NS-3网络仿真

一：作业要求

用 NS-3仿真某个特定的网络环境，并输出相应的仿真参数（队列拥塞程度，时延，吞吐量等）。

二：软件介绍

NS-3 是一款离散事件网络模拟驱动器，操作者能够由 C++编写自己所需要的网络拓扑以及网络环境，来模拟一个网络的数据传输，并输出其性能参数。软件中包含很多模块：节点模块（创造节点），网络模块（不同的通信协议），随机模块（生成随机错误模型），应用模块（创建 packet 数据包以及接受 packet 数据包），统计模块（输出统计数据，网络性能参数），移动模块（仿真 WIFI，LTE 可使用），等等。

三：实验原理及步骤

1. NS-3 的下载安装：
   参考 https://www.nsnam.org/wiki/Installation#Installation_with_Bake

   这里强烈推荐使用 bake 进行下载安装：
   （1）安装几个前导包：

```
sudo apt-get install gcc g++ Python
sudo apt-get install mercurial
sudo apt-get install bzr
sudo apt-get install gdb valgrind
sudo apt-get install gsl-bin libgsl0-dev libgsl0ldbl
sudo apt-get install flex bison
sudo apt-get install g++-3.4 gcc-3.4
sudo apt-get install tcpdump
sudo apt-get install sqlite sqlite3 libsqlite3-dev
sudo apt-get install libxml2 libxml2-dev
sudo apt-get install libgtk2.0-0 libgtk2.0-dev
```

sudo apt-get install vtun lxc
sudo apt-get install uncrustify
sudo apt-get install doxygen graphviz imagemagick
sudo apt-get install texlive texlive-pdf texlive-latex-extra texlive-generic-extra texlive-generic-recommended
sudo apt-get install texinfo dia texlive texlive-pdf texlive-latex-extra texlive-extra-utils texlive-generic-recommended
sudo apt-get install python-pygraphviz python-kiwi python-pygoocanvas libgoocanvas-dev
sudo apt-get install libboost-signal-dev libboost-filesystem-dev

提示：这里面不是所有的包都能安装成功的，个人使用的时候：gcc g++ python，libgsloldbl，两个 textlive-pdf 以及最后一个 libboost 包都没有安装成功，对后面的使用没有任何影响。

（2） 使用 bake 进行下载安装，按顺序输入指令：

hg clone http://code.nsnam.org/bake
export BAKE_HOME=`pwd`/bake
export PATH=$PATH:$BAKE_HOME
export PYTHONPATH=$PYTHONPATH:$BAKE_HOME

到这一步，检查一下之前预装的包是不是都在：运行：bake.py check

你可能会看到：

```
> Python - OK
   > GNU C++ compiler - OK
   > Mercurial - OK
   > CVS - OK
   > GIT - OK
   > Bazaar - OK
   > Tar tool - OK
   > Unzip tool - OK
   > Unrar tool - OK
   > 7z  data compression utility - OK
   > XZ data compression utility - OK
   > Make - OK
   > cMake - OK
   > patch tool - OK
   > autoreconf tool - OK
   > Path searched for tools: /usr/lib64/qt-3.3/bin
 /usr/lib64/ccache /usr/local/bin  /usr/bin/bin/usr/local/sbin /usr/sbin
 /sbin /user/dcamara/home/scripts/user/dcamara/home/INRIA/Programs/bin
 /user/dcamara/home/INRIA/repos/llvm/build/Debug+Asserts/bin
```

之后就可以做下面的两步：

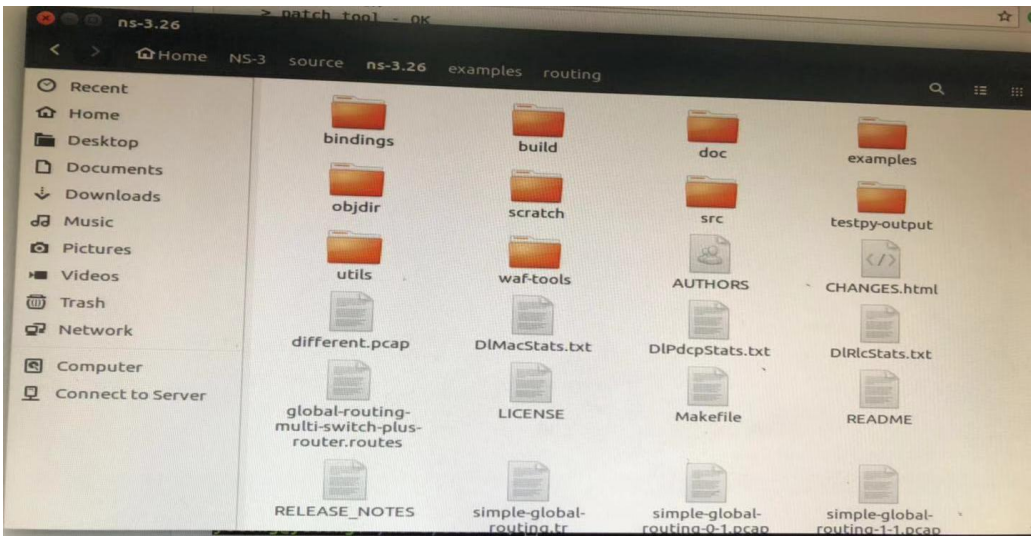
bake.py configure -e ns-3.26
bake.py deploy （这一步要等待很久很久大概一小时左右，这一步做完 ns-3 就装好了）

（3） 编译 ns-3

先进入到 ns-3.26 的文件夹下（我的是在~/NS-3/source/ns-3.26）

./waf configure --build-profile=debug --enable-examples --enable-tests
./waf build

编译好之后的文件夹显示为：



./test.py -c core(测试一下是否成功编译)
./waf --run hello-simulator（跑一个 hello 自带代码）
如果输出为：
Hello Simulator 则安装编译成功，可以使用了。

2. 示例代码讲解入门

首先假设一个简单的网络拓扑：两个节点之间使用点对点链路，使用 TCP 协议进行通信，假设随机错误率为0.00001，节点不可移动（因为不是无线网络），具体代码如下：

NodeContainer nodes;

nodes.Create (2);

创建两个节点；

PointToPointHelper pointToPoint;

pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));

pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

设置链路的传输速率为5Mbps，时延为2ms；

NetDeviceContainer devices;

devices = pointToPoint.Install (nodes);

为每个节点添加网络设备

Ptr<RateErrorModel>em=CreateObject<RateErrorModel> ();

em->SetAttribute("ErrorRate",DoubleValue(0.00001));

devices.Get(1)->SetAttribute("ReceiveErrorModel",PointerValue (em));

创建一个错误模型，讲错误率设置为0.00001，仿真 TCP 协议的重传机制。

InternetStackHelper stack;

stack.Install (nodes);

为每个节点安装协议栈;

Ipv4AddressHelper address;

address.SetBase ("10.1.1.0", "255.255.255.252");

Ipv4InterfaceContainer interfaces = address.Assign (devices);

为每个节点的网络设备添加 IP 地址;

这样一个简单的网络拓扑就建立完成。

接下来就是为这个网络节点添加应用程序，让他们在这个网络中模拟传输数据，具体代码如下:

uint16_t sinkPort = 8080;

Address sinkAddress (InetSocketAddress (interfaces.GetAddress (1), sinkPort));

PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), sinkPort));

ApplicationContainer sinkApps = packetSinkHelper.Install (nodes.Get (1));

sinkApps.Start (Seconds (0.));

sinkApps.Stop (Seconds (10.));

将接受数据的应用程序设置在 Node.Get(1)节点上，端口设置为8080；程序起始时间为0s，终止时间为10s;

Ptr<MyApp> app = CreateObject<MyApp> ();

app->Setup (ns3TcpSocket, sinkAddress, 1040, 1000, DataRate ("1Mbps"));

nodes.Get (0)->AddApplication (app);

app->SetStartTime (Seconds (1.));

app->SetStopTime (Seconds (10.));

将发送数据的应用程序设置在 Node.Get(0)；发送起始时间为1s；结束时间为10s;

这样网络拓扑和节点之间应用程序的设定已完成，接下来就是应用统计模块，输出节点之间具体通信性能的参数，及时延，吞吐量，抖动率，丢包率;

NS-3中，有一个回调机制，方便我们来输出具体某个条件发生改变时就自动执行某个函数，回调的实现是 TraceConnectWithoutContext 函数，举个例子，在我的时延仿真中，输出时延的代码是这样写的:

```
static void
CalculateDelay (Ptr<const Packet>p,const Address &address)
{
   static float k = 0;
   k++;
   static float m = -1;
   static float n = 0;
   n += (p->GetUid() - m)/2-1;
   delayJitter.RecordRx(p);
   Time t = delayJitter.GetLastDelay();
   std::cout << Simulator::Now ().GetSeconds () << "\t" << t.GetMilliSeconds() <<
std::endl;
   m = p->GetUid();
}
```

首先定义一个时延的计算函数，是全局变量函数；

其次在 main 函数中使用回调机制：

```
sinkApps.Get(0)->TraceConnectWithoutContext("Rx",
MakeCallback(&CalculateDelay));
```

含义就是当接受端节点每收到一个 TCP 包，就会执行一次 CalculateDelay 函数，计算这个数据包在网络中传输的时延，并输出；

这样就完成了程序的编写；接下来就是输出具体数据：

在终端打开，到指定的文件夹中，输入

```
./waf --run scratch/delay >delay.dat 2>&1
```

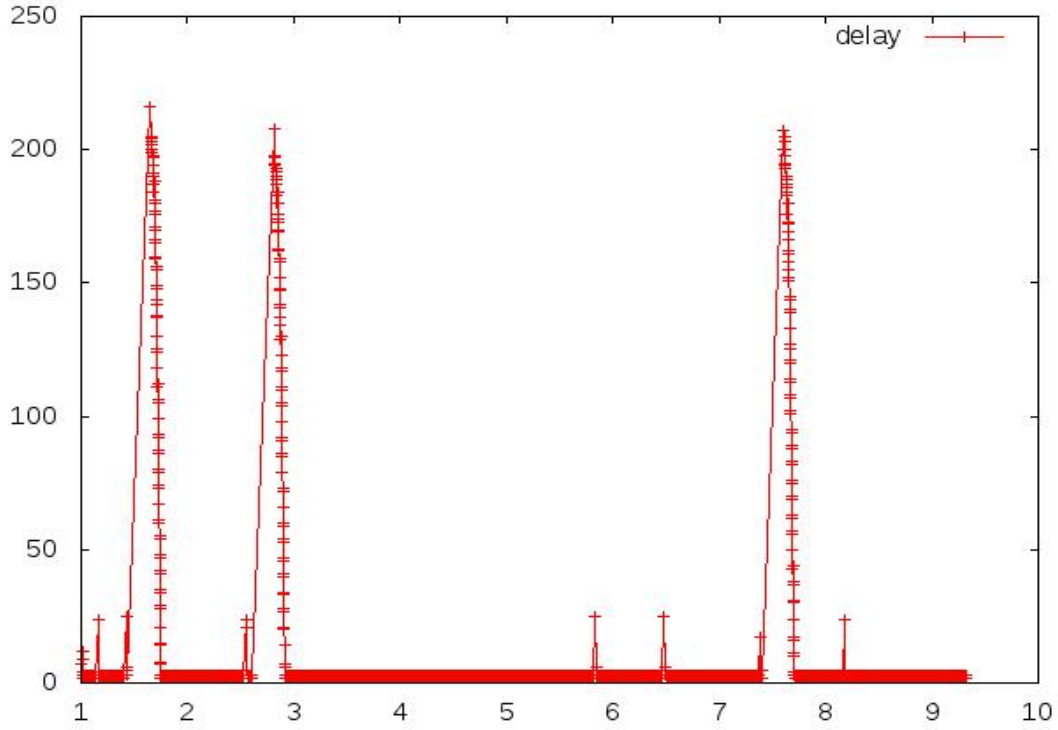
按指定格式输出.dat 文件之后，再在终端用 GNUPLOT 来作出.dat 文件中的图形即可（GNUPLOT 的使用可以参考 dsec.pku.edu.cn/~tanghz/gnuplot.htm）：当然不画图，用 print 输出，或者 log 日志的形式也是可以的（如 cwnd website）。

Linux安装

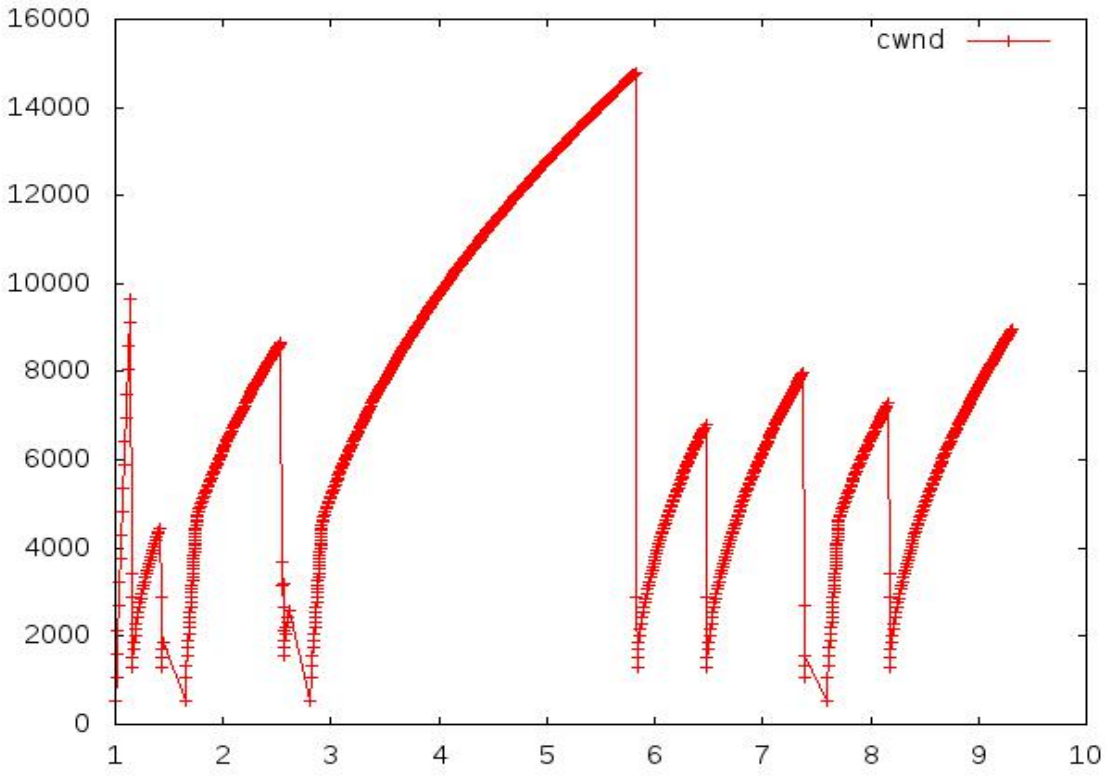
终端输入命令 **$ sudo apt-get install gnuplot** 系统自动获取包信息、处理依赖关系，完成安装

安装完毕后，在终端运行命令 **$ gnuplot** 进入gnuplot

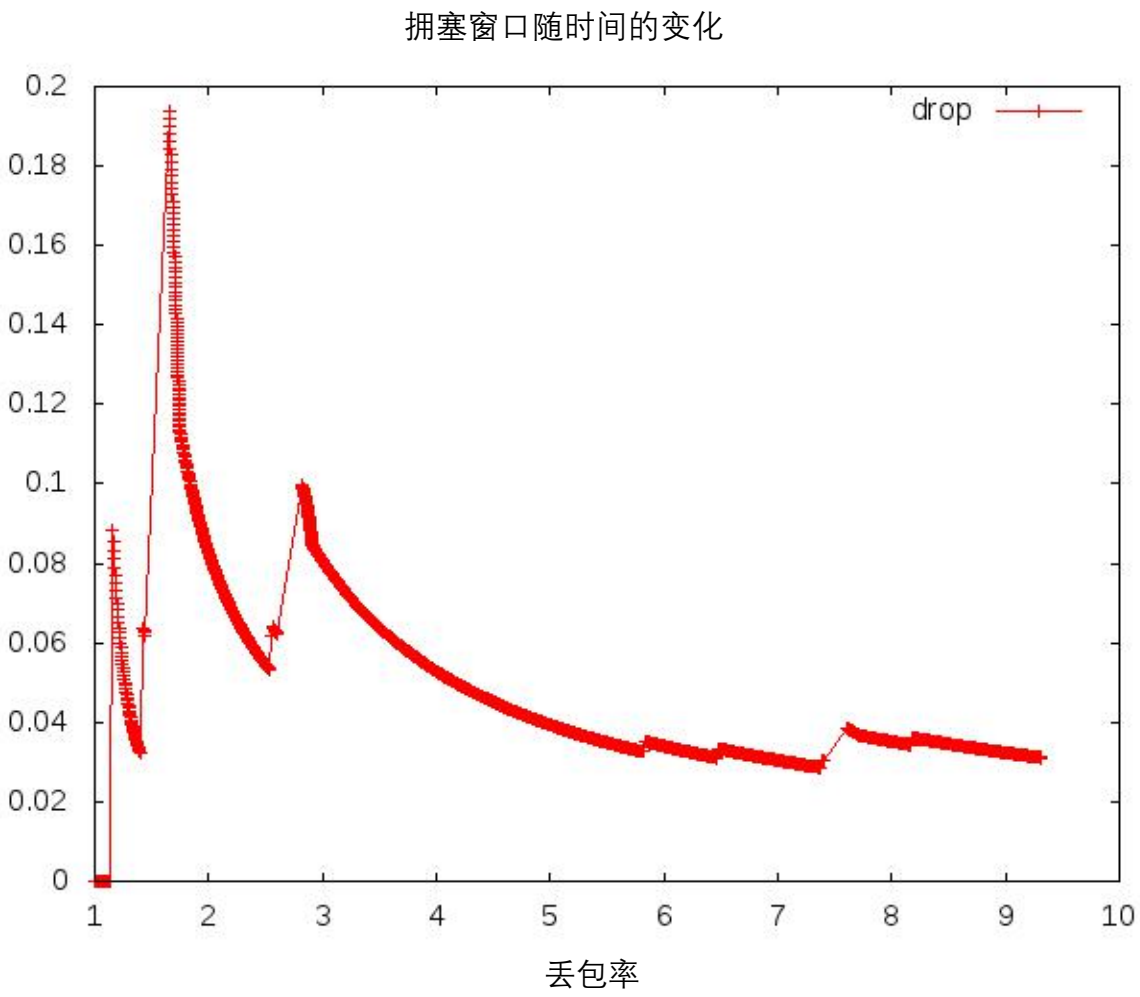
系统出现：gnuplot>是提示符，所有gnuplot命令在此输入

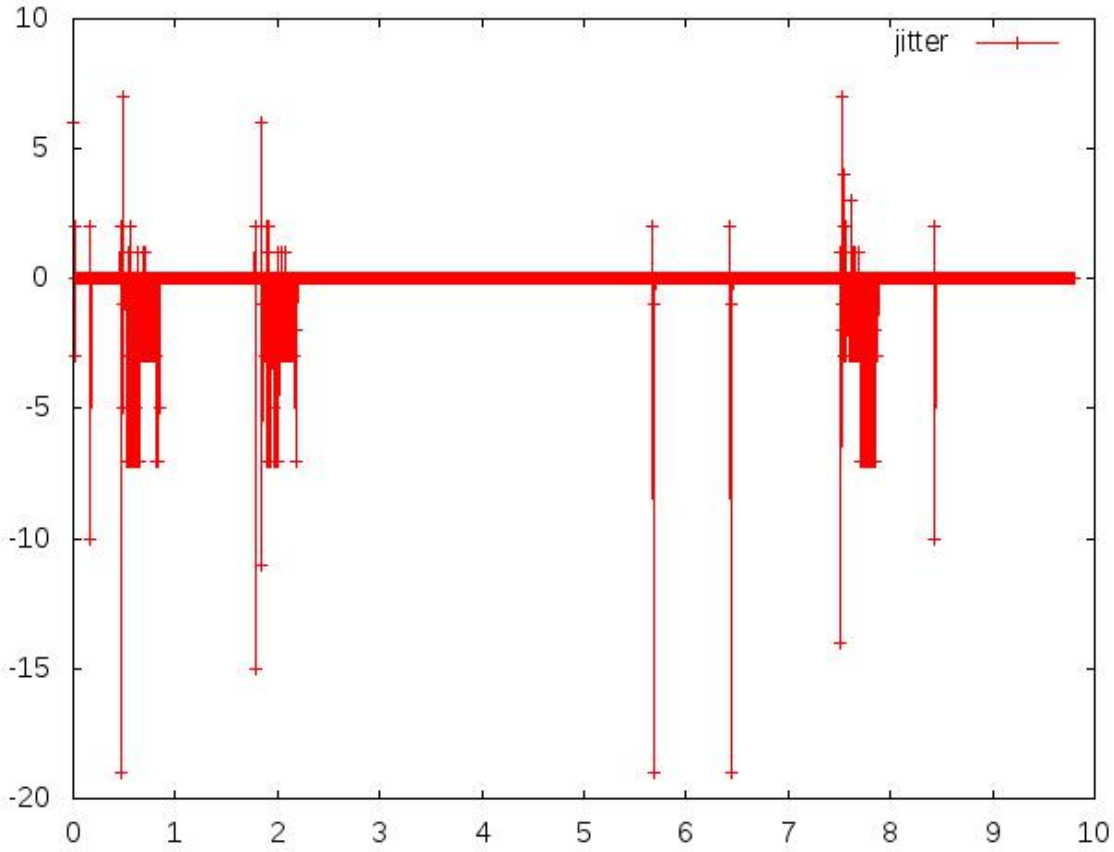


下面用同样的拓扑，应用程序以及同样的错误模型仿真输出 TCP 拥塞窗口值随时间的变

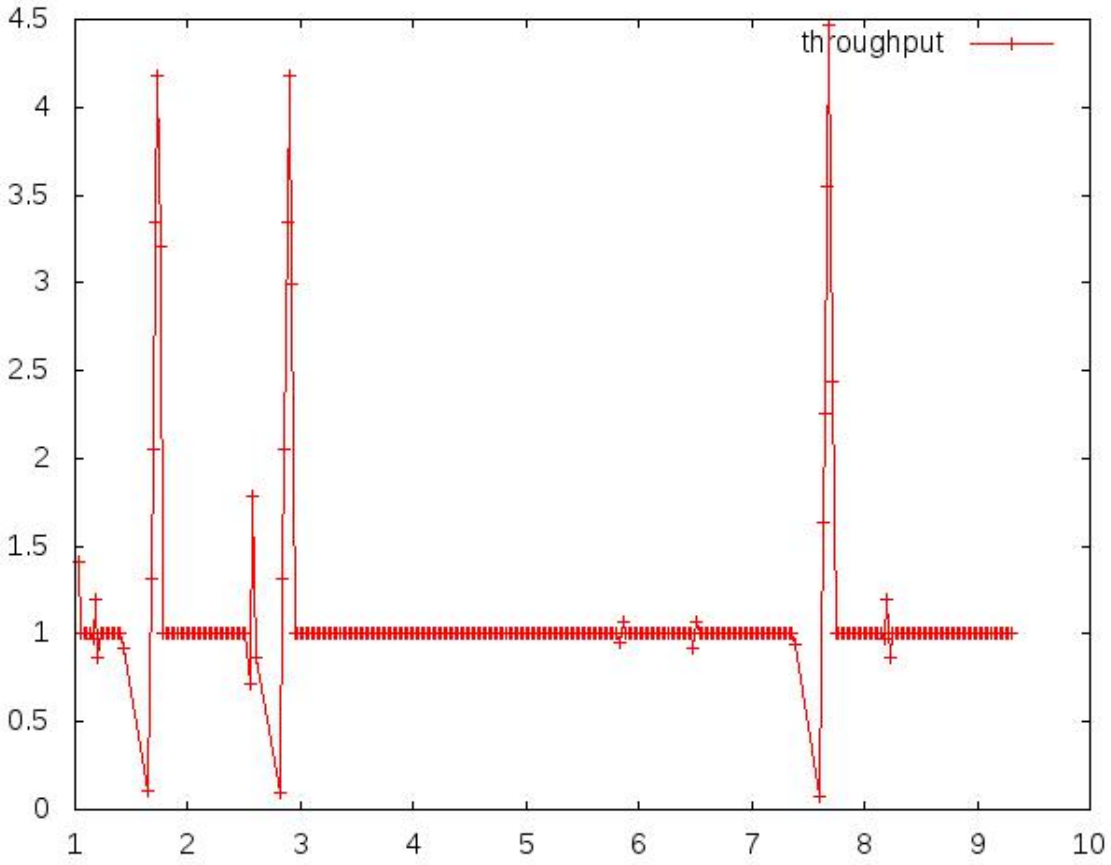


化，抖动率，丢包率，吞吐量：


拥塞窗口随时间的变化
drop
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
1
2
3
4
5
6
7
8
9
10
丢包率


jitter
10
5
0
-5
-10
-15
-20
0
1
2
3
4
5
6
7
8
9
10

抖动率


throughput
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1
2
3
4
5
6
7
8
9
10

吞吐量

在仿真结果中我们可以看到：当网络传输出现差错传输，导致链路拥塞，使得拥塞窗口值陡然降低，致使链路的时延变大，抖动率变化也比较明显，吞路量也变小。

更多内容详见官方 tutorial（附件中）