实验一 实验报告

题目：无线网络中隐藏站点仿真

一．题目背景

不同于通过接收传输信号来行使 CSMA/CD 载波侦听的功能的以太网络，无线网络的界线比较模糊，有时候并不是每个节点都可以跟其他节点直接通信。例如，可能出现这种情况：节点 2 可以直接跟节点 1 和节点 3 通信，不过某些因素导致节点 1 与节点 3 无法直接通信(这与障碍物的关系并不大，节点 1 与 3 之间可能只是因为距离远，无法收到对方的无线电波)。这样的情况下，从节点 1 的角度来看，节点 3 属于隐藏节点。如果使用简单的 transmit-and-pray 协议，节点 1 与节点 3 有可能在同一时间传送数据，这会造成节点 2 无法辨识任何信息。此外，节点 1 与节点 3 将无从得知错误发生，因为只有节点 2 才知道有冲突发生。

在无线网络中，上述由隐藏节点所导致的碰撞问题相当难以监听，因为无线收发器通常是半双工工作模式，即无法同时收发数据。为了防止碰撞发生，一种解决方法是允许工作站使用请求发送(RTS)和允许发送(CTS)帧来清空传送区域。如在上例中，节点 1 有个数据帧待传送，因此送出一个 RTS 帧启动整个过程。RTS 帧本身带有两个目的：预约无线链路的使用权，并要求接收到这一消息的其他的工作站停止发言。一旦收到 RTS 帧，接收端会以 CTS 帧应答。和 RTS 帧一样，CTS 帧也会令附近的工作站保持沉默。等到 RTS/CTS 完成交换过程，节点 1 即可传送上面要传送的帧，无须担心来自其他隐藏节点的干扰。

本次实验利用 NS2（NetworkSimulatorversion2）仿真上述无线网络中的隐藏站点问题。NS2 是由加州伯克利大学开发的面向对象的网络仿真器，本质上是一个离散事件模拟器。它本身有一个虚拟时钟，所有的仿真都由离散事件驱动。本次实验首先在 Linux(Centos)下配置 NS2 环境，然后进行隐藏站点问题仿真。

二．Linux 下 NS-2 环境配置

1.安装 gcc 和 g++

```
yum install gcc
```

```
yum install gcc-c++
```

2.安装依赖

```
yum install libX11-devel*
```

```
yum install xorg-x11-proto-devel*
```

```
yum install libXt-devel*
```

```
yum install libXmu-devel*
```

3. 下载 ns-allinone-2.35.tar.gz 并解压

```
wget http://sourceforge.net/projects/nsnam/files/allinone/ns-allinone-2.34/ns-allinone-2.34.tar.gz/download
```

```
tar -xzf download
```

4. 进入 ns2 源码目录，通过 ns-allinone 包安装 NS2 及所有依赖包

```
cd ns-allinone-2.34/

./install
```

5. 完成安装并得到如下提示：

```
Ns-allinone package has been installed successfully.
Here are the installation places:
tcl8.4.18:       /home/lijingyao/桌面/ns-allinone-2.34/{bin,include,lib}
tk8.4.18:               /home/lijingyao/桌面/ns-allinone-2.34/{bin,include,lib}
otcl:            /home/lijingyao/桌面/ns-allinone-2.34/otcl-1.13
tclcl:           /home/lijingyao/桌面/ns-allinone-2.34/tclcl-1.19
ns:              /home/lijingyao/桌面/ns-allinone-2.34/ns-2.34/ns
nam:     /home/lijingyao/桌面/ns-allinone-2.34/nam-1.14/nam
gt-itm:    /home/lijingyao/桌面/ns-allinone-2.34/itm, edriver, sgb2alt, sgb2ns, sgb2comns, sgb2hierns

---------------------------------------------------------------------------

Please put /home/lijingyao/桌面/ns-allinone-2.34/bin:/home/lijingyao/桌面/ns-allinone-2.34/tcl8.4.18/unix:/home/lijingyao/桌面/ns-allinone-2.34/tk8.4.18/unix
into your PATH environment; so that you'll be able to run itm/tclsh/wish/xgraph.

IMPORTANT NOTICES:

(1) You MUST put /home/lijingyao/桌面/ns-allinone-2.34/otcl-1.13, /home/lijingyao/桌面/ns-allinone-2.34/lib,
    into your LD_LIBRARY_PATH environment variable.
    If it complains about X libraries, add path to your X libraries
    into LD_LIBRARY_PATH.
    If you are using csh, you can set it like:
                setenv LD_LIBRARY_PATH <paths>
    If you are using sh, you can set it like:
                export LD_LIBRARY_PATH=<paths>

(2) You MUST put /home/lijingyao/桌面/ns-allinone-2.34/tcl8.4.18/library into your TCL_LIBRARY environmental
    variable. Otherwise ns/nam will complain during startup.


After these steps, you can now run the ns validation suite with
cd ns-2.34; ./validate

For trouble shooting, please first read ns problems page
http://www.isi.edu/nsnam/ns/ns-problems.html. Also search the ns mailing list archive
for related posts.

[root@hd-master ns-allinone-2.34]#
```

根据提示将提及的几个环境变量加入/root/.bashrc 中：

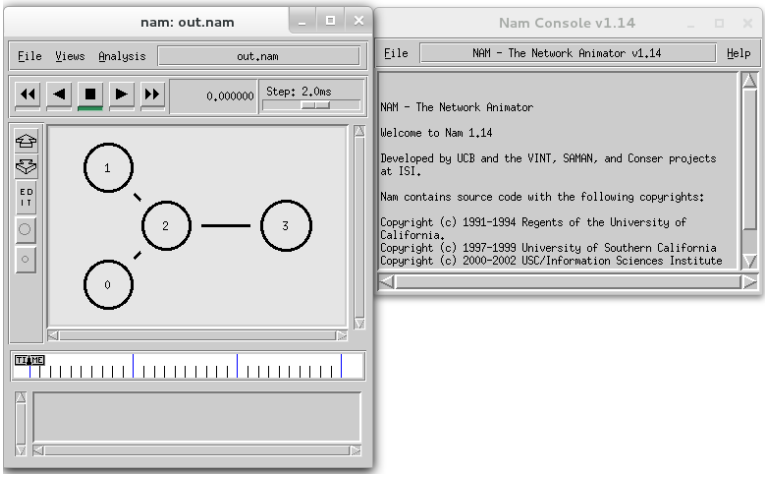
```
#NS2 Variable
export NS_HOME=/home/lijingyao/桌面/ns-allinone-2.34
export PATH=$NS_HOME/tcl8.4.18/unix:$NS_HOME/tk8.4.18/unix:$NS_HOME/bin:$PATH
export LD_LIBRARY_PATH=$NS_HOME/tcl8.4.18/unix:$NS_HOME/tk8.4.18/unix:$NS_HOME/otcl-1.13:$NS_HOME/lib:$LD_LIBRARY_PATH
export TCL_LIBRARY=$NS_HOME/tcl8.4.18/library
```

使更改生效：

```
source /root/.bashrc
```

以一个简单的例子测试 NS2 是否安装成功：

```
[root@hd-master 桌面]# cd ns-allinone-2.34/ns-2.34/tcl/ex
[root@hd-master ex]# ns simple.tcl
210
0.0037499999999999999
running nam...
[root@hd-master ex]#
```



NS2 环境配置成功！

三．实验过程

1. mUDP, mUdpSink 模块安装

UDP 是一种用于无连接传输的用户数据报协议，mUDP 模块是 UDP 的延伸，除了具有 UDP 的功能外，还能记录所发送的包的信息。mUdpSink 可以把接收到的包的信息记录到文件中。

（1）获取 mUDP, mUdpSink 的模块文件 mudp.cc，mudp.h，mudpsink.cc，mudpsink.h，在/ns-allinone-2.34/ns-2.34/下新建 measure 文件夹，把这四个文件放入其中；

（2）修改/ns-allinone-2.34/ns-2.34/common/packet.h 文件，增加如下内容：

```
// AOMDV patch
int aomdv_salvage_count_;

int frametype_;
double sendtime_;
unsigned int pkt_id_;
unsigned int frame_pkt_id_;

// called if pkt can't obtain media or isn't ack'd. not called if
// droped by a queue
FailureCallback xmit_failure_;
void *xmit_failure_data_;
```

（3）修改/ns-allinone-2.34/ns-2.34/Makefile 文件，增加如下内容：

```
    xcp/xcpq.o xcp/xcp.o xcp/xcp-end-sys.o \
    wpan/p802_15_4csmaca.o wpan/p802_15_4fail.o \
    wpan/p802_15_4hlist.o wpan/p802_15_4mac.o \
    wpan/p802_15_4nam.o wpan/p802_15_4phy.o \
    wpan/p802_15_4sscs.o wpan/p802_15_4timer.o \
    wpan/p802_15_4trace.o wpan/p802_15_4transac.o \
    apps/pbc.o \
    measure/mudp.o measure/mudpsink.o \
    $(OBJ_STL)
```

（4）修改/ns-allinone-2.34/ns-2.34/tcl/lib/ns-default.tcl 文件，增加如下内容：

```
Agent/PBC set payloadSize 200
Agent/PBC set periodicBroadcastInterval 1
Agent/PBC set periodicBroadcastVariance 0.1
Agent/PBC set modulationScheme_0
Agent/mUDP set packetSize_ 1000
-- 插入 --
```

（5）/ns-allinone-2.34/ns-2.34 目录下执行 make clean; make 命令，完成编译。

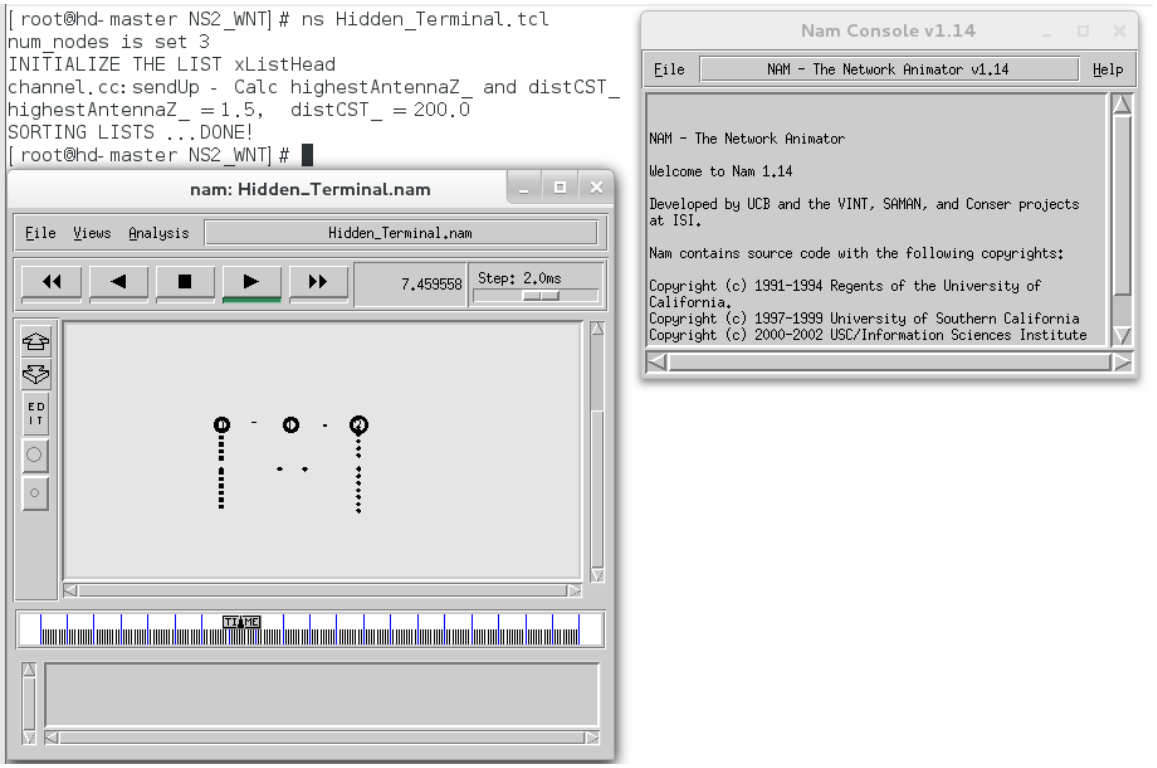
2. 隐藏站点问题仿真

（1）root 目录下新建一个 NS2_WNT 目录，用于放置运行的仿真脚本

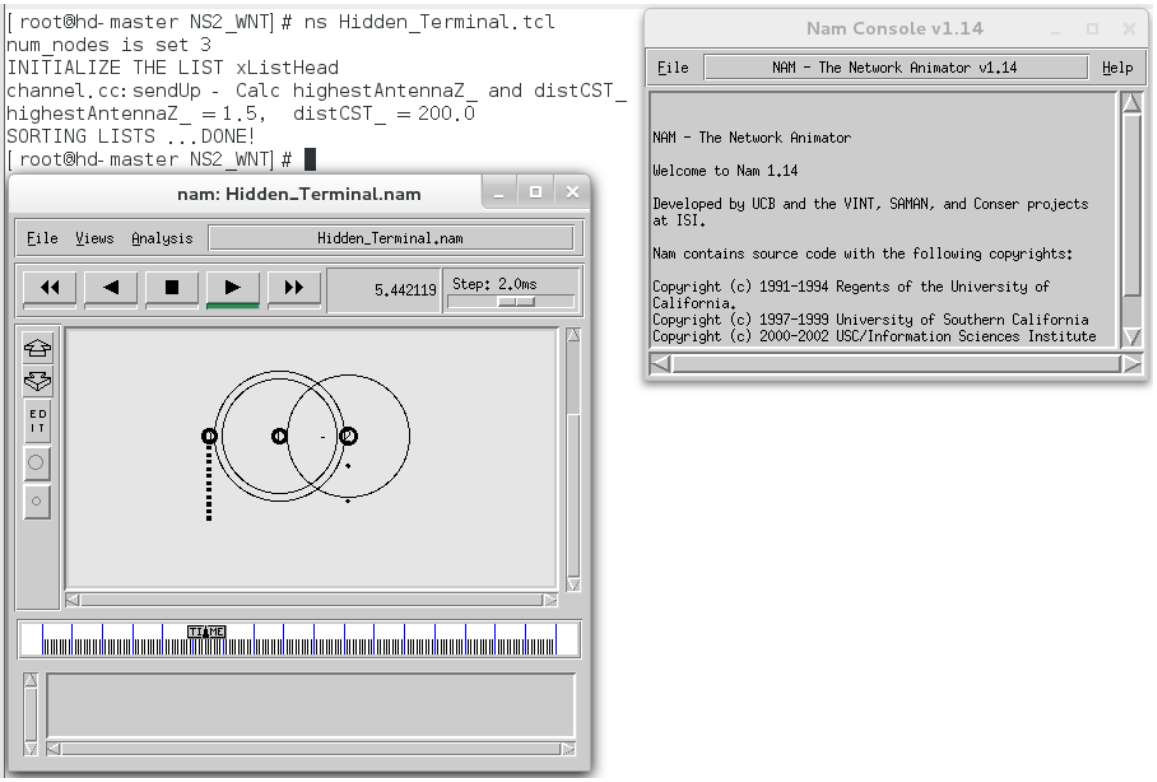
（2）编写隐藏站点问题仿真脚本 Hidden_Terminal.tcl，并放置在 NS2_WNT 目录下

（3）/root/NS2_WNT 目录下运行仿真 ns Hidden_Terminal.tcl，得到仿真结果如下：

设置 set RTSThreshold_　　3000 时的仿真结果：



设置 set RTSThreshold_ 0 时的仿真结果：



注意，在观看 NAM 动画时，运行的步长调整为 2ms 即可，不宜过大，以免无法观察到详细的仿真过程。

四．遇到的问题及解决办法

问题 1. 安装 NS2 时出现报错

```
tools/ranvar.cc: 在成员函数 'virtual double GammaRandomVariable::value()'中:
tools/ranvar.cc:219:70: 错误：不能直接调用构造函数 'GammaRandomVariable::GammaRandomVariable' [-fpermissive]
   return GammaRandomVariable::GammaRandomVariable(1.0 + alpha_, beta_).value() * pow (u, 1.0 / alpha_);

tools/ranvar.cc:219:70: 错误：对于函数类型的类型转换，移除冗余的 '::GammaRandomVariable' [-fpermissive]
make: *** [tools/ranvar.o] 错误 1
Ns make failed!
See http://www.isi.edu/nsnam/ns/ns-problems.html for problems
```

解决方法：

找到对应的 ns-allinone-2.34\ns-2.34\tools\ranvar.cc 文件，对第 219 行做如下更改：

将

```
return GammaRandomVariable::GammaRandomVariable(1.0 + alpha_, beta_).value() * pow (u, 1.0 / alpha_);
```

改为

```
return GammaRandomVariable(1.0 + alpha_, beta_).value() * pow (u, 1.0 / alpha_);
```

问题 2. 安装 NS2 时报错

```
In file included from mac/mac-802_11Ext.cc:66:0:
mac/mac-802_11Ext.h: 在成员函数 'u_int32_t PHY_MIBExt::getHdrLen11()'中:
mac/mac-802_11Ext.h:175:19: 错误：expected primary-expression before 'struct'
   return(offsetof(struct hdr_mac802_11, dh_body[0]))
                   ^
mac/mac-802_11Ext.h:175:41: 错误： 'dh_body'在此作用域中尚未声明
   return(offsetof(struct hdr_mac802_11, dh_body[0]))
                                         ^
mac/mac-802_11Ext.h:175:51: 错误： 'offsetof'在此作用域中尚未声明
   return(offsetof(struct hdr_mac802_11, dh_body[0]))
                                                 ^
mac/mac-802_11Ext.h:177:3: 警告：在有返回值的函数中，控制流程到达函数尾 [-Wreturn-type]
   }
   ^
make: *** [mac/mac-802_11Ext.o] 错误 1
Ns make failed!
See http://www.isi.edu/nsnam/ns/ns-problems.html for problems
```

解决方法：

找到对应的 ns-allinone-2.34\ns-2.34\mac\mac-802_11Ext.h，添加头文件#include <cstddef>

问题 3.安装 NS2 时报错

```
linkstate/ls.cc: 在成员函数 'int LsRetransmissionManager::ackIn(int, const LsMessage&)'中:
linkstate/ls.cc:447:50: 警告：建议在 '||'的操作数中出现的 '&&'前后加上括号 [-Wparentheses]
     (peerPtr->tpmSeq_ == LS_INVALID_MESSAGE_ID) &&
                                                 ^
In file included from linkstate/ls.cc:67:0:
linkstate/ls.h: In instantiation of 'void LsMap<Key, T>::eraseAll() [with Key = int; T = LsIdSeq]':
linkstate/ls.cc:396:28:   required from here
linkstate/ls.h:137:58: 错误： 'erase' was not declared in this scope, and no declarations were found by argument-dependent lookup at the point of instantiation [-fpermissive]
  void eraseAll() { erase(baseMap::begin(), baseMap::end()); }
                                                          ^
linkstate/ls.h:137:58: 附注：declarations in dependent base 'std::map<int, LsIdSeq, std::less<int>, std::allocator<std::pair<const int, LsIdSeq> > >' are not found by unqualified lookup
linkstate/ls.h:137:58: 附注：use 'this->erase' instead
make: *** [linkstate/ls.o] 错误 1
Ns make failed!
See http://www.isi.edu/nsnam/ns/ns-problems.html for problems
```

解决办法：

找到对应的 ns-allinone-2.34\ns-2.34\linkstate\ls.h

将

```
void eraseAll() { erase(baseMap::begin(), baseMap::end()); }
```

改为

```
void eraseAll() { this->erase(baseMap::begin(), baseMap::end()); }
```

五．实验结果分析及操作体会

1. 动画分析：

仿真开始时(0~1s)，节点之间相互广播路由(DSDV)信息，建立节点之间的路由表。从 1.5s 开始，节点 0 开始向节点 1 发送数据，需要注意，此时节点 0 需要先发送 RTS 报文(实验中设置 802.11 的 RTSThreshold=0，满足数据包大于 RTSThreshold，表示开启 RTS/CTS，解决隐藏终端问题)，以保证信道(0→1)的顺利占用。

在 2s 时，节点 2 也开始向节点 1 发送数据，同样的，节点 2 也发送 RTS 报文，此时可得知节点 1 的信道不空闲，因此节点 2 开始退避，等待一段时间重新尝试发送。在仿真过程(2~15s)中，可看到节点 2 和节点 0 出现交替的向节点 1 发送数据，两个节点没有同时传输。仿真时间到达 15s 时，两条流的传输同时结束，此后无数据传输，偶尔有链路保活报文。在 20s 时，仿真实验结束。

通过观察仿真动画，发现在没有 RTS/CTS 机制的情况下，因为 0 和 2 节点互相无法感知，他们同时向 1 发送数据的时候，由于互相冲突，两方的数据都不能送达，丢包率很高。这极大的降低了网络的吞吐量，增大了时延，这说明数据无线局域网隐藏节点问题对网络的传输影响较大。

在有 RTS/CTS 控制帧的网络环境中，CTS/RTS 机制则通过通知传送范围内的其他节点不要有动作，来避免发送冲突。仿真中首先 0 节点不断向 1 发送数据，1 节点接收；然后 2 节点不断向 1 发送，1 节点也能顺利接收。虽然 0 和 2 节点互相无法感知，但因为 CTS/RTS 机制，他们同时向 1 发送数据的时候，2 节点的数据都被 1 收到，0 节点则被通知不要动作。随着 2 节点数据发送减缓之后 0 节点送达的数据慢慢增多，而且相反的情况也会出现，即有时 0 节点的数据发送多的时候，2 节点的数据发送变少，这三种情况都有效提升了网络的吞吐量，减小了时延，这说明 RTS/CTS 机制较好的解决了隐藏节点问题，丢包率也明显降低。这也减轻了网络负担和压力，优化了网络环境。

2. 数据分析：

Trace 文件是 tcl 仿真过程中产生的结果文件，记录了仿真过程中每一个 packet 的传递情况。分析 tr 文件可以定量分析一段仿真过程的各种评测数值。

样例如图所示，其中各字段含义为：

字段 1： 封包事件发生的原因：

s -- sent；

r -- received；

d -- dropped；

f -- forward（转送）；

字段 2：事件发生的时间；

字段 3：事件发生的节点 ID（开始端）；

字段 4：说明这是发生在哪一层的事件（目的端）：

AGT -- 应用层；

RTR -- 路由层；

LL -- 链路层（在这里完成 ARP）；

IFQ -- 传出数据包队列（在链路层和媒体存取层之间）；

MAC -- 媒体存取层；

PHY -- 物理层；

字段 5：------ 分隔符

字段 6：封包的 ID；

字段 7：封包的类型：

cbr -- CBR 数据流包　　DSR -- DSR 路由包（路由生成的控制包）

RTS -- MAC 802.11 生成的 RTS 数据包　　ARP -- 链路层 ARP 报文

字段 8：封包的大小；

字段 9-12：[a b c d]

a -- mac 层标头中的数据包持续时间　　b --目的 mac 地址

c --来源 mac 地址　　d --数据包主体的 mac 类型

字段 13：--------- 分隔符

字段 14-17：[.....]

来源节点地址——（节点编号：端口号）

目的节点地址——（节点编号（-1 表示广播）：端口号）」

```
s 0.001635381 _0_ MAC --- 0 message 90 [0 ffffffff 0 800] ------- [0:255 -1:255 32 0]
r 0.002355881 _1_ MAC --- 0 message 32 [0 ffffffff 0 800] ------- [0:255 -1:255 32 0]
s 0.037009082 _2_ MAC --- 1 message 90 [0 ffffffff 2 800] ------- [2:255 -1:255 32 0]
r 0.037729582 _1_ MAC --- 1 message 32 [0 ffffffff 2 800] ------- [2:255 -1:255 32 0]
s 1.120739667 _1_ MAC --- 2 message 90 [0 ffffffff 1 800] ------- [1:255 -1:255 32 0]
r 1.121460167 _0_ MAC --- 2 message 32 [0 ffffffff 1 800] ------- [1:255 -1:255 32 0]
r 1.121460167 _2_ MAC --- 2 message 32 [0 ffffffff 1 800] ------- [1:255 -1:255 32 0]
s 1.500000000 _0_ AGT --- 3 cbr 1000 [0 0 0 0] ------- [0:0 1:0 32 0] [0] 0 0
s 1.500415000 _0_ MAC --- 0 ARP 86 [0 ffffffff 0 806] ------- [REQUEST 0/0 0/1]
r 1.501103500 _1_ MAC --- 0 ARP 28 [0 ffffffff 0 806] ------- [REQUEST 0/0 0/1]
s 1.501318500 _1_ MAC --- 0 RTS 44 [52e 0 1 0]
r 1.501671000 _0_ MAC --- 0 RTS 44 [52e 0 1 0]
s 1.501681000 _0_ MAC --- 0 CTS 38 [3f4 1 0 0]
r 1.501985500 _1_ MAC --- 0 CTS 38 [3f4 1 0 0]
s 1.501995500 _1_ MAC --- 0 ARP 86 [13a 0 1 806] ------- [REPLY 1/1 0/0]
r 1.502684000 _0_ MAC --- 0 ARP 28 [13a 0 1 806] ------- [REPLY 1/1 0/0]
s 1.502694000 _0_ MAC --- 0 ACK 38 [0 1 0 0]
r 1.502998500 _1_ MAC --- 0 ACK 38 [0 1 0 0]
s 1.503128000 _0_ MAC --- 0 RTS 44 [242e 1 0 0]
r 1.503480500 _1_ MAC --- 0 RTS 44 [242e 1 0 0]
s 1.503490500 _1_ MAC --- 0 CTS 38 [22f4 0 0 0]
r 1.503795000 _0_ MAC --- 0 CTS 38 [22f4 0 0 0]
s 1.503805000 _0_ MAC --- 3 cbr 1078 [13a 1 0 800] ------- [0:0 1:0 32 1] [0] 0 0
s 1.508000000 _0_ AGT --- 4 cbr 1000 [0 0 0 0] ------- [0:0 1:0 32 0] [1] 0 0
r 1.512429500 _1_ MAC --- 3 cbr 1020 [13a 1 0 800] ------- [0:0 1:0 32 1] [0] 1 0
s 1.512439500 _1_ MAC --- 0 ACK 38 [0 0 0 0]
r 1.512454500 _1_ AGT --- 3 cbr 1020 [13a 1 0 800] ------- [0:0 1:0 32 1] [0] 1 0
r 1.512744000 _0_ MAC --- 0 ACK 38 [0 0 0 0]
```

下面利用 python 来计算如下指标，具体代码请见附录。

（1）平均时延

平均时延的计算公式为：$$avg\_delay = \frac{\sum(receive\_time - send\_time)}{receives}$$

结果如下图所示，可以看到有 RTS/CTS 机制的平均时延比没有该机制时的平均时延降低了 61.69%，所以这说明了 RTS/CTS 机制确实能显著降低时延。

```
has rts/cts average delay: 1.7050174790163912
no rts/cts average delay: 4.4513845368852465
```

（2）丢包率

丢包率的计算公式为：$$drop\_rate = \frac{drop\_num}{send\_num}$$

结果如下图所示，可以看到有 RTS/CTS 机制的丢包率是 0，而没有该机制时的丢包率高达 91.41%，所以这说明了 RTS/CTS 机制确实能有效解决隐藏节点带来的丢包问题。

```
has rts/cts drop rate: 0.0
no rts/cts drop rate: 0.9140652402665731
```

（3）吞吐量

吞吐量的计算公式为：$$throughput = \frac{tol\_packetsize}{tol\_time}$$

结果如下图所示，可以看到有 RTS/CTS 机制的吞吐量是没有该机制时吞吐量的 7.46 倍，所以这说明了 RTS/CTS 机制确实能显著提升吞吐量。

```
has rts/cts throughput rate: 100492.86079141665
no rts/cts throughput rate: 13476.632476101355
```

3. 操作体会

仿真过程生动地展现了数据包传输和丢包情况，包括频率和数量。而数据分析过程也让我充分理解了 trace 文件的含义和信息的丰富程度。这二者都让我们深刻理解了隐藏节点的问题严重性和 RTS/CTS 解决方案的有效性，这让我觉得这个实验极有意义。

附录：数据分析代码 trace_analysis.py

```python
import pandas as pd
import numpy as np


# 输入dataframe格式的数据，返回array
def delay(data):
    send_data = data[(data['3']=='AGT') & (data['7']=='cbr') & (data['0']=='s') & (data['2'].isin(['_0_', '_2_']))].copy().sort_values(by=['6'])
    receive_data = data[(data['3']=='AGT') & (data['7']=='cbr') & (data['0']=='r') & (data['2']=='_1_')].copy().sort_values(by=['6'])
    target_id = receive_data['6']
    target_send_data = send_data[send_data['6'].isin(target_id)].copy()
    receive_time = np.array(receive_data['1'])
    send_time = np.array(target_send_data['1'])
    duration_time = receive_time - send_time
    return duration_time

# 输入dataframe格式的数据，返回float
def drop_rate(data):
    send_data = data[(data['3']=='MAC') & (data['7']=='cbr') & (data['0']=='s') & (data['2'].isin(['_0_', '_2_']))].copy().sort_values(by=['6'])
    drop_data = data[(data['3']=='MAC') & (data['7']=='cbr') & (data['0']=='D') & (data['2']=='_1_')].copy().sort_values(by=['6'])
    return len(drop_data)/len(send_data)

# 输入dataframe格式的数据，返回float
def throughput(data):
    send_data = data[(data['3']=='AGT') & (data['7']=='cbr') & (data['0']=='s') & (data['2'].isin(['_0_', '_2_']))].copy().sort_values(by=['6'])
    receive_data = data[(data['3']=='AGT') & (data['7']=='cbr') & (data['0']=='r') & (data['2']=='_1_')].copy().sort_values(by=['6'])
    start_time = send_data['1'].min()
    end_time = receive_data['1'].max()
    total_size = receive_data['8'].sum()
    return total_size/(end_time-start_time)
```

```python
if __name__ == "__main__":
    column_names = []
    for i in range(19):
        column_names.append(str(i))
    data = pd.read_csv("Hidden_Terminal_0.tr", sep=' ', header=None, names = column_names)
    print(data.head())
    duration_time = delay(data)
    avg_delay = sum(duration_time)/len(duration_time)
    drop_rates = drop_rate(data)
    throughput_rate = throughput(data)
    print("has rts/cts average delay:", avg_delay)
    print("has rts/cts drop rate:", drop_rates)
    print("has rts/cts throughput rate:", throughput_rate)
    data = pd.read_csv("Hidden_Terminal_3000.tr", sep=' ', header=None, names = column_names)
    duration_time = delay(data)
    avg_delay = sum(duration_time)/len(duration_time)
    drop_rates = drop_rate(data)
    throughput_rate = throughput(data)
    print("no rts/cts average delay:", avg_delay)
    print("no rts/cts drop rate:", drop_rates)
    print("no rts/cts throughput rate:", throughput_rate)
```