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：

- 建立连接
- 差错检测
- 确认重传
- 流量控制采用停等机制

2. 完成给定测试文件的传输

2 实验原理

本实验基于 rdt3.0。

1. 发送端的状态机如图1所示。发送端先发送序号为 0 的报文段，然后开启定时器，等待接收端发送 ack0：

- 接收的报文段损坏或是 ack1，则进行等待
- 接收的报文段未损坏并且是 ack0，停止定时器，进入下一发送周期，等待发送序号为 1 的报文段
- 定时器超时，重传序号为 0 的报文段，重启定时器

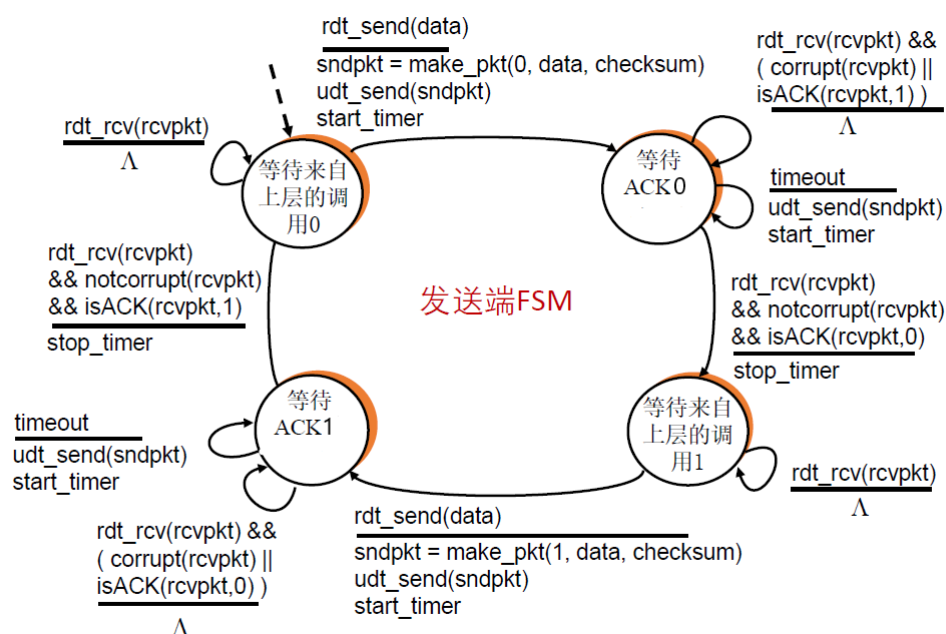


图 1: rdt3.0 发送端状态机

2. 接收端的状态机如图2所示。接收端先等待序号为 0 的报文段, 收到报文段后:

- 报文段损坏或是序号 1, 则发送 ack1
- 报文段未损坏并且是序号 0, 发送 ack0, 进入下一接收周期

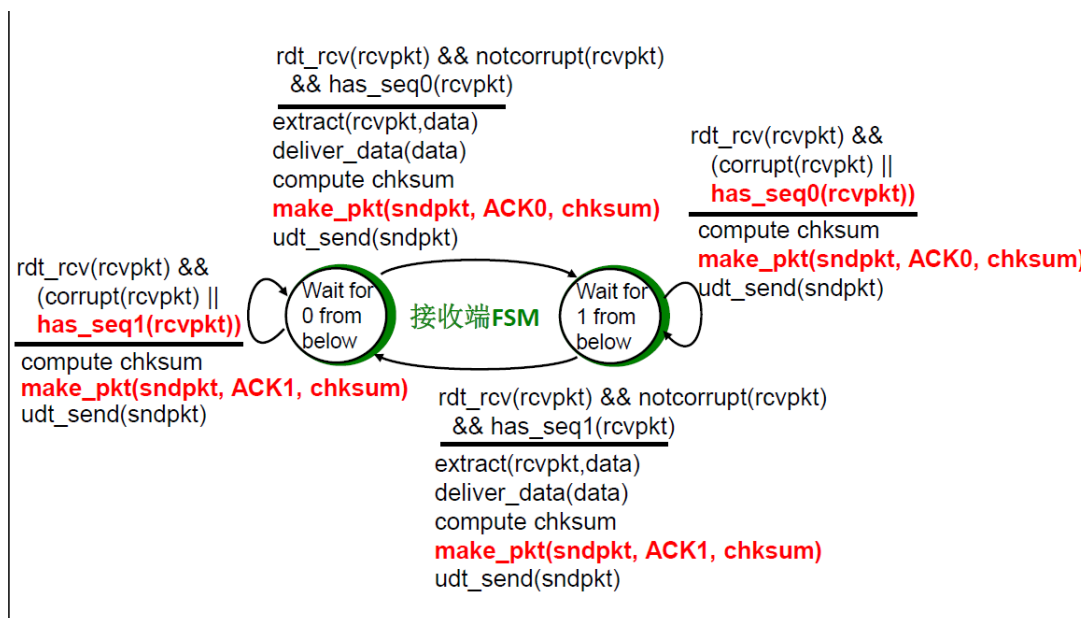


图 2: rdt3.0 接收端状态机

3. 由于实现的是单向传输, 进行了所谓的两次握手过程, 连接建立过程如图3所示。发送端先向接收端发送 syn 报文, 序号为 0, 接收端收到 syn 后发送 ack 报文, ack=0, 接收端收到 ack0 后, 连接建立。使用了连接建立定时器, 75 秒后连接还未建立, 则连接建立失败。

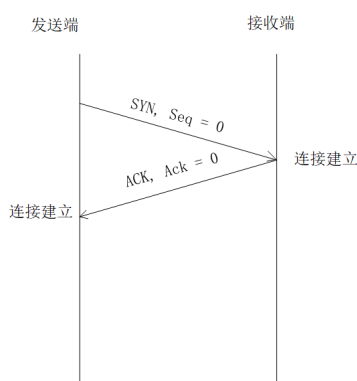


图 3: 连接建立

4. 连接关闭过程如图4所示。发送端先发送 fin 报文, 接收端收到后发送 ack 报文并关闭连接, 发送端收到 ack 后关闭连接

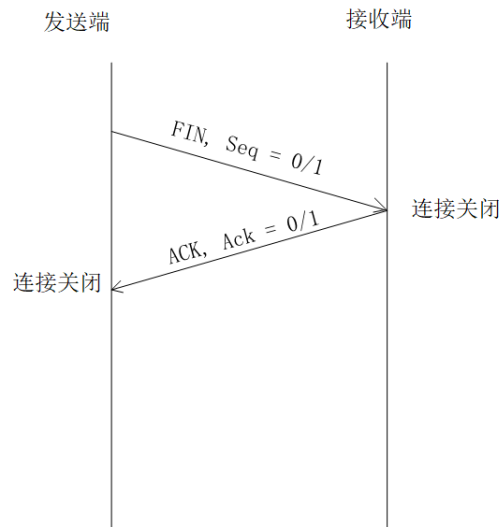


图 4: 连接关闭

3 实验步骤

3.1 预备工作

3.1.1 定时器

编写定时器类如下

```
1 //计时器初始化
2 Timer::Timer()
3 {
4     timeout = 0;
5     QueryPerformanceFrequency(&f);
6     isStart = false;
7 }
8
9 //计时器开始计时
10 void Timer::startTimer()
11 {
12     isStart = true;
13     setTimer();
14 }
15
16 //得到当前 click 数
17 void Timer::setTimer()
18 {
19     QueryPerformanceCounter(&timer_start);
```

```

20 }
21
22 //计时器停止计时
23 void Timer::stopTimer()
24 {
25     isStart = false;
26 }
27
28 //得到时间差
29 double Timer::getDiff()
30 {
31     QueryPerformanceCounter(&timer_end);
32     return ((timer_end.QuadPart - timer_start.QuadPart) * 1000.0 / f.QuadPart);
33 }
34
35 //检测是否超时
36 bool Timer::testTimeOut()
37 {
38     return getDiff() > timeout;
39 }
40
41 //设置超时时间
42 void Timer::setTimeOut(double time)
43 {
44     timeout = time;
45 }

```

定时器类用于检测是否超时、计算传输时延

3.1.2 重传超时时间计算

采用 Jacobson/Karels 算法，对 RTO 进行估算

```

1 //增加一次 RTT，估算下一次 RTO
2 void RTO::addSampleRTT(double sampleRTT)
3 {
4     EstimatedRTT = 0.875 * EstimatedRTT + 0.125 * sampleRTT;
5     DevRTT = 0.75 * DevRTT + 0.25 * abs(sampleRTT - EstimatedRTT);
6     rto = EstimatedRTT + 4 * DevRTT;
7 }

```

3.2 协议设计

实验 3-1 使用的协议如下:

```
1 struct packet
2 {
3     unsigned char saf;           //序列号、确认号、标志字段 (ACK,SYN,FIN)
4                                   //0:seqnum 1:acknum 2:finflag 3:synflag 4:ackflag
5     unsigned short checksum;     //检验和
6     unsigned short dataLen;      //数据长度, 以字节为单位
7     unsigned char data[MAXBUFSIZE]; //数据
8 };
```

各字段的含义如下:

saf 序列号 (bit0)、确认号 (bit1)、ACK 标志字段 (bit2)、SYN 标志字段 (bit3)、FIN 标志字段 (bit4)。由于实现的是停等协议, 所以序列号和确认号都只需要一个 bit

checksum 检验和字段, 采用和 IP 和 TCP 类似的计算方法

dataLen 数据字段的长度, 由于单个报文段的大小通常在一千字节左右, 因此使用最大为 65535 的 unsigned short 类型已经足够可以表示了

data 应用层的数据

3.3 建立连接

3.3.1 发送端

```
1 //建立连接
2 int rdt::connect(SOCKET s, const sockaddr *name)
3 {
4     /******/
5     //封装 SYN 报文
6     packet *p = new packet;
7     memset(p, 0, sizeof(packet));
8     p->saf = setSyn(p->saf);
9     p->saf = setSeqNum(p->saf, curSeq[s]);
10    p->dataLen = 0;
11    p->checksum = ~getChecksum(p);
12    int len = sizeof(packet) - MAXBUFSIZE;
13
14    /******/
15
16    Timer connection_timer; //设置超时时间为 75 秒
```

```

17     connection_timer.setTimeout(connTimeOut);
18     connection_timer.startTimer();
19     while (1)
20     {
21         if(connection_timer.testTimeOut()) return -1;
22         if(sendto(s, (char*)p, len, 0, name, sizeof(sockaddr)) == -1) continue;    //发送 SYN 报文
23         if(recvfrom(s, (char*)&t, len, 0, &recvaddr, &fromlen) < 0) continue;    //收到报文
24         //收到的报文段检验和正确并且按序, 连接建立成功
25         if(check(&t) && isAck(t.saf) && (getAckNum(t.saf) == curSeq[s]))
26         {
27             connection[s] = *name;
28             curSeq[s] = !curSeq[s];
29             return 0;
30         }
31     }
32     return -1;    //连接建立失败
33 }

```

发送端使用 connect 函数建立连接。封装 SYN 报文, 序号为 0, 数据长度为 0, 开启连接建立定时器, 向接收端发送建连请求, 如果收到 ack0, 则连接建立成功。

3.3.2 接收端

```

1 //应用层接收连接
2 int rdt::accept(SOCKET s)
3 {
4     /*****/
5     while (1)
6     {
7         if(recvfrom(s, (char*)&pkt, sizeof(packet) - MAXBUFSIZE, 0, &addr, &addrlen) > 0)
8         {
9             if(check(&pkt) && isSyn(pkt.saf))    //检验和正确并且是 SYN 报文段
10            {
11                int seq = getSeqNum(pkt.saf);
12                packet ackPkt;
13                connection[s] = addr;    //建立连接
14                expectedSeq[s] = !seq;    //期待序列号取反
15                makeAckPkt(s, &ackPkt, seq);
16                sendto(s, (char*)&ackPkt, sizeof(packet) - MAXBUFSIZE,
17                    0, &addr, sizeof(sockaddr_in));    //发送对 syn 报文段的 ack

```

```

18         return 0;
19     }
20 }
21 }
22 }

```

接收端收到 SYN 报文后，向发送端发送 ACK 报文，然后将发送端的地址存起来，接收端的 socket/发送端的地址映射形成一条连接。

3.4 差错检测

检验和的计算如下

```

1 unsigned short getChecksum(packet *p)
2 {
3     unsigned int ans = 0;
4     unsigned short res;
5     ans += (unsigned short)p->saf << 8;
6     ans += p->checksum;
7     ans += p->dataLen;
8     for(int i = 0; i < p->dataLen; i++)
9         if(i%2) ans += p->data[i];
10        else ans += (unsigned short)p->data[i] << 8;
11    ans = (ans >> 16) + (ans & 0xffff);
12    res = (ans >> 16) + (ans & 0xffff);
13    return res;
14 }
15
16 #define check(p) (getChecksum(p) == 0xffff)    //检查校验和

```

通过检验和是否全 1 判断报文是否损坏。

3.5 确认重传

3.5.1 发送端

```

1 //应用层发送数据
2 int rdt::send(SOCKET s, const char *buf, int len)
3 {
4     /******//
5     makePkt(s, &pkt, buf, len);

```



```

6
7  /******/
8  int flag = true;    //记录是否进行了重传，如没有进行重传，则计算 RTO
9  Timer timer;
10 timer.setTimeout(curRTO[s].rto);
11 //将应用层数据封装到报文中发送
12 if(sendto(s, (const char *)&pkt, sizeof(pkt) - MAXBUFSIZE + len, 0,
13 &connection[s], sizeof(sockaddr)) == -1) return -1;
14
15 while(1)
16 {
17     if(timer.testTimeout())    //超时重传
18     {
19         flag = false;
20         if (sendto(s, (const char *)&pkt, sizeof(pkt) - MAXBUFSIZE + len, 0,
21 &connection[s], sizeof(sockaddr)) == -1) return -1;
22         timer.setTimeout(timer.timeout * 2);
23         timer.setTimer();
24     }
25
26     if(recvfrom(s, (char *)&ackPkt, sizeof(pkt) - MAXBUFSIZE, 0, &recvaddr,
27 &fromlen) <= 0)
28         continue;
29     //ack 报文段检验和正确，并且按序
30     if(check(&ackPkt) && isAck(ackPkt.saf) && (getAckNum(ackPkt.saf) == curSeq[s]))
31     {
32         curSeq[s] = !curSeq[s];
33         timer.stopTimer();
34         if(flag) curRTO[s].addSampleRTT(timer.getDiff());    //计算下一次 RTO
35         return 0;
36     }
37 }
38 }

```

发送端首先将应用层的数据封装进报文，序号为当前序号，开启定时器后发送。如果收到的报文正常并且 ack 为当前序号，则序号取反，进入下一发送周期，等待应用层发送数据；如果收到的报文损坏或者序号不是当前序号则进行循环；如果超时则重传当前序号的报文段，然后重启定时器。如果当前报文段未进行重传被确认，则将改报文段的 RTT 加入到 curRTO 中，估算下一次超时时间¹。

¹参考 TCP 决不为已被重传的报文段计算 SampleRTT，它仅为传输一次的报文段测量 SampleRTT

3.5.2 接收端

```
1 //接受数据并返回给应用层
2 int rdt::recv_deliver(SOCKET s, char *buf, int len)
3 {
4     /***/
5     while (1)
6     {
7         if(recvfrom(s, (char*)&pkt, sizeof(packet) - MAXBUFSIZE + len, 0,
8             &connection[s], &fromlen) <= 0) continue;
9         //数据没有差错并且没有重复接收, 发送 ack, 向应用层传递数据
10        if(check(&pkt) && (getSeqNum(pkt.saf) == expectedSeq[s]))
11        {
12            datalen = pkt.dataLen;
13            makeAckPkt(s, &ackPkt, getSeqNum(pkt.saf));
14            sendto(s, (char*)&ackPkt, sizeof(packet) - MAXBUFSIZE, 0,
15                &connection[s], sizeof(sockaddr));
16            if(isFin(pkt.saf))
17            {
18                sendto(s, (char*)&ackPkt, sizeof(packet) - MAXBUFSIZE, 0,
19                    &connection[s], sizeof(sockaddr));
20                connection.erase(s);
21                expectedSeq.erase(s);
22                return -1;
23            }
24            expectedSeq[s] = !expectedSeq[s];
25            for (int i = 0; i < len; i++)
26                buf[i] = pkt.data[i];
27            return len;
28        }
29        else //数据发生错误或重复接收, 发送 ack
30        {
31            makeAckPkt(s, &ackPkt, !expectedSeq[s]);
32            sendto(s, (char*)&ackPkt, sizeof(packet) - MAXBUFSIZE, 0,
33                &connection[s], sizeof(sockaddr));
34        }
35    }
36 }
```

接收端收到报文段后, 如果报文段有差错或不是当前期待的序号, 则发送上一序号的 ack; 如果报文段没有

差错并且序号是当前的序号则将数据返回给应用层。

3.6 连接关闭

3.6.1 发送端

```
1 //关闭连接
2 int rdt::close(SOCKET s)
3 {
4     /*make fin packet*/
5     while (1)
6     {
7         //发送 FIN 报文端
8         if(sendto(s, (char*)&p, len, 0, &recvaddr, sizeof(sockaddr)) == -1) continue;
9         if(recvfrom(s, (char*)&t, len, 0, &recvaddr, &fromlen) < 0) continue;
10        //报文段检验和正确并且按序, 连接关闭成功
11        if(check(&t) && isAck(t.saf) && getAckNum(t.saf) == curSeq[s])
12        {
13            connection.erase(s);
14            return 0;
15        }
16    }
17    return -1; //连接关闭失败
18 }
```

发送端发送 FIN 报文段, 如果收到无差错按序的 ack, 则关闭连接。

3.6.2 接收端

```
1 if(isFin(pkt.saf))
2 {
3     sendto(s, (char*)&ackPkt, sizeof(packet) - MAXBUFSIZE, 0, &connection[s], sizeof(sockaddr));
4     connection.erase(s);
5     expectedSeq.erase(s);
6     return -1;
7 }
```

接收端收到 FIN 报文段后返回 ack 报文, 关闭连接。

3.7 发送端

```
1 //发送单个文件
2 void sendFile()
3 {
4     /***/
5     if (rdt::send(sockSender, (const char*)&f, sizeof(f)) < 0)
6     {
7         cerr << " 传输文件失败\n";
8         return;
9     }
10    cout<<" 正在传输文件"<<filename<<" ..... \n";
11    char buf[MAXBUFSIZE];
12
13    fseek(file,0,SEEK_SET);
14    long long num = f.size / MAXBUFSIZE;
15    while (num--)
16    {
17        fread(buf, 1, MAXBUFSIZE, file);
18        if (rdt::send(sockSender, buf, MAXBUFSIZE) < 0)
19        {
20            cerr << " 传输文件失败\n";
21            fclose(file);
22            return;
23        }
24    }
25    int rem;
26    if ((rem = f.size % MAXBUFSIZE) > 0)
27    {
28        fread(buf, 1, rem, file);
29        if (rdt::send(sockSender, buf, rem) == -1)
30        {
31            cerr << " 传输文件失败\n";
32            fclose(file);
33            return;
34        }
35    }
36    cout<<" 传输文件成功\n";
37    fclose(file);
38 }
```

发送端调用传输层 send 函数传输文件

3.8 接收端

```
1  int recvFile(const char *dir)
2  {
3      /*****/
4      if(rdt::recv_deliver(sockRcvr, (char*)f, sizeof(ftp)) == -1) return -1;
5      strcat_s(path, f->name);
6
7      cout<<" 开始接收文件"<<f->name<<'\n';
8
9      FILE* file;
10     char buf[MAXBUFSIZE];
11     fopen_s(&file, path, "wb");
12     if(!file) return -1;
13     long long num = f->size / MAXBUFSIZE;
14     Timer timer;
15     timer.startTimer();
16     while(num--)
17     {
18         if(rdt::recv_deliver(sockRcvr, buf, MAXBUFSIZE) < 0) return -1;
19         fwrite(buf, 1, MAXBUFSIZE, file);
20     }
21     int rem;
22     if((rem = f->size % MAXBUFSIZE) > 0)
23     {
24         if(rdt::recv_deliver(sockRcvr, buf, rem) == -1) return -1;
25         fwrite(buf, 1, rem, file);
26     }
27     double t = timer.getDiff();
28     cout<<" 接收文件成功\n";
29     cout << " 传输时间:" << t << "ms  平均吞吐率:" << f->size * 1000 * 8 / (1024 * t) << "kbps\n";
30     fclose(file);
31     return 0;
32 }
```


接收端调用传输层的 recv_deliver 函数接收文件，计算文件传输时间和平均吞吐率。

4 实验结果

发送端向接收端传输 4 个测试文件，传输时间和平均吞吐率如表1所示。

文件名称	文件大小 (byte)	传输时间 (ms)	平均吞吐率 (kbps)
1.jpg	1,857,353	96.85	149822
2.jpg	5,898,505	298.56	154350
3.jpg	11,968,994	595.08	157135
helloworld.txt	1,655,808	87.31	148167

表 1: 测试文件传输结果



```
Microsoft Visual Studio 调试控制台
请输入发送端IP地址:127.0.0.1
请输入发送端口号:3235
请输入接收端的IP地址:127.0.0.1
请输入接收端的端口号:3245
请输入要传输的文件数:4
请输入要发送的文件C:\\Users\\94266\\Desktop\\course\\ComputerNetWorking\\work\\lab3\\hw3-1\\test\\1.jpg
正在传输文件C:\\Users\\94266\\Desktop\\course\\ComputerNetWorking\\work\\lab3\\hw3-1\\test\\1.jpg . . . . .
传输文件成功
传输时间:96.8523ms 平均吞吐率:149822kbps
请输入要发送的文件C:\\Users\\94266\\Desktop\\course\\ComputerNetWorking\\work\\lab3\\hw3-1\\test\\2.jpg
正在传输文件C:\\Users\\94266\\Desktop\\course\\ComputerNetWorking\\work\\lab3\\hw3-1\\test\\2.jpg . . . . .
传输文件成功
传输时间:298.556ms 平均吞吐率:154350kbps
请输入要发送的文件C:\\Users\\94266\\Desktop\\course\\ComputerNetWorking\\work\\lab3\\hw3-1\\test\\3.jpg
正在传输文件C:\\Users\\94266\\Desktop\\course\\ComputerNetWorking\\work\\lab3\\hw3-1\\test\\3.jpg . . . . .
传输文件成功
传输时间:595.08ms 平均吞吐率:157135kbps
请输入要发送的文件C:\\Users\\94266\\Desktop\\course\\ComputerNetWorking\\work\\lab3\\hw3-1\\test\\helloworld.txt
正在传输文件C:\\Users\\94266\\Desktop\\course\\ComputerNetWorking\\work\\lab3\\hw3-1\\test\\helloworld.txt . . . . .
传输文件成功
传输时间:87.3068ms 平均吞吐率:148167kbps
```

图 5: 发送端



```
C:\> Microsoft Visual Studio 调试控制台
请输入接收端的IP地址:127.0.0.1
请输入接收端的端口号:3245
请输入文件保存的位置:C:\\Users\\94266\\Desktop
开始接收文件1.jpg
接收文件成功
开始接收文件2.jpg
接收文件成功
开始接收文件3.jpg
接收文件成功
开始接收文件helloworld.txt
接收文件成功
```

图 6: 接收端