### 1、ns软件cbr数据包大小，默认发送的最大字节数为1000

使用文件的位置：/usr/ns-allinone-2.34/ns-2.34/link/delay.cc

使用的函数：void UdpAgent::sendmsg(int nbytes, AppData\* data, const char\* flags)

其中nbytes是用户传经来的发包速率，函数中的size\_时候程序初始化绑定配置文件中的默认值

UdpAgent::UdpAgent(packet\_t type) : Agent(type)

{

bind("packetSize\_", &size\_);

}

使用位置：81行，n = nbytes / size\_;

修改文件位置：/usr/ns-allinone-2.34/ns-2.34/tcl/lib/ns-default.tcl

修改位置：864

修改内容：原来是Agent/UDP set packetSize\_ 1000，修改为Agent/UDP set packetSize\_ 1472字节

### 2、ll层事件调度时延调整（数据链路层）

使用文件的位置：/usr/ns-allinone-2.34/ns-2.34/mac/ll.cc

使用的函数：void LL::sendDown(Packet\* p)

使用位置：252行s.schedule(downtarget\_, p, delay\_);设置mac的时延

修改文件位置：/usr/ns-allinone-2.34/ns-2.34/tcl/lib/ns-default.tcl

修改位置：674行

修改内容：原来是LL set delay\_ 25us，修改为LL set delay\_ 2us

### 3、MAC层数据包发送时延计算

#### 修改1：

使用文件的位置：/usr/ns-allinone-2.34/ns-2.34/mac/mac-802\_11.cc

使用的函数：double Mac802\_11::txtime(double psz, double drt)

使用位置：1919行

double t = (((double)plcp\_hdr)/phymib\_.getPLCPDataRate()) + (((double)datalen)/drt);

修改文件位置：/usr/ns-allinone-2.34/ns-2.34/tcl/lib/ns-default.tcl

修改位置：693行

修改内容：

原来是：

Mac/802\_11 set PLCPDataRate\_ 1.0e6

修改为：

Mac/802\_11 set PLCPDataRate\_ 1.0e8

#### 修改2：

使用文件的位置：/usr/ns-allinone-2.34/ns-2.34/mac/mac-802\_11.cc

使用的函数：double Mac802\_11::txtime(double psz, double drt)

使用位置：1919行

double t = (((double)plcp\_hdr)/phymib\_.getPLCPDataRate()) + (((double)datalen)/drt);

修改文件位置：/usr/ns-allinone-2.34/ns-2.34/tcl/lan/ns-mac.tcl

修改位置：43行、64行、65行

修改内容：

原来是：

Mac set bandwidth\_ 1Mb

Mac/802\_11 set basicRate\_ 1Mb

Mac/802\_11 set dataRate\_ 1Mb

修改为：

Mac set bandwidth\_ 100Mb

Mac/802\_11 set basicRate\_ 100Mb

Mac/802\_11 set dataRate\_ 100Mb

### 4、MAC层接收数据的时间计算

#### 修改1

使用文件的位置：/usr/ns-allinone-2.34/ns-2.34/mac/mac-802\_11.cc

使用的函数：int Mac802\_11::check\_pktCTRL()等

使用位置：983行

timeout = txtime(phymib\_.getCTSlen(), basicRate\_)

+ DSSS\_MaxPropagationDelay // XXX

+ sec(mh->dh\_duration)

+ DSSS\_MaxPropagationDelay // XXX

- phymib\_.getSIFS()

- txtime(phymib\_.getACKlen(), basicRate\_);

修改文件位置：/usr/ns-allinone-2.34/ns-2.34/mac/mac-802\_11.h

修改位置：227行

修改内容：

原来是：

#define DSSS\_MaxPropagationDelay 0.000002

修改为：

#define DSSS\_MaxPropagationDelay 0.00000002

#### 修改2：

使用文件的位置：/usr/ns-allinone-2.34/ns-2.34/mac/mac-802\_11.cc

使用的函数：int Mac802\_11::check\_pktCTRL()等

使用位置：983行

timeout = txtime(phymib\_.getCTSlen(), basicRate\_)

+ DSSS\_MaxPropagationDelay

+ sec(mh->dh\_duration)

+ DSSS\_MaxPropagationDelay

- phymib\_.getSIFS()

- txtime(phymib\_.getACKlen(), basicRate\_);

修改文件位置：/usr/ns-allinone-2.34/ns-2.34/tcl/lib/ns-default.tcl

修改位置：715行

修改内容：

原来是：

Mac/802\_11 set SIFS\_ 0.000010

修改为：

Mac /802\_11 set SIFS\_ 0.00000010

5、MAC层信道忙状态退避时延计算

使用文件的位置：/usr/ns-allinone-2.34/ns-2.34/mac/mac-802\_11.cc

使用的函数：Mac802\_11::tx\_resume()等

使用位置：632

mhBackoff\_.start(cw\_, is\_idle(),

phymib\_.getDIFS());

修改文件位置：/usr/ns-allinone-2.34/ns-2.34/tcl/lib/ns-default.tcl

修改位置：715行

修改内容：

原来是：

Mac/802\_11 set SlotTime\_ 0.000020

Mac/802\_11 set SIFS\_ 0.000010

修改为：

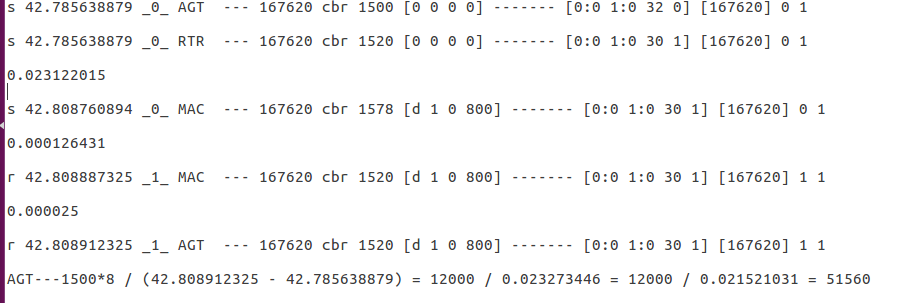
Mac/802\_11 set SlotTime\_ 0.00000020

Mac/802\_11 set SIFS\_ 0.00000010

### 5、trace文件分析注意问题

默认的trace文件没有只包含AGT（应用层）、RTR（网络层）、MAC层，而在传递过程中的LL层（数据链路层）以及LinkDelay（有线网络仿真使用的数据链路层）没有相应的日志记录，

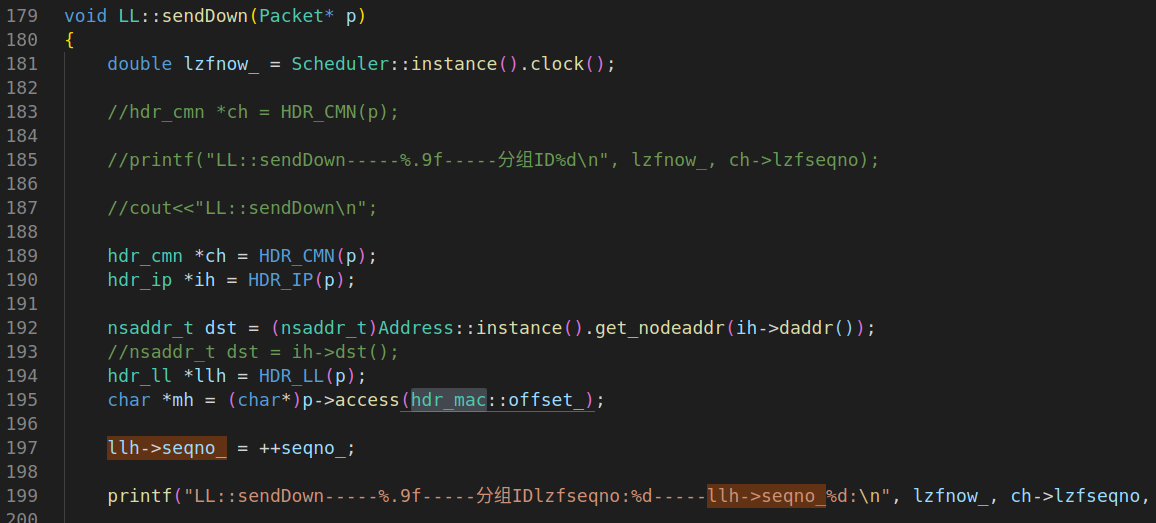
如果想在对应的层想要获得对应的信息，第一种方法修改源码的trace文件的源码，第二种利用printf函数将结果打印出来，输出到日志文件，然后进行分析



这是167620号数据包的发送到接收的整个trace记录

### 6、如何在LL层获得数据包的到达和发送时间

double lzfnow\_ = Scheduler::instance().clock();



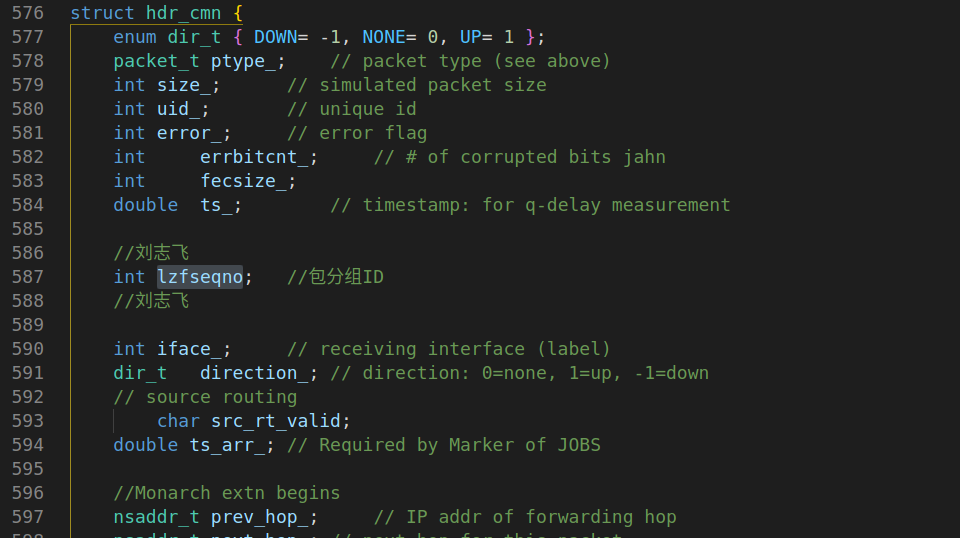
### 7、自己打印出来的数据有可能时乱象的

所以在打印数据包的相关信息，应带带上数据包的序号，但是有可能在某一层无法获取，该包的序号，所以我们可以自己在包内添加一个自己使用的变量，用来存储包序号，

添加方法：

在/usr/ns-allinone-2.34/ns-2.34/common/packet.h中的packet.h的struct hdr\_cmn结构体中添加lzfseqno变量，这样在LL层或者队列层就可以利用

struct hdr\_cmn \*hdr = HDR\_CMN(p); hdr->lzfseqno获得包的序号



### 8、吞吐量的计算方法

我们一般统计的是平均吞吐量，测试的cbr业务流量使用的就是UDP，总的发送比特数处理第一包的发送到最后一包接收的时间间隔，

统计的时候使用gawk脚本对trace文件进行统计，trace文件的第一列代表事件的类型，第二列就是该事件发生的时间，第四列就是数据所在的层次。

if ( event == "s" && layer == "AGT" && pkt\_type == "cbr"){

if(recvnum == 0){

printf("开始时间是%d %.2f\n", recvnum, time);

}

}

if ( event == "r" && layer == "AGT" && pkt\_type == "cbr")

{

if(recvnum == 0)

{

start\_time=time;

#printf("第一次接收时间%d %.2f\n", recvnum, start\_time);

}

endtime[recvnum] = time;

#printf("%d %.2f\n", recvnum, time);

recvnum++;

}

### 9一些结构体变量的初始化

如果遇到bind("Prefer\_Routing\_Protocols", &Prefer\_Routing\_Protocols);这样的初始化语句，想要修改它的初始化值，一般是在ns-mac.tcl和ns-default.tcl文件中，在文件搜索前面的名字，文件的位置/usr/ns-allinone-2.34/ns-2.34/tcl/lib/ns-default.tcl和

/usr/ns-allinone-2.34/ns-2.34/tcl/lib/ns-default.tcl

### 10、CBR业务整体的发包流程

1、cbr业务流的c++代码定时发送信号给UDP数据包发送代理，UDP数据包发送代理接收到信号以后，内存空间生成一个分组，然后UDP代理将这个分组交到下一层，调用的就是apps/udp.cc中的sendmsg函数中的target\_->recv(p)；其中target\_代表下一层的对象指针，实际上时CMUTrace类的一个对象

2、CMUTrace类的作用就是对无线网络的分组进行跟踪和记录，并将该记录写入模拟过程中生成的“.tr”文件，然后在其recv函数中调用send(p, h)，将分组交付给Classifier类的recv函数

3、Classifier类调用Connect类的send函数将分组传递给路由协议的recv函数（这里使用ADOV进行分析）

4、AODV类会在forward函数中执行Scheduler::instance().schedule(target\_, p, delay);将分组加入调度器等待调度，当调度时刻到达时，将分组发送到CMUTrace类的recv函数，记录结束后，再调用Connect类的send函数将分组传递给LL层的recv函数

5、LL层做完地址解析后，就在void LL::sendDown(Packet\* p)函数中调用

s.schedule(downtarget\_, p, delay\_);将分组发送给队列实例

6、队列一般使用的是PriQueue队列，数据包被recv函数接收，然后该函数有又调用

recvHighPriority函数接收分组，然后调用target\_->recv(p, &qh\_);将分组发送到CMUTrace类的recv函数，记录结束后，再调用Connect类的send函数将分组传递给MAC层的recv函数

7、MAC层（这里使用mac-802\_11.cc进行分析）的recv函数调用Mac802\_11::transmit(Packet \*p, double timeout)函数接收分组，然后调用downtarget\_->recv(p->copy(), this); 将分组发送到CMUTrace类的recv函数，记录结束后，再调用Connect类的send函数将分组传递给phy层的recv函数

8、phy层的recv函数调用sendDown(p);在WirelessPhy::sendDown(Packet \*p)中调用channel\_->recv(p, this);将分组发送到WirelessChannel类的recv函数，

9、WirelessChannel类的recv函数在一定的延时后调用邻居节点的WirelessChannel::sendUp(Packet\* p, Phy \*tifp)用来接收分组，实现无线信道的分组拷贝

### 11结果展示

修改之后的

14000 11215.35

19000 15214.90

24000 19215.13

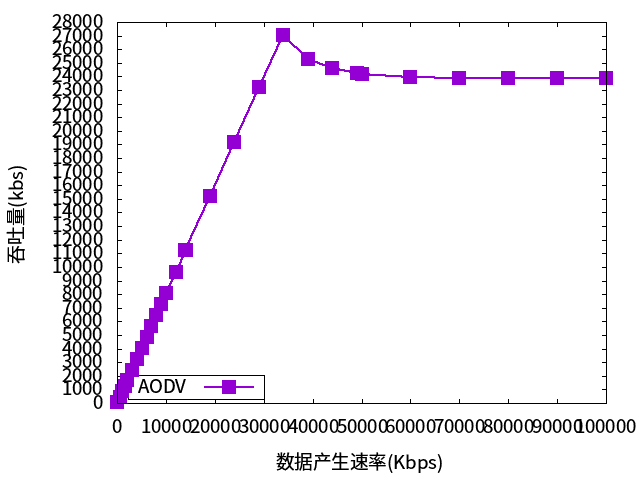
29000 23215.33

34000 27037.66

39000 25291.02

44000 24590.30

49000 24246.17



未修改的

50 50.12

75 75.15

100 100.21

125 125.22

150 150.25

175 153.48

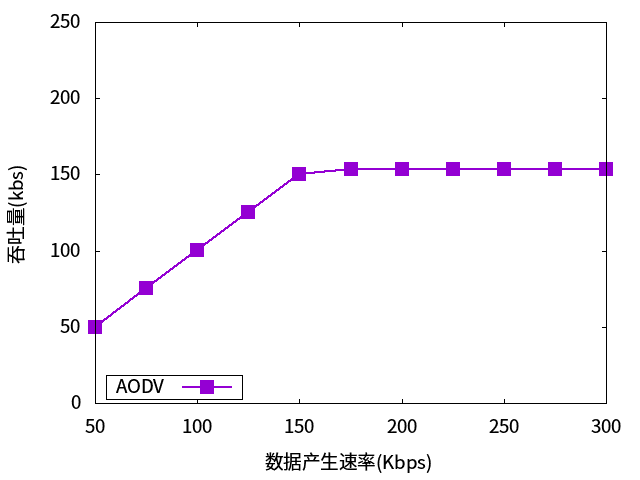
200 153.53

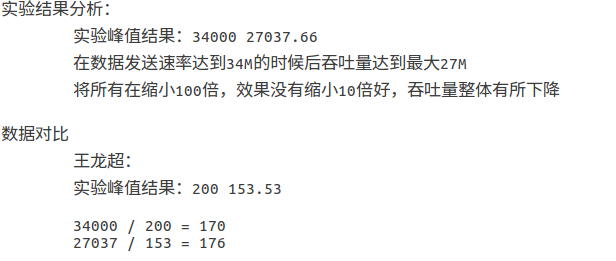
225 153.53

250 153.48

275 153.48

300 153.53





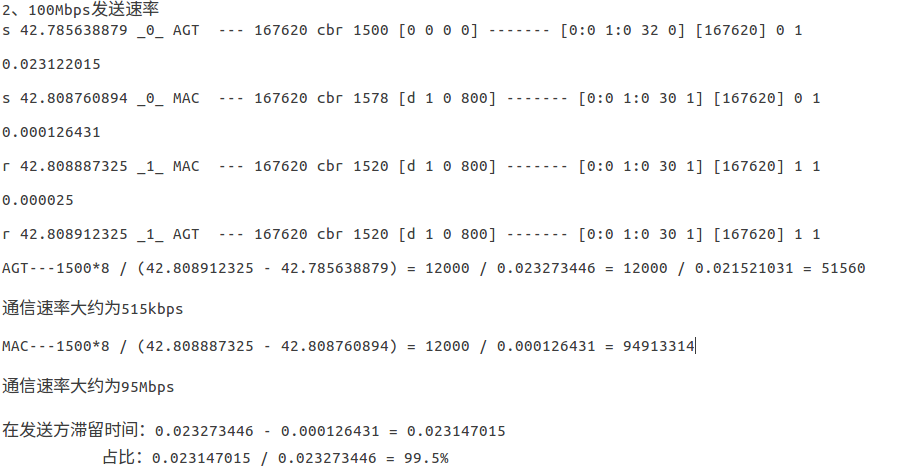
实验峰值结果：200 153.53

34000 / 200 = 170

27037 / 153 = 176

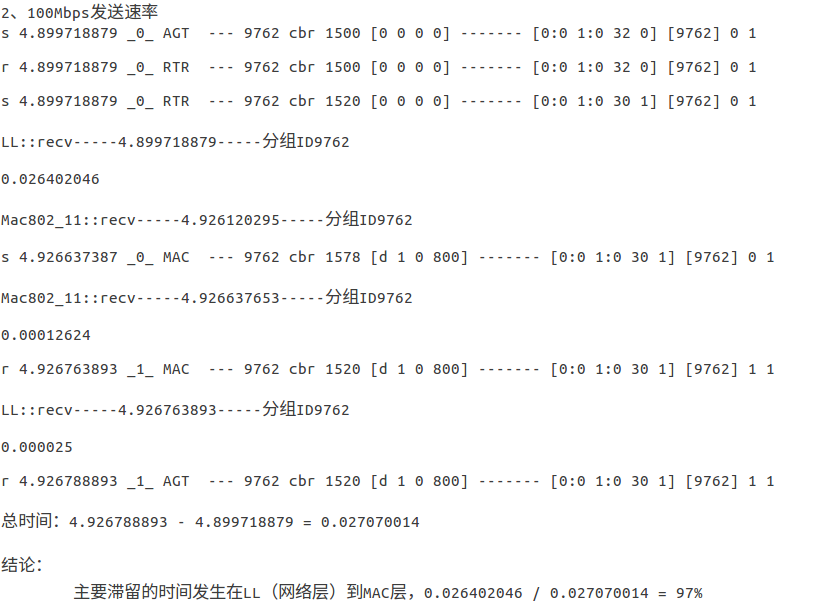
### 12研究过程分析

1、首先使用两个节点测试100M分析通信时延



通过分析trace文件的实验数据，可以知道从源节点的MAC到目的节点的MAC层的通信速率基本可以达到100M，但是从源节点的应用层的通信速率却远远低于100M。

2、具体分析TCP/IP协议栈中每一层上下交付过程中的延时，由于trace文件无法统计TCP/IP协议栈中数据包上下交付过程中更加详细的信息，所以为了研究每一层交付过程中的时延，在数据包的MAC头中添加数据分组的编号和交付给下一层的当前时间



有图中计算信息可以得到，TCP/IP协议栈中LL层到MAC层所占用的时间最多，这是因为LL层并不是直接将数据包交付给MAC层，而是将数据包放在一个延时队列当中，当延时时间达到的时候，MAC层才可以得到这个数据分组，从而进行接下来的工作；而这个延时时间就和通信带宽有这密切的关系，大致是这样一个关系，通信带宽越大，所需的延时时间就越小，延时时间主要是为了防止上一个数组分组没有发送结束。