NS2 学习笔记

燕志伟

西安交通大学电信学院

2005-6-11

序

学习 NS 已经有两、三月有余。期间涉及操作系统的安装，NS 的安装与调试等等方方面面的内容。有时，一个命令的错误百思不得其解，往往会困扰我一日，甚至数日之久。后来，发现所有的内容都是在逻辑上成立的。只是，我当时的理解有所偏差而已。

刚开始，触及 NS 时，被其搞得一头雾水；不得其要领。而且，初学的书籍较少，多是网上的资料。而又多是多人转手 Copy, Paste，更是令人头大。经过这两、三个月的初步摸索，基本弄清了 NS 的逻辑关系。

这个学习笔记，是我这二、三个月的学习日志的总结和整理。以前习惯于在纸上写笔记，为了能记录在计算机的操作，需要在屏幕上截图，迫使我开始习惯于在计算机上记录我的学习笔记。现在想来，这也是一件好事情。

因为 NS 是在类似 UNIX 的系统上安装运行的，而我们又习惯于用 Windows 的系统，所以我在 WinXP 系统上安装了 VMWare 系统，然后又在其上安装 FreeBSD4.10 系统。（其实我也试过 cyWin 的，总是错误不断，也是迫不得已呀。）不过这个 VMWare 确是不错的，老外，真是历害，能把这个软件做到这个程度真是不易。这个软件还有一个好处，就是配置 XFree86 的时候较为容易，装一下 VMTools 后，会在虚拟光盘上多一个文件，解压并运行它，并接受它的默认值就可以了。这个对 XFee86 的兼容性极好，省下不小麻烦。

学习 NS 是要有一定基础的，要对 C 语言和 C＋＋的类与对象的概念比较清楚，才能搞清 NS 要做的事情。

在这次的学习中，参考了不少网上的资料。其中许多是论坛中的，能记得在最后的文献中都注明了。记不得的，也就无法一一注明。总之，我在此一并谢过了。

燕志伟

于　西安交通大学西一楼

2005-6-11

目录

1. 在 FreeBSD 下 NS 的安装

1.1.FreeBSD 的安装

启动 VMware 虚拟机，并放入 FreeBSD 安装光盘，

选择 stand 安装，并安装 Xwindows—Gome，装一下 VMTools 后，会在虚拟光盘上多一个文件，解压并运行它，并接受它的默认值就可以了。这样基本系统就安装好了。

1.2.NS2 安装

用 root 用户登录，

通过 NS 的 ftp 命令，登录到一个FTP 上，把 ns2.28-allinone-tar.gz 文件下载到/usr/home/的目录中去；

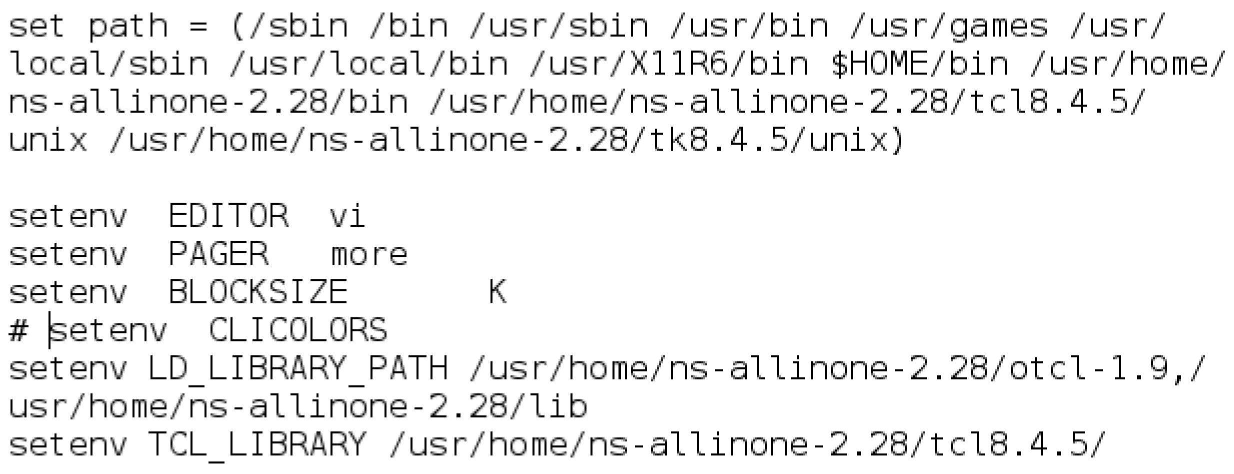
用命令：

Tar –zxf ns2.28-allinone-tar.gz

会释放文件到目录 ns-2.28 下；

进入 Ns-2.28 这个目录会有一个 install 的命令，运行它./install 可完成文件的编译和安装；

打开/root/.cshrc 文件修改如下：（加入路径和与环境参数）



```
set path = (/sbin /bin /usr/sbin /usr/bin /usr/games /usr/
local/sbin /usr/local/bin /usr/X11R6/bin $HOME/bin /usr/home/
ns-allinone-2.28/bin /usr/home/ns-allinone-2.28/tcl8.4.5/
unix /usr/home/ns-allinone-2.28/tk8.4.5/unix)

setenv  EDITOR  vi
setenv  PAGER   more
setenv  BLOCKSIZE       K
# setenv  CLICOLORS
setenv LD_LIBRARY_PATH /usr/home/ns-allinone-2.28/otcl-1.9,/
usr/home/ns-allinone-2.28/lib
setenv TCL_LIBRARY /usr/home/ns-allinone-2.28/tcl8.4.5/
```

图表 1

重新启动后，即可使用 NS.

1.3.建立一个用于学习 NS 用户：

用 root 用户登录，命令：Adduser –v　建立一个名为 nsuser 的用户。

使用 csh 的 shell

拷贝\root\.cshrc 文件到\usr\home\nsuser 目录下，覆盖其原有的文件。即是，把路径设置拷贝过去。（这一步，可做可不做）（注意文件名是 dot.cshrc）

2. NS2 中 OTCL 的基本概念

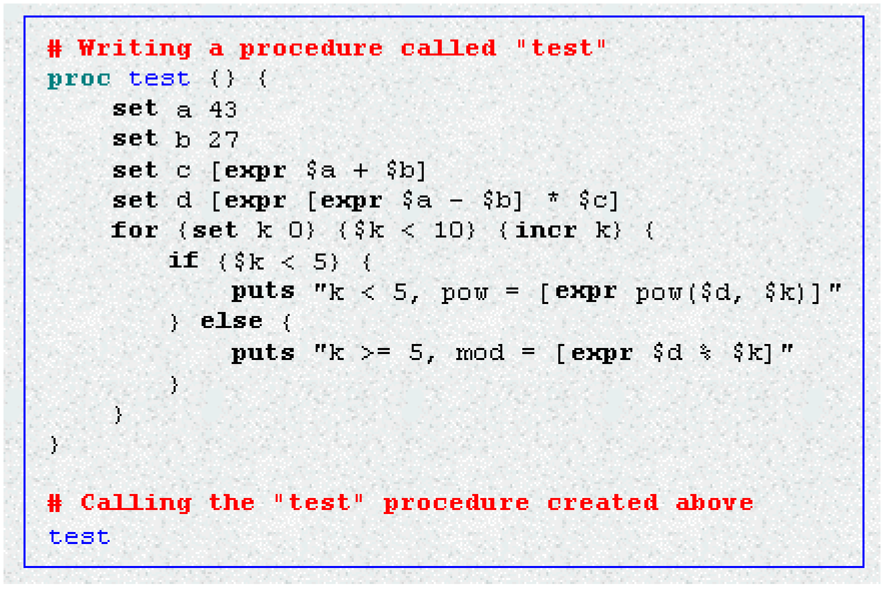
2.1.OTcl 变量与表达式基本用法

变量名前加一个美元符号$，如$a，$b。

函数及程序中所有的分隔符号是大括号 curly brackets {}

变量赋值用 set 命令，如 set $a 5；

表达式用[expr …]来求值，如[expr $a + $b]；**注意，中括号是不能省略的，且可以嵌套使用。**



```
# Writing a procedure called "test"
proc test {} {
    set a 43
    set b 27
    set c [expr $a + $b]
    set d [expr [expr $a - $b] * $c]
    for {set k 0} {$k < 10} {incr k} {
        if {$k < 5} {
            puts "k < 5, pow = [expr pow($d, $k)]"
        } else {
            puts "k >= 5, mod = [expr $d % $k]"
        }
    }
}

# Calling the "test" procedure created above
test
```

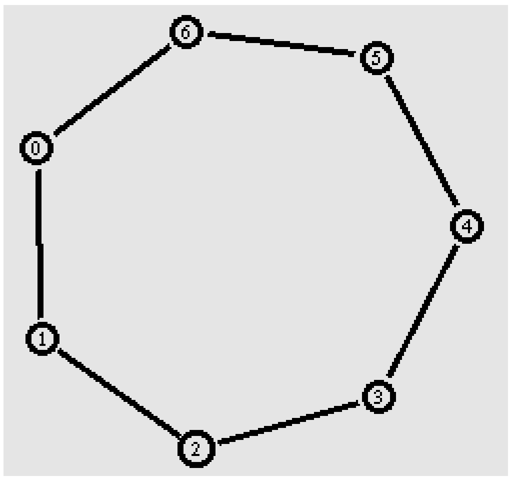
注意循环语句和选择语句的用法

循环结构（不需事先声明变量）

```
for {set i 0} {$i < 7} {incr i} {
  set n($i) [$ns node]
}
```

将七个结点相连接

```
for {set i 0} {$i < 7} {incr i} {
  $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail
}
```



图表 2

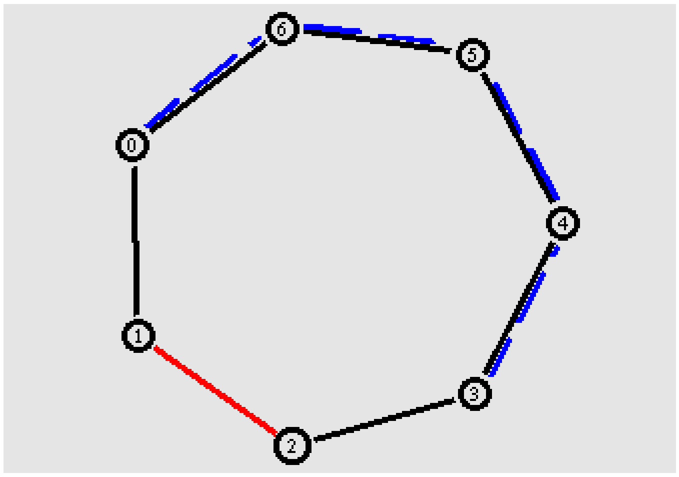
让结点 1 和结点 2 间的链接中断 1 秒

$ns rtmodel-at 1.0 down $n(1) $n(2)

$ns rtmodel-at 2.0 up $n(1) $n(2)

使用动态路由算法

$ns rtproto DV



图表 3

Puts 是用来输出字符的。

2.2.OTcl 面向对象的用法

Tcl 和 OTcl 的关系有点象 C 和 C++的关系

Class 用来定义一个类的声明结构

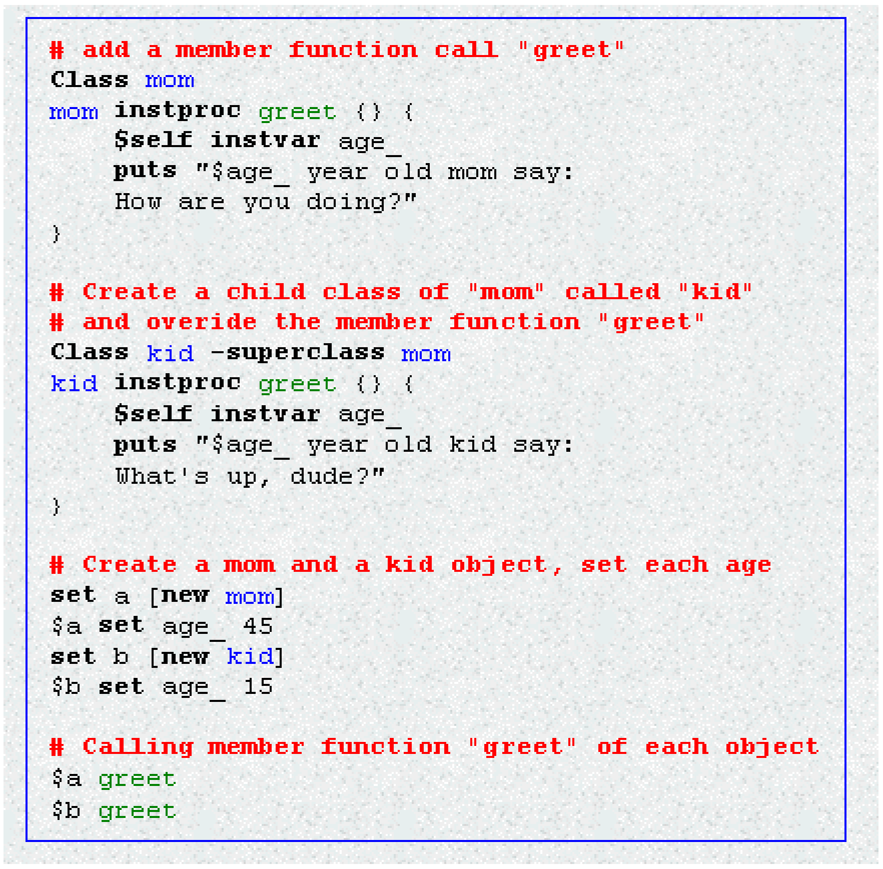
Instproc 用来定义类中的成员函数

-superclass 用来指定子类与父类之间的继承关系，相当于 C++中的冒号“：”

$self 的作用和 C++中的 this 指针的作用是类似的，表明是本类中。主要是用来区别继承下来的变量与本类中重载的变量。

Instvar 有二个作用，一是用来检查变量的定义是否出现过，如果在父类中或以前定义过了，这里就是一个引用；否则是一个**真正的定义**。

New 是用来完成一个对象的生成的。

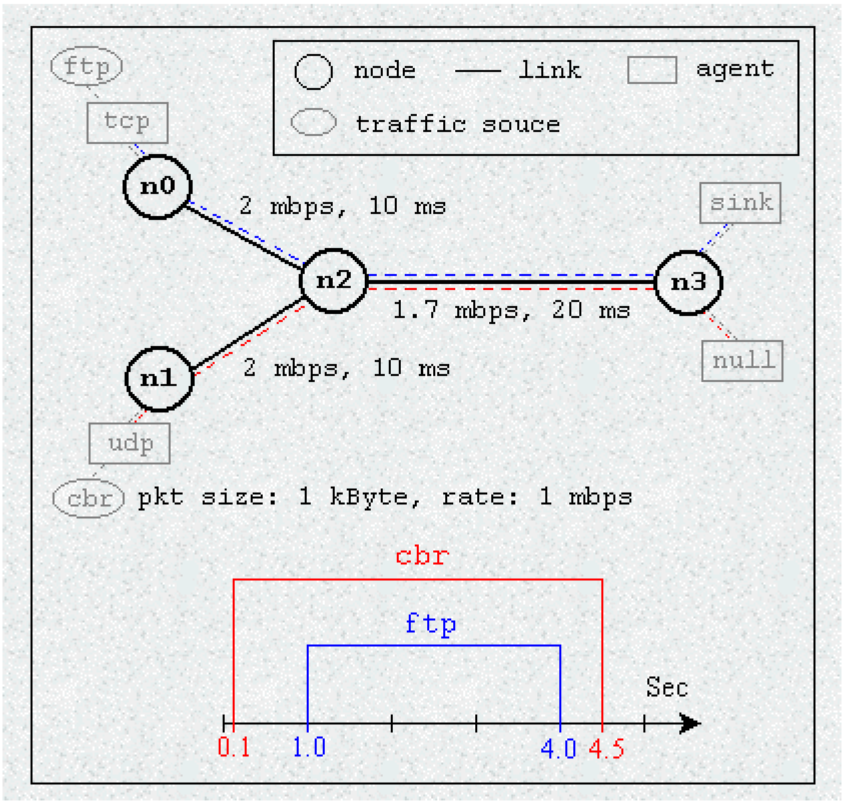


```
# add a member function call "greet"
Class mom
mom instproc greet {} {
     $self instvar age_
     puts "$age_ year old mom say:
     How are you doing?"
}

# Create a child class of "mom" called "kid"
# and overide the member function "greet"
Class kid -superclass mom
kid instproc greet {} {
     $self instvar age_
     puts "$age_ year old kid say:
     What's up, dude?"
}

# Create a mom and a kid object, set each age
set a [new mom]
$a set age_ 45
set b [new kid]
$b set age_ 15

# Calling member function "greet" of each object
$a greet
$b greet
```

图表 4

3. NS2 模拟网络第一例：

3.1. 例子 1：

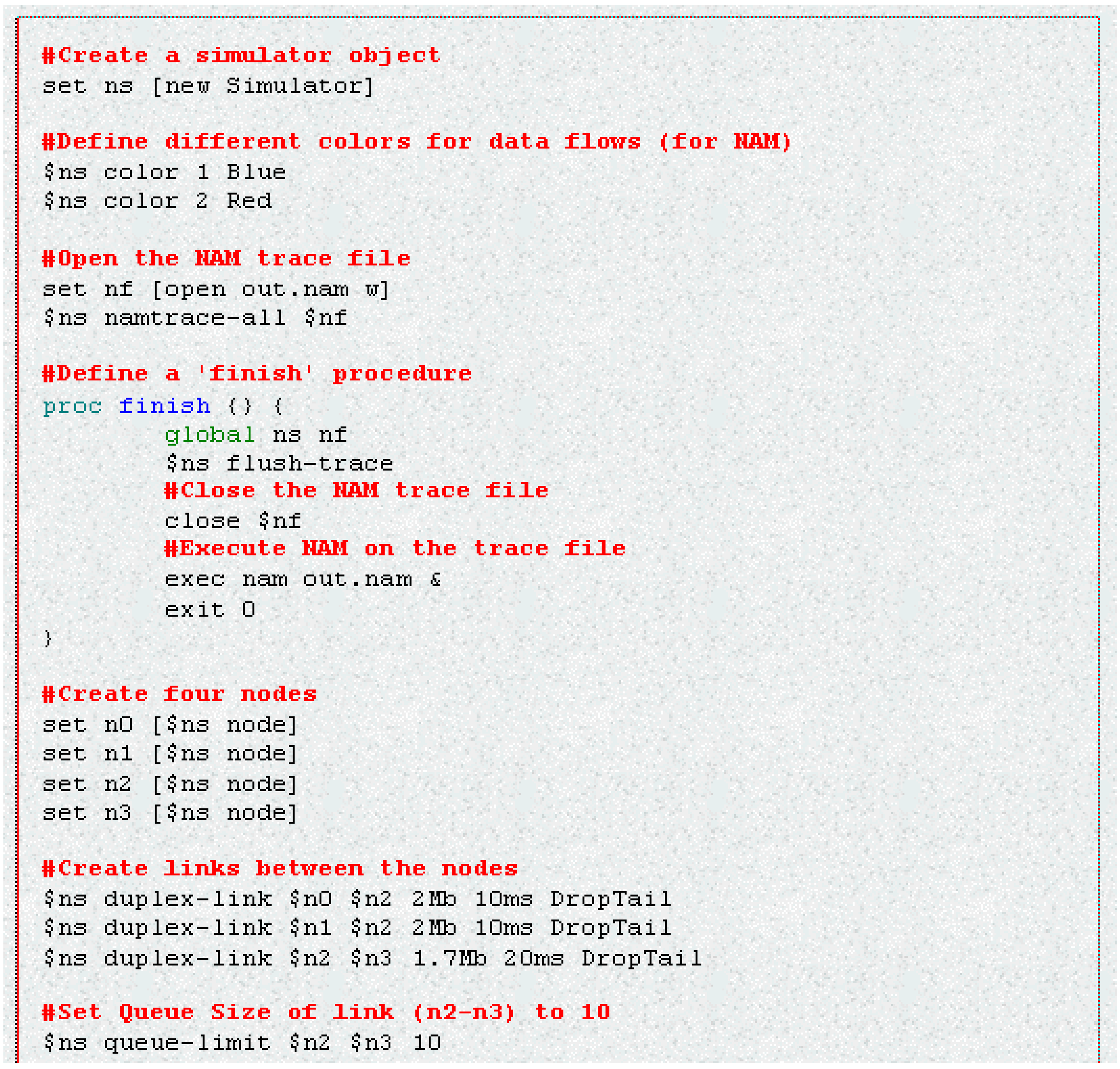


缺省设置下，TCP 的包最大为 1KByte

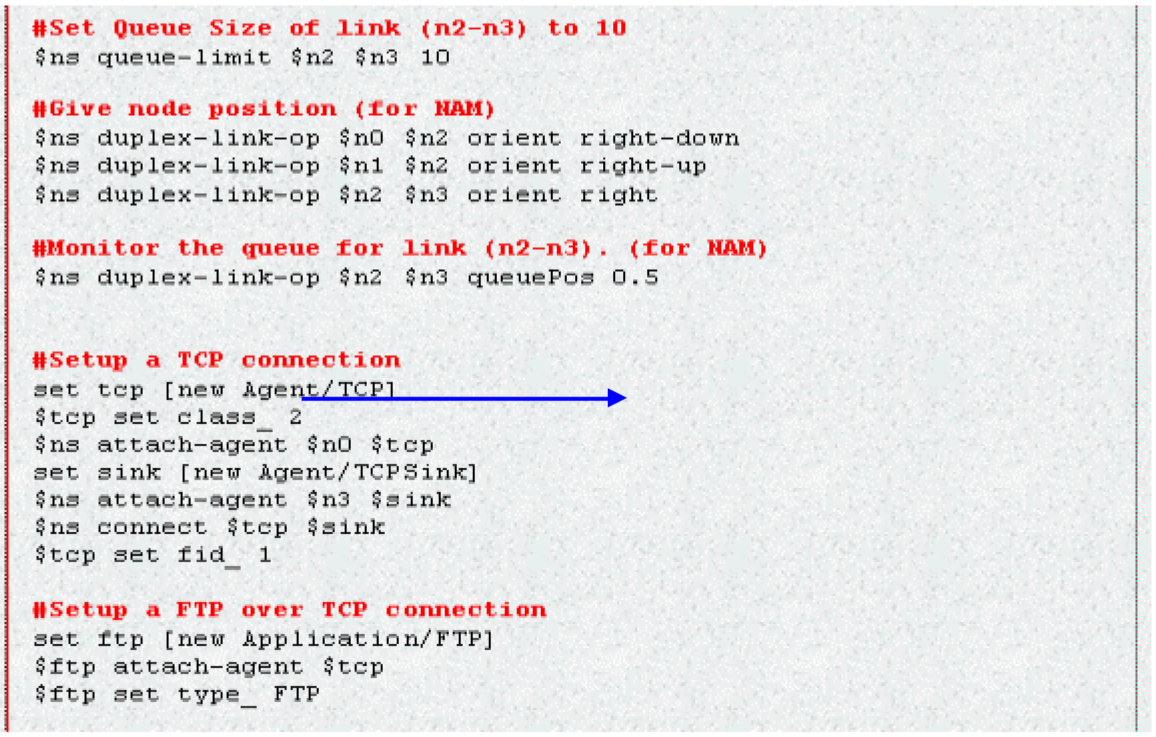
每个节点的排队方法为 DropTail，排队长度为 10

Udp 和 Tcp 是 agent，Ftp 和 Cbr 是流量生成器。

TCP 的目的地是 TCP 的 sink Agent，它会吸收接收的 TCP 数据包，生成和发送 ACK 数据包。UDP 的目的地是 UDP 的 null Agent，它会吸收接收到的 UDP 数据包。



```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10
```



```
#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10
```

```
#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5


#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
```

```
#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false


#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```

将上面这个程序存盘为 ex-example.tcl

然后运行命令(只有在 Xwindows 的终端上运行才会有窗口出现):

ns ex-example.tcl

3.2. 各条语句的解释

3.2.1. 原 (英)文 解 释

set ns [new Simulator]: generates an NS simulator object instance, and assigns it to variable ns (italics is used for variables and values in this section). What this line does is the following:

- Initialize the packet format (ignore this for now)
- Create a scheduler (default is calendar scheduler)
- Select the default address format (ignore this for now)

The "Simulator" object has member functions that do the following:

- Create compound objects such as nodes and links (described later)
- Connect network component objects created (ex. attach-agent)
- Set network component parameters (mostly for compound objects)
- Create connections between agents (ex. make connection between a "tcp" and "sink")
- Specify NAM display options
- Etc.

Most of member functions are for simulation setup (referred to as plumbing functions in the Overview section) and scheduling, however some of them are for the NAM display. The "Simulator" object member function implementations are located in the " ns-lib.tcl" file.

armor Note: /ns-2/tcl/lib/ns-lib.tcl is an important file. Read it as fast as you can.

$ns color fid color: is to set color of the packets for a flow specified by the flow id (fid) . This member function of "Simulator" object is for the NAM display, and has no effect on the actual simulation. (设置标记颜色)

$ns namtrace-all file-descriptor: This member function tells the simulator to record simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command $ns flush-trace. Similarly, the member function trace-all is for recording the simulation trace in a general format.

proc finish {}: is called after this simulation is over by the command $ns at 5.0 "finish". In this function, post-simulation processes are specified.

Node

set n0 [$ns node]: The member function node creates a node. A node in NS is compound object made of address and port classifiers (described in a later section). Users can create a node by separately creating an address and a port classifier objects and connecting them together. However, this member function of Simulator object makes the job easier. To see how a node is created, look at the files: "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-node.tcl".

armor Note: ns-2/tcl/libs /ns-node.tcl is an important file. Read it as fast as you can.

Link

```
#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail
```

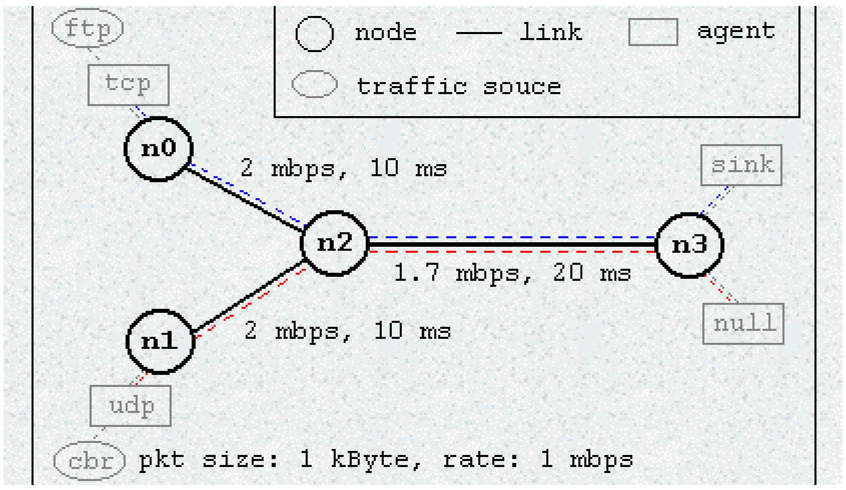
$ns duplex-link node1 node2 bandwidth delay queue-type: creates two simplex links of specified bandwidth and delay, and connects the two specified nodes. In NS, the output queue of a node is implemented as a part of a link, therefore users should specify the queue-type when creating links. In the above simulation script, DropTail queue is used. If the reader wants to use a RED queue, simply replace the word DropTail with RED. The NS implementation of a link is shown in a later section. Like a node, a link is a compound object, and users can create its sub-objects and connect them and the nodes. Link source codes can be found in "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns -link.tcl" files. One thing to note is that you can insert error modules in a link component to simulate a lossy link (actually users can make and insert any network objects). Refer to the NS documentation to find out how to do this.

armor Note: ns-link.tcl is an important file. Read it as fast as you can.

```
#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10
```

$ns queue-limit node1 node2 number: This line sets the queue limit of the two simplex links that connect node1 and node2 to the number specified. At this point, the authors do not know how many of these kinds of member functions of Simulator objects are available and what they are. Please take a look at "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-link.tcl", or NS documentation for more information. (队列长度为 10)

$ns duplex-link-op node1 node2 ...: The next couple of lines are used for the NAM display. To see the effects of these lines, users can comment these lines out and try the simulation.

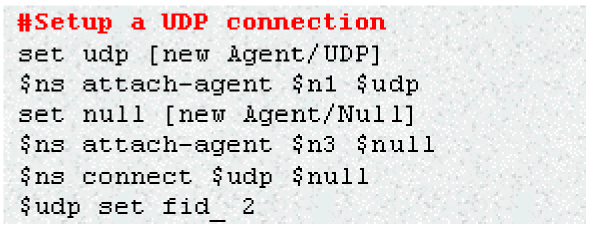


Agent (个人认为是在传输层设置的 Agent)

Now that the basic network setup is done, the next thing to do is to setup traffic agents such as TCP and UDP, traffic sources such as FTP and CBR, and attach them to nodes and agents respectively.

set tcp [new Agent/TCP]: This line shows how to create a TCP agent. But in general, users can create any agent or traffic sources in this way. Agents and traffic sources are in fact basic objects (not compound objects), mostly implemented in C++ and linked to OTcl. Therefore, there are no specific Simulator object member functions that create these object instances. To create agents or traffic sources, a user should know the class names these objects (Agent/TCP, Agnet/TCPSink, Application/FTP and so on). This information can be found in the NS documentation or partly in this documentation. But one shortcut is to look at the "ns-2/tcl/libs/ns-default.tcl" file. This file contains the default configurable parameter value settings for available network objects. Therefore, it works as a good indicator of what kind of network objects are available in NS and what are the configurable parameters.

$ns attach-agent node agent: The attach-agent member function attaches an agent object created to a node object. Actually, what this function does is call the attach member function of specified node, which attaches the given agent to itself. Therefore, a user can do the same thing by, for example, $n0 attach $tcp. Similarly, each agent object has a member function attach-agent that attaches a traffic source object to itself.



```
#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2
```

$ns connect agent1 agent2: After two agents that will communicate with each other are created, the next thing is to establish a logical network connection between them. This line establishes a network connection by setting the destination address to each others' network and port address pair.

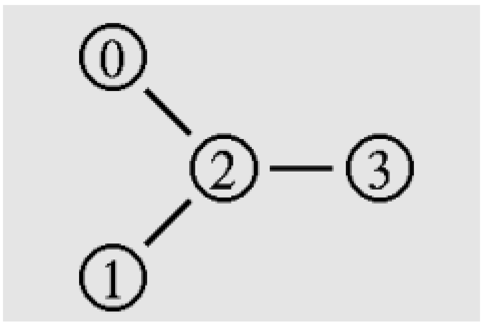
Assuming that all the network configuration is done, the next thing to do is write a simulation scenario (i.e. simulation scheduling). The Simulator object has many scheduling member functions. However, the one that is mostly used is the following:

$ns at time "string": This member function of a Simulator object makes the scheduler (scheduler_ is the variable that points the scheduler object created by [new Scheduler] command at the beginning of the script) to schedule the execution of the specified string at given simulation time. For example, $ns at 0.1 "$cbr start" will make the scheduler call a start member function of the CBR traffic source object, which starts the CBR to transmit data. In NS, usually a traffic source does not transmit actual data, but it notifies the underlying agent that it has some amount of data to transmit, and the agent, just knowing how much of the data to transfer, creates packets and sends them.

After all network configuration, scheduling and post-simulation procedure specifications are done, the only thing left is to run the simulation. This is done by $ns run.

3.2.2.我自己的笔记解释

# 设置拓扑 topology 步骤

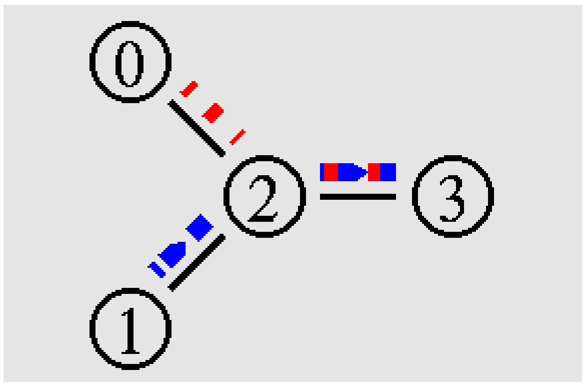
A 设置若干个结点 Node;	set n0 [$ns node] set n1 [$ns node] set n2 [$ns node] set n3 [$ns node]
B 对各个结点进行连接;	$ns duplex-link $n0 $n2 1Mb 10ms DropTail $ns duplex-link $n1 $n2 1Mb 10ms DropTail $ns duplex-link $n3 $n2 1Mb 10ms DropTail
C 对各个结点进行布局 Layout;	$ns duplex-link-op $n0 $n2 orient right-down $ns duplex-link-op $n1 $n2 orient right-up $ns duplex-link-op $n2 $n3 orient right
 图表 5	

设置若干个数据源Agent；将数据源与结点相结合；数据类型与数据源匹配

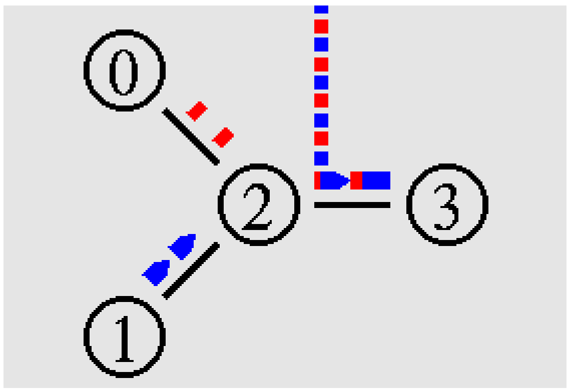
These lines create a UDP agent and attach it to the node, then attach a CBR traffic generatot to the UDP agent.

#Create a UDP agent and attach it to node n0	set udp0 [new Agent/UDP] $ns attach-agent $n0 $udp0
# Create a CBR traffic source and attach it to udp0	set cbr0 [new Application/Traffic/CBR] $cbr0 set packetSize_ 500 $cbr0 set interval_ 0.005 $cbr0 attach-agent $udp0
#Create a UDP agent and attach it to node n1	set udp1 [new Agent/UDP] $ns attach-agent $n1 $udp1
# Create a CBR traffic source and attach it to udp1	set cbr1 [new Application/Traffic/CBR] $cbr1 set packetSize_ 500 $cbr1 set interval_ 0.005 $cbr1 attach-agent $udp1
注：结点 Node🡐数据源 Agent(UDP)🡐 数据类型 CBR(参数) ?为何不连接 $N2 到 $null0： 表明 N0 和 N1 结点发出数据的最后目的是 N3 结点 UPD 的终点是 NULL，TCP 的终点是 SINK 网络参考模型是对等网络，每层相对	set null0 [new Agent/Null] $ns attach-agent $n3 $null0 $ns connect $udp0 $null0 $ns connect $udp1 $null0
Create a Null agent which acts as traffic sink and attach it to node n1.	set null0 [new Agent/Null] $ns attach-agent $n3 $null0
Now the two agents have to be connected with each other.	$ns connect $udp1 $null0

#标记数据流

$udp0 set class_ 1 $udp1 set class_ 2 $ns color 1 Blue $ns color 2 Red	 图表 6

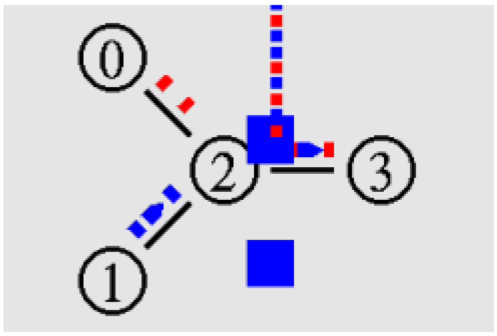
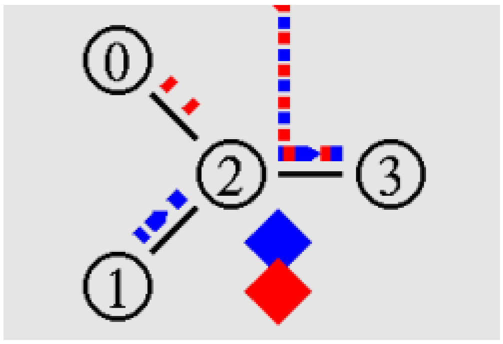
```
# 监视在结点2 的排队情况
```

$ns duplex-link-op $n2 $n3 queuePos 0.5	 图表 7

```
# 在结点2 处的丢包情况
```

DropTail 即是只当暂存区满载时，若再有数据包传送过来，就立即丢掉

SFQ (stochastic fair queueing) 随机公平排队：？？？？

 图表 8	 图表 9
$ns duplex-link $n3 $n2 1Mb 10ms DropTail	$ns duplex-link $n3 $n2 1Mb 10ms SFQ

SFQ(stochastic fair queueing)

原文：

stochastic fair queueing tries to divide the available bandwidth equally among all users. Ideally, a separate queue would be used for each ongoing connection, but high-performance routers lack the resources to do things that way. So a smaller number of queues is used, with each connection being assigned to a queue via a hash function. Packets are then taken from each queue in turn, dividing the bandwidth between them. If two high-bandwidth connections happen to land on the same queue, they will be penalized relative to the other queues; to address this problem, the hash function is periodically changed to redistribute connections among the queues. The algorithm works reasonably well and is easy to make fast; the Linux networking code has had a stochastic queueing module available for some time.

中文翻译： （by Yan Zhiwei）

随机公平排队的目的是将所有的带宽平均地分配给每一个用户。在理想的情况下，N 个队列为 N 个链接进行服务。但是，即使是高性能的路由器也没有足够的资源和能力来完成这个功能。在实际中，路由器使用较少的几个队列（例如 M 个：M<<N）进行排队算法的操作。对于 N 个链接中的一个链接，可以通过一个 Hash 函数把它指定到其中的一个队形中去排队。对于这 M 个队列来说，是公平的。它们平均分配带宽，依次从每个队列中取得数据包进行处理。有时，会出现两个或多个需要高带宽资源的链接被分配到同一个队列中去排队，从而使得服务质量下降。为了防止这种问题的出现，在这些队列中，Hash 函数要被周期的改变来重新分配链接。这个算法在实际当中，工作性能比较好，速度也很快。

参考文献：

http://lwn.net/Articles/22028/

4. NS2 的组成机理及模型

4.1.OTCL 的网络组成

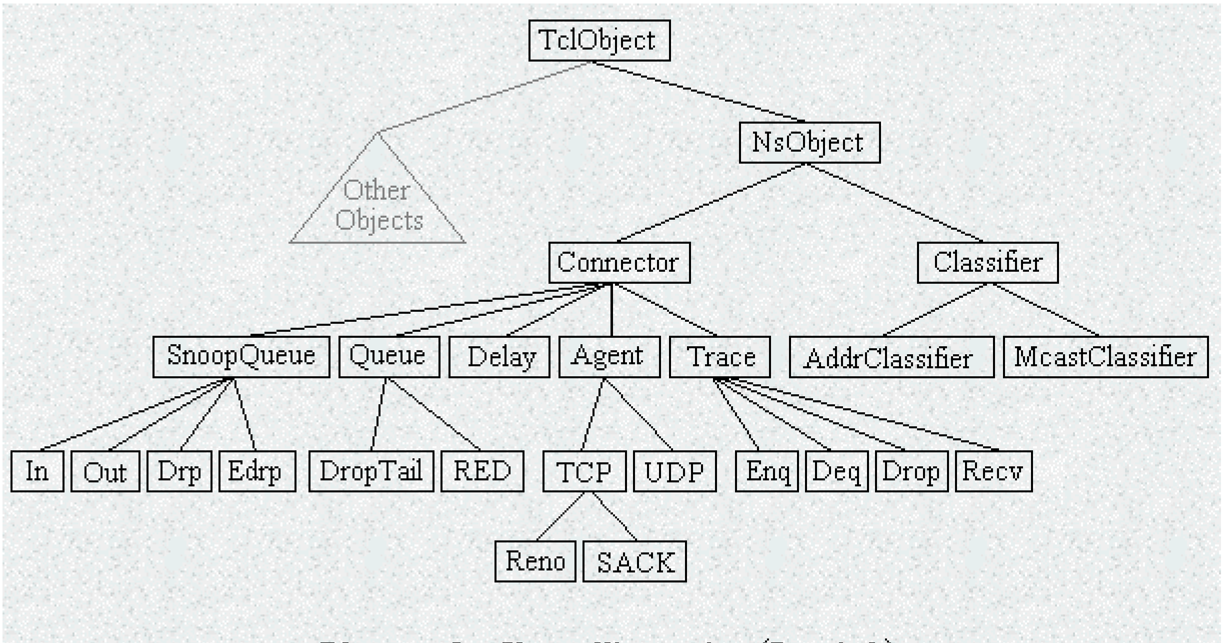


Figure 6. Class Hierarchy (Partial)

TclOject 是子类中层次最低的，NsObject 是又有两个父类，根据输出端口的数目，有 Connector(只有一个输出)和 Classfier(可以有多个输出)。

4.2.结点与路由

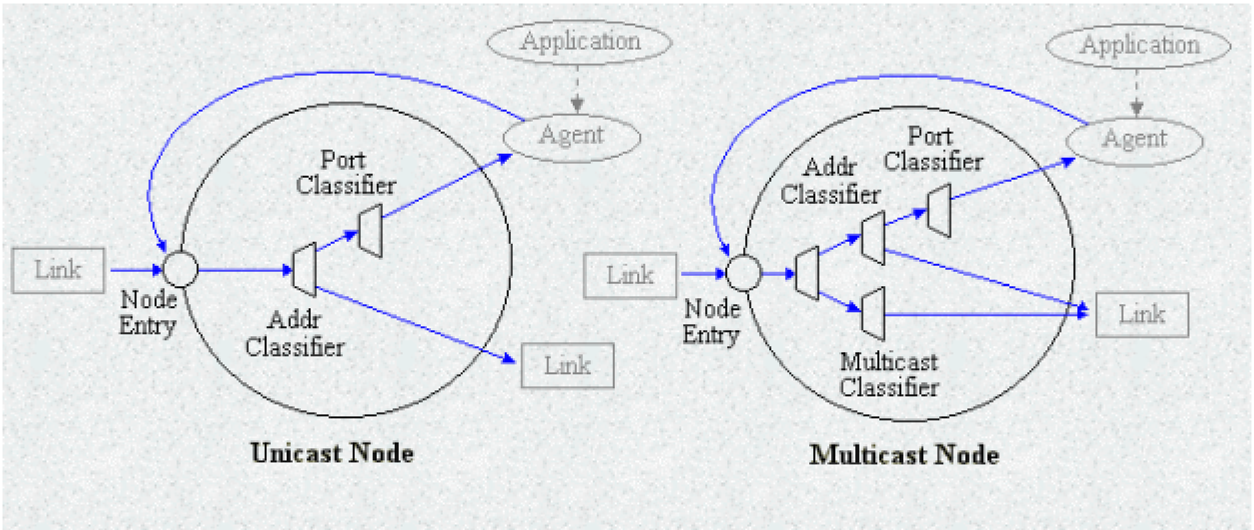


Figure 7. Node (Unicast and Multicast)

缺省为单播的，多播须人为指定。

语法：

- **Unicast**
  - *$ns* rtproto *type*
  - *type*: Static, Session, DB, cost, multi-path

- **Multicast**
  - *$ns* multicast (right after set *$ns* [new Scheduler])
  - *$ns* mrtproto *type*
  - *type*: CtrMcast, DM, ST, BST

4.3. 链接 link

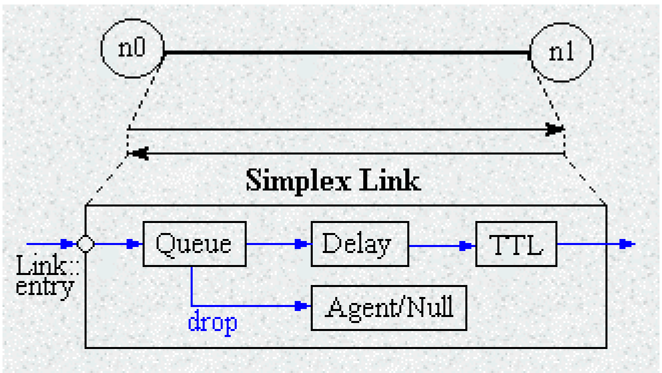


Figure 8. Link

Link 对象实际上包括了一个结点的输出的队列，数据包出队列后，经延迟和生存时间 TTL 的改变后，再进入下一个结点的输入队列。也就是说，队列是在一个 link 上的，而不在结点上。

如果队列满了，则丢弃—实际上是进入了一个吸收的结点 NULL

4.3.1. 跟 踪 trace

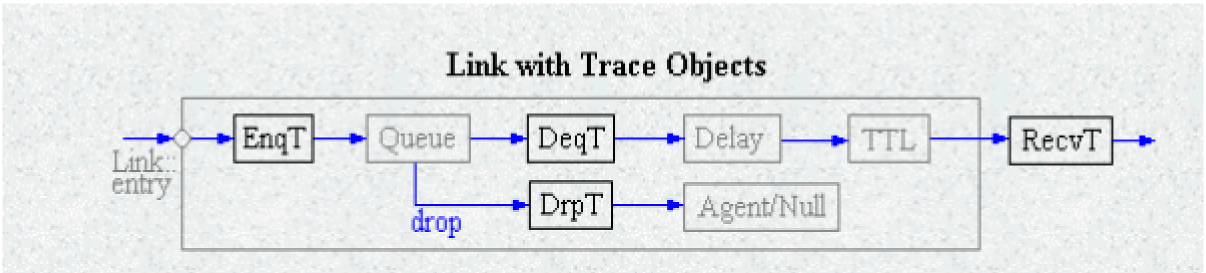


Figure 9. Inserting Trace Objects

可以和图 8 比较，加入了四种跟踪对象。

语法：

跟踪所有的，

$ns trace-all file or $ns namtrace-all file

跟踪一部分，

create-trace {type file src dst}

4.3.2. 监视队列 Queue Monitor

Link with Snoop Queue Objects

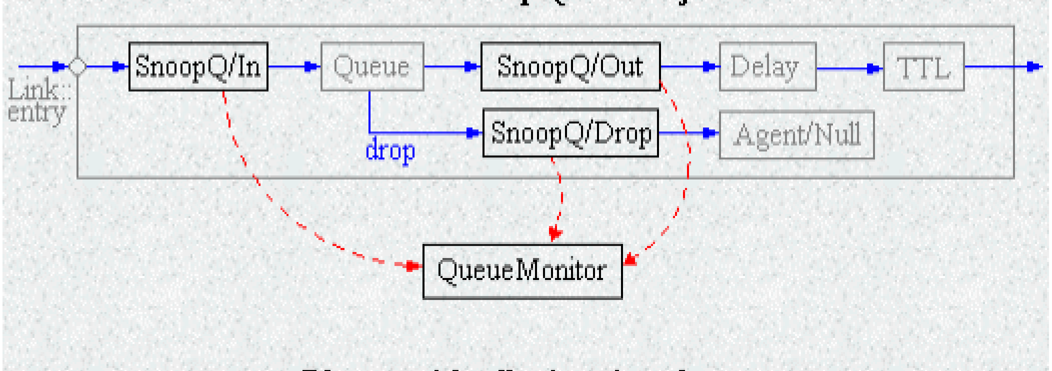


Figure 10. Monitoring Queue

如果对输出队列长度感兴趣的话，加入 snoop 的队列，用于监视队列长度。

如图 10，当一个数据包到达后，Snoop 会通知 Queue Monitor 进行记录队列的信息。

4.4. 一个完整的例子

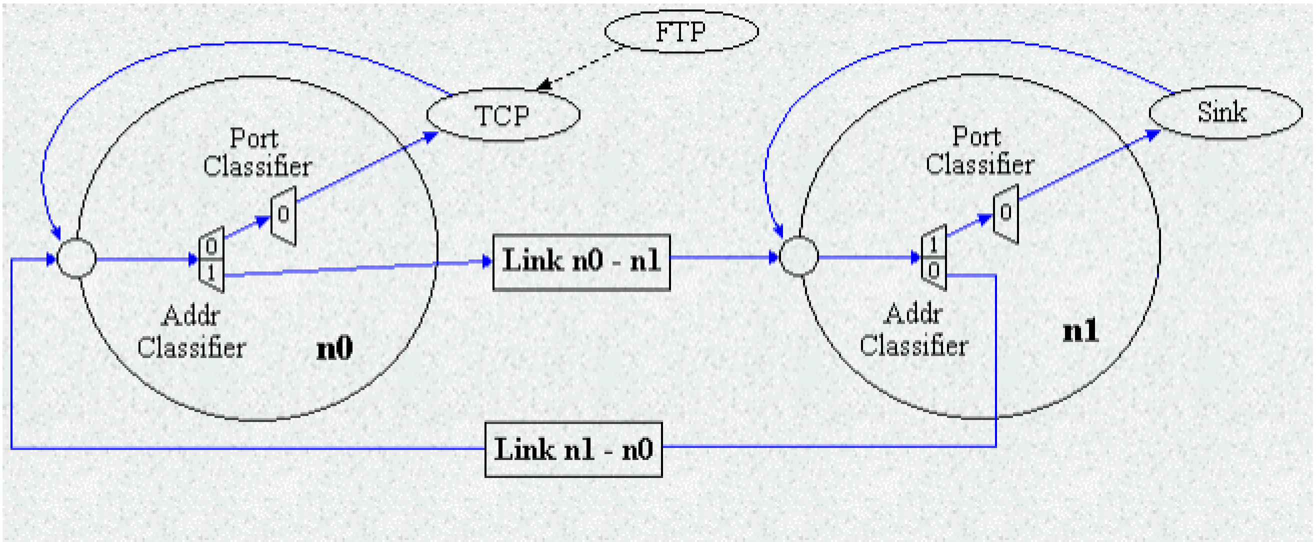


Figure 11. Packet Flow Example

N0：port:0 address:0

N1：port:0 address:1

5. NS 数据包结构

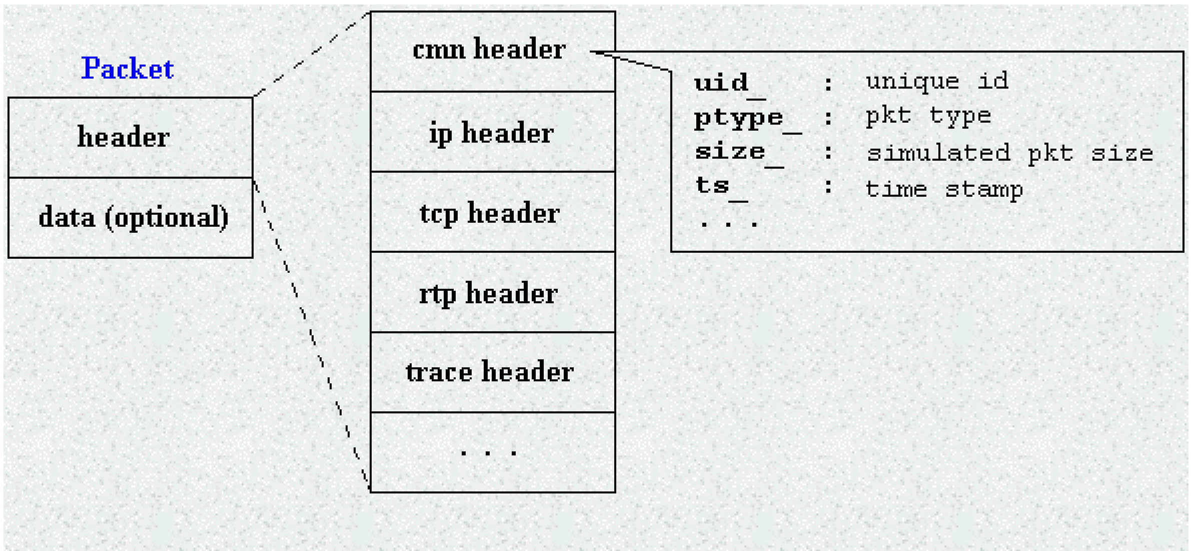


Figure 12. NS Packet Format

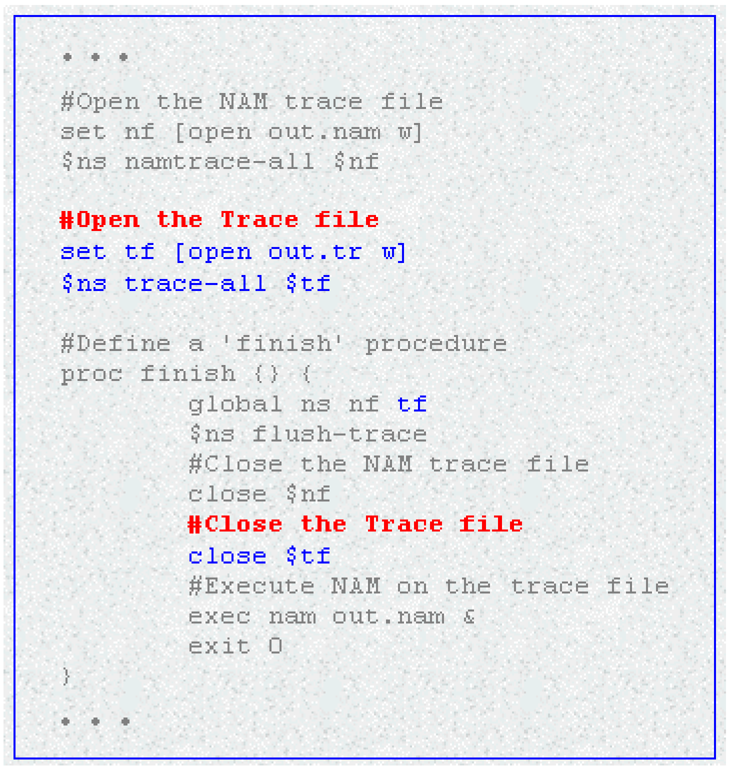
注意，UDP 使用 rtp 的数据头。

Cmn header 是公用的数据头。

通常 NS 使用的数据包是没有数据段的，这是因为在一个不是实时仿真的环境中，加上了数据段的意义不是很大。不过，如果需要，可以修改这个数据结构加入实际的数据，或是自己再定义一个新的数据结构。

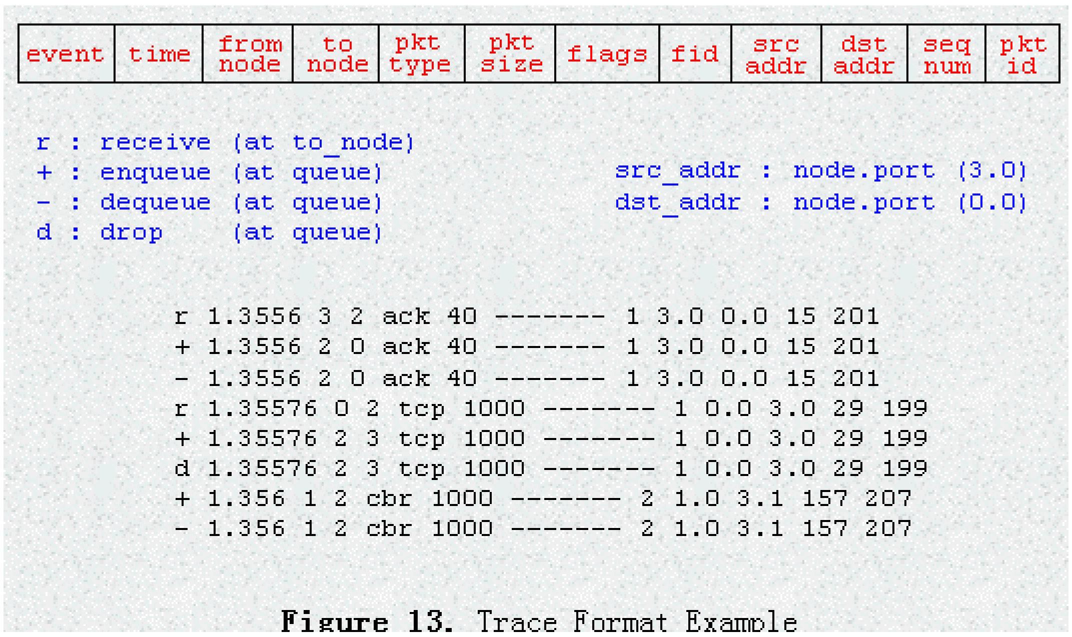
6. NS2 跟踪及后期数据处理

6.1. 设置跟踪文件



```
. . .
#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Close the Trace file
        close $tf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}
. . .
```

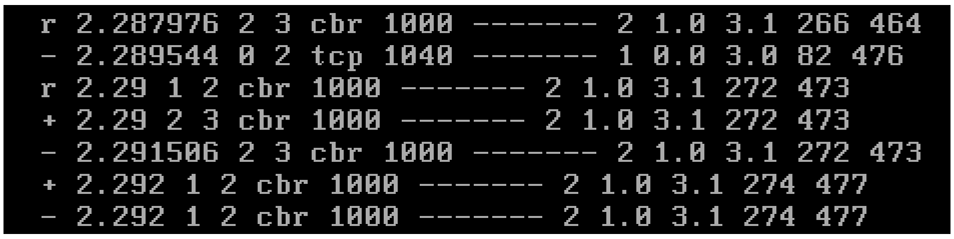
文件格式



event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

```
r : receive (at to_node)
+ : enqueue (at queue)            src_addr : node.port (3.0)
- : dequeue (at queue)            dst_addr : node.port (0.0)
d : drop    (at queue)

        r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
        + 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
        - 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
        r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
        + 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
        d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
        + 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
        - 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

Figure 13. Trace Format Example



```
r 2.287976 2 3 cbr 1000 -------- 2 1.0 3.1 266 464
- 2.289544 0 2 tcp 1040 -------- 1 0.0 3.0 82 476
r 2.29 1 2 cbr 1000 -------- 2 1.0 3.1 272 473
+ 2.29 2 3 cbr 1000 -------- 2 1.0 3.1 272 473
- 2.291506 2 3 cbr 1000 -------- 2 1.0 3.1 272 473
+ 2.292 1 2 cbr 1000 -------- 2 1.0 3.1 274 477
- 2.292 1 2 cbr 1000 -------- 2 1.0 3.1 274 477
```

可以利用命令将感兴趣的数据提取出来

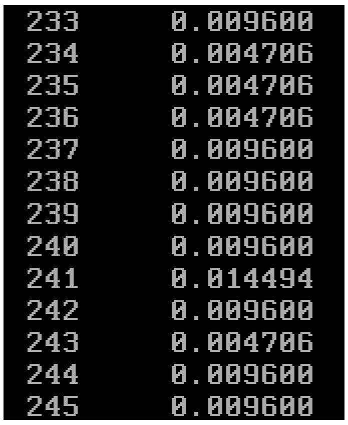
原文：`cat out.tr | grep " 2 3 cbr " | grep ^r | column 1 10 | awk '{dif = $2 - old2; if(dif==0) dif = 1; if(dif > 0) {printf("%d\t%f\n", $2, ($1 - old1) / dif); old1 = $1; old2 = $2}}' > jitter.txt`

我修改后的：

```
cat out.tr | grep " 2 3 cbr " | grep ^r | ./column 1 10 | awk '{dif = $2 - old2; if(dif==0) dif = 1; if(dif > 0) {printf("%d\t%f\n", $2, ($1 - old1) / dif); old1 = $1; old2 = $2}}' > jitter.txt
```

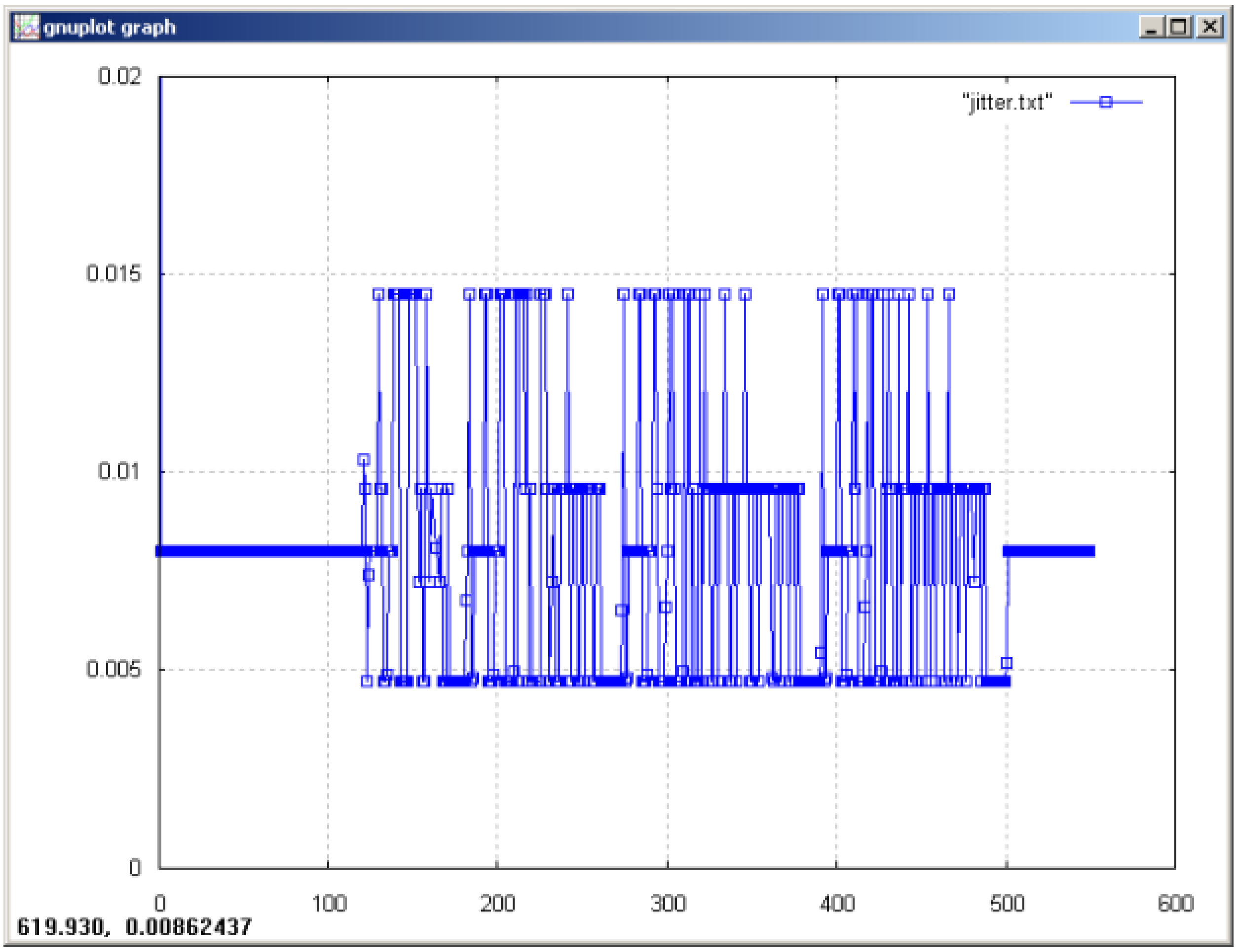
把 column 命令放在当前目录下了。

文件 jitter 的一部分：



```
233      0.009600
234      0.004706
235      0.004706
236      0.004706
237      0.009600
238      0.009600
239      0.009600
240      0.009600
241      0.014494
242      0.009600
243      0.004706
244      0.009600
245      0.009600
```

6.2.用 gnuplot 作图



6.3. RED Queue Monitor Example

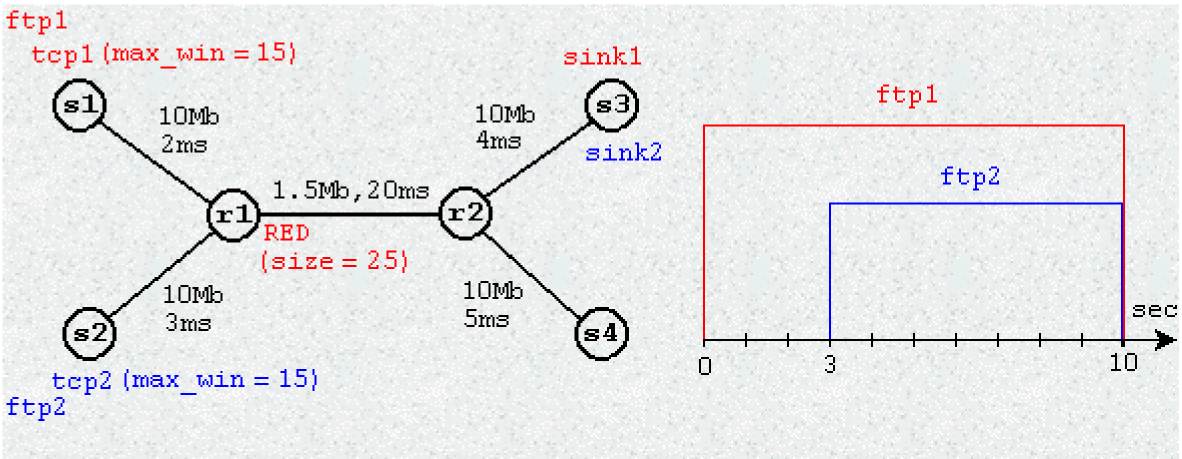


Figure 15. RED Queue Monitor Example Setup

```
set ns [new Simulator]

set node_(s1) [$ns node]
set node_(s2) [$ns node]
set node_(r1) [$ns node]
set node_(r2) [$ns node]
set node_(s3) [$ns node]
set node_(s4) [$ns node]

$ns duplex-link $node_(s1) $node_(r1) 10Mb 2ms DropTail
$ns duplex-link $node_(s2) $node_(r1) 10Mb 3ms DropTail
$ns duplex-link $node_(r1) $node_(r2) 1.5Mb 20ms RED
$ns queue-limit $node_(r1) $node_(r2) 25
$ns queue-limit $node_(r2) $node_(r1) 25
$ns duplex-link $node_(s3) $node_(r2) 10Mb 4ms DropTail
$ns duplex-link $node_(s4) $node_(r2) 10Mb 5ms DropTail
  .
```

```
$ns duplex-link-op $node_(s1) $node_(r1) orient right-down
$ns duplex-link-op $node_(s2) $node_(r1) orient right-up
$ns duplex-link-op $node_(r1) $node_(r2) orient right
$ns duplex-link-op $node_(r1) $node_(r2) queuePos 0
$ns duplex-link-op $node_(r2) $node_(r1) queuePos 0
$ns duplex-link-op $node_(s3) $node_(r2) orient left-down
$ns duplex-link-op $node_(s4) $node_(r2) orient left-up
```

```
  .
  .

set tcp1 [$ns create-connection TCP/Reno $node_(s1) TCPSink $node_(s3) 0]
$tcp1 set window_ 15
set tcp2 [$ns create-connection TCP/Reno $node (s2) TCPSink $node (s3) 1]
$tcp2 set window_ 15
set ftp1 [$tcp1 attach-source FTP]
set ftp2 [$tcp2 attach-source FTP]
```

其中，create-connection 是对象 ns 的一个高级成员函数

Window_ 是发送窗口的大小

这，没有使用 Agent/TCP 的形式定义，就可直接将数据发送源和接收源连接在一起。

? ? ?

函数的各个参数作用和位置？ ？ ？

```
# Tracing a queue
set redq [[$ns link $node_(r1) $node_(r2)] queue]
set tchan_ [open all.q w]
$redq trace curq_
$redq trace ave_
$redq attach $tchan_


$ns at 0.0 "$ftp1 start"
$ns at 3.0 "$ftp2 start"
$ns at 10 "finish"
```

定义了一个变量 redq 来跟踪队列的情况，并把当前排队长度和平均长度记录在文件 all.q 中。队列实际是第一个结点的输出的队列。

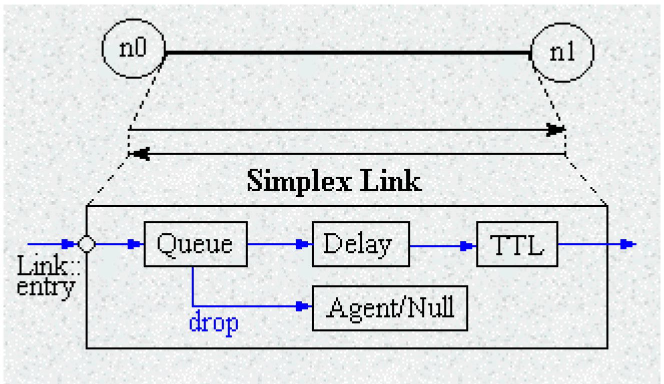
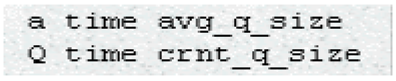
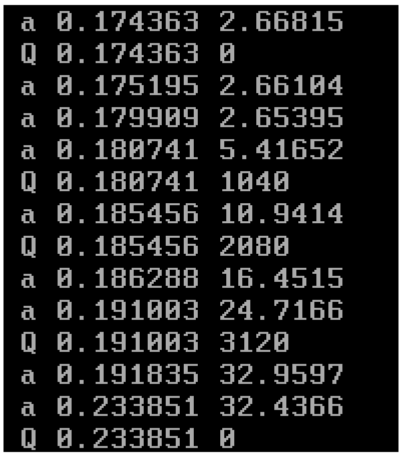


Figure 8. Link



```
a time avg_q_size
Q time crnt_q_size
```



```
a 0.174363 2.66815
Q 0.174363 0
a 0.175195 2.66104
a 0.179909 2.65395
a 0.180741 5.41652
Q 0.180741 1040
a 0.185456 10.9414
Q 0.185456 2080
a 0.186288 16.4515
a 0.191003 24.7166
Q 0.191003 3120
a 0.191835 32.9597
a 0.233851 32.4366
Q 0.233851 0
```

```
# Define 'finish' procedure (include post-simulation processes)
proc finish {} {
        global tchan_
        set awkCode {
            {
                if ($1 == "Q" && NF>2) {
                    print $2, $3 >> "temp.q";
                    set end $2
                }
                else if ($1 == "a" && NF>2)
                print $2, $3 >> "temp.a";
            }
        }
        set f [open temp.queue w]
        puts $f "TitleText: red"
        puts $f "Device: Postscript"

        if { [info exists tchan_] } {
            close $tchan_
        }
        exec rm -f temp.q temp.a
        exec touch temp.a temp.q

        exec awk $awkCode all.q

        puts $f \"queue
        exec cat temp.q >@ $f
        puts $f \n\"ave_queue
        exec cat temp.a >@ $f
        close $f
        exec xgraph -bb -tk -x time -y queue temp.queue &
        exit 0
}

$ns run
```

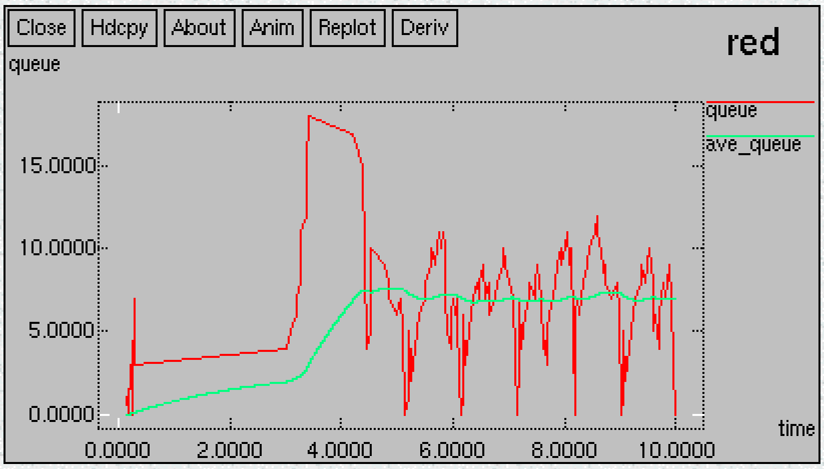


Figure 16. Red Queue Trace Graph

7. NS2 扩展功能

NS 是一个面向对象的模拟器，用 C++编写，且前端使用 OTCL 解释器。

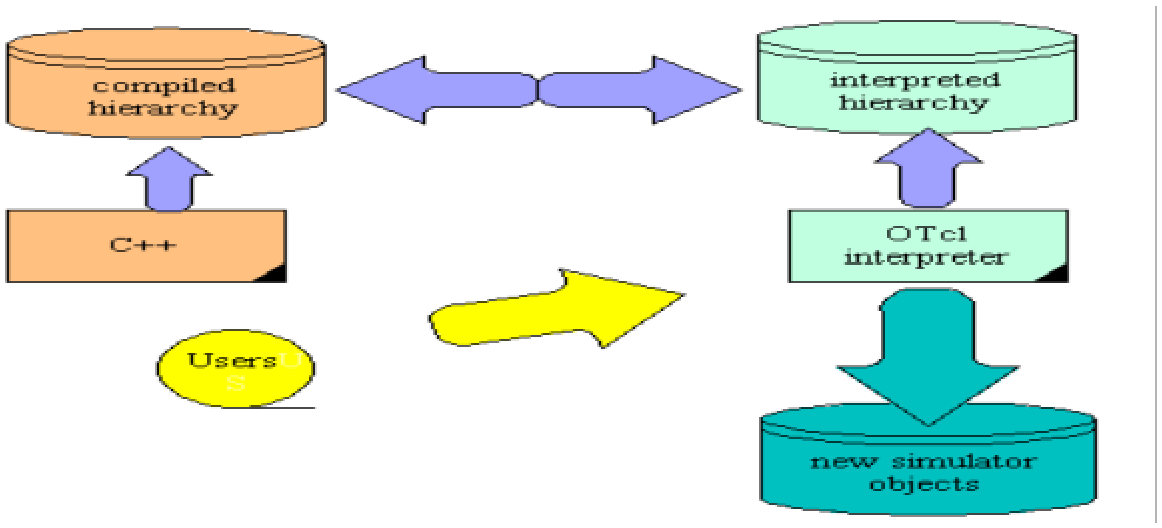
存在有两个类框架：

C++类框架：文档中称为编译的类编译框架。

OTcl 类框架：文档中称为解释的类框架。

两个类彼此紧密相关。从用户的观点，在两类中是一一对应的关系。在解释类中的基本父类是 TclObject，用户通过解释器来建立一个模拟对象；对象在解释器中创建并映射到编译类中去。

通过在 TclClass 中定义的方法建立了解释类的框架，在类 TclObject 中定义的方法来映射用户的使用的对象



使用二种语言的比较

语言	C++	OTcl
建立模拟器	Difficult configuration	Easy configuration
运行速度	fast	much slowe
修改代码	slower to change	Changed very quickly
适合完成的工作	anything that requires processing each packet of a flow	configuration, setup, ``one-time'' stuff

六个重要的类

类名	作用
Class TclSectionsec:Tcl	contains the methods that C++ code will use to access the interpreter

TclObjectSectionsec:TclObject	base class for all simulator objects that are also mirrored in the compiled hierarchy
TclClassSectionsec:TclClass	defines the interpreted class hierarchy, and the methods to permit the user to instantiate TclObjects.
TclCommandSectionsec:TclCommand	define simple global interpreter commands
EmbeddedTclSectionsec:EmbeddedTcl	contains the methods to load higher level builtin commands that make configuring simulations easier
InstVarSectionsec:InstVar	contains methods to access C++ member variables as OTcl instance variables.

7.1.NS 的目录结构

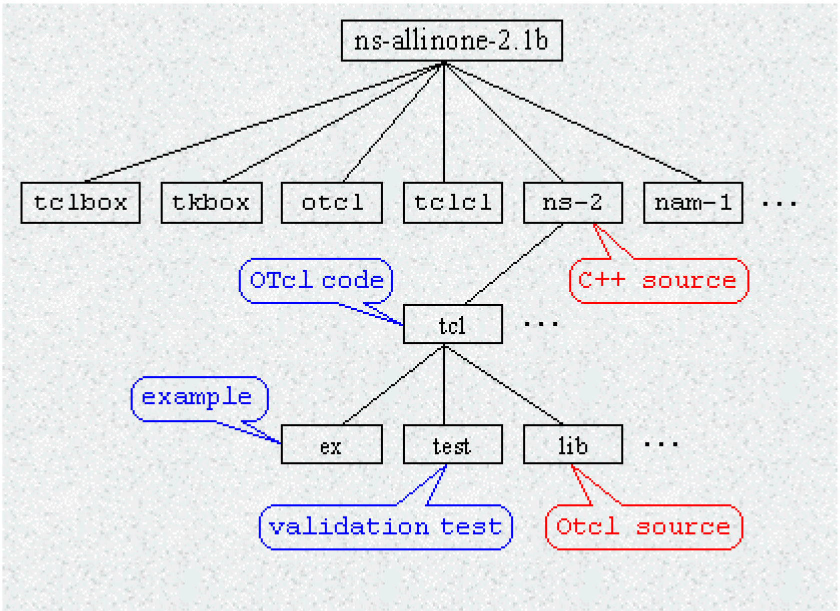


Figure 17. NS Directory Structure

NS-2： most of C++ code, which implements event scheduler and basic network component object classes 在 NS-2 这一层的目录中。如 UPD 协议的定义 upd.h 和 upd.cc 等。

Tcl 目录中的 lib 目录：

用户和开发人员，最常用的目录之一。agent, node, link, packet, address, routing, 等等重要的 NS 组件均在此目录下。

- ns-lib.tcl： The simulator class and most of its member function definitions

  except for LAN, Web, and Multicast related ones are located here. If you want to know which member functions of the Simulator object class are available and how they work, this is a place to look.

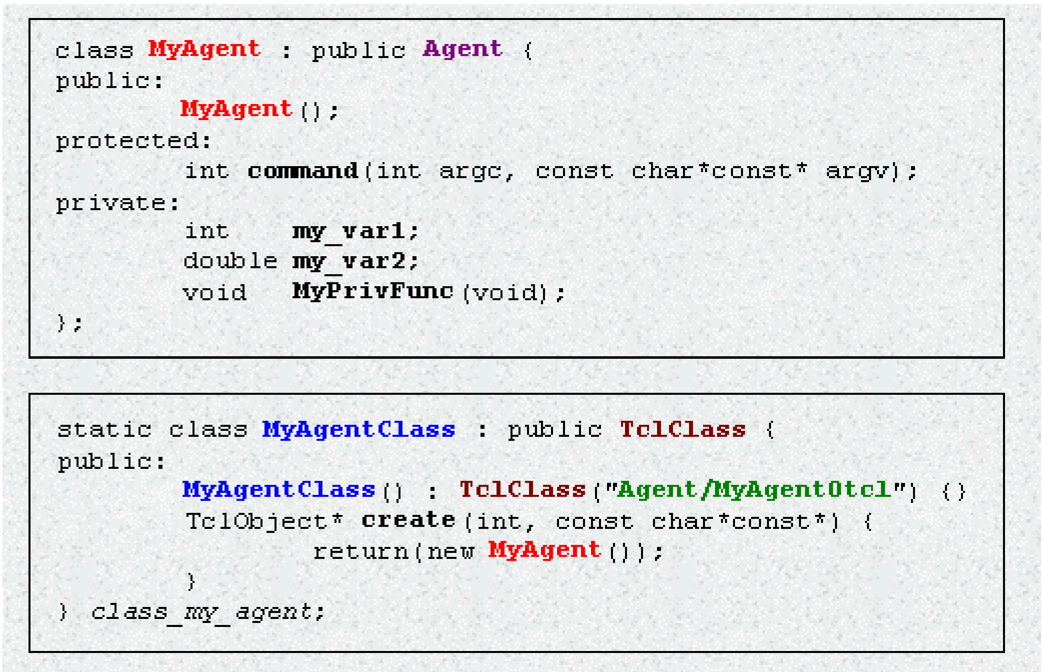
- ns-default.tcl: The default values for configurable parameters for various network components are located here. Since most of network components are implemented in C++, the configurable parameters are actually C++ variables made available to OTcl via an OTcl linkage function, bind(*C++_variable_name*, *OTcl_variable_name*). How to make OTcl linkages from C++ code is described in the next section.

- ns-packet.tcl: The packet header format initialization implementation is located here. When you create a new packet header, you should register the header in this file to make the packet header initialization process to include your header into the header stack format and give you the offset of your header in the stack. A new header creating example is shown in "Add New Application and Agent" section.

- other OTcl files: Other OTcl files in this directory, contain OTcl implementation of compound network objects or the front end (control part) of network objects in C++. The FTP application is entirely implemented in OTcl and the source code is located in "ns-source.tcl".

7.2. 扩展的方法

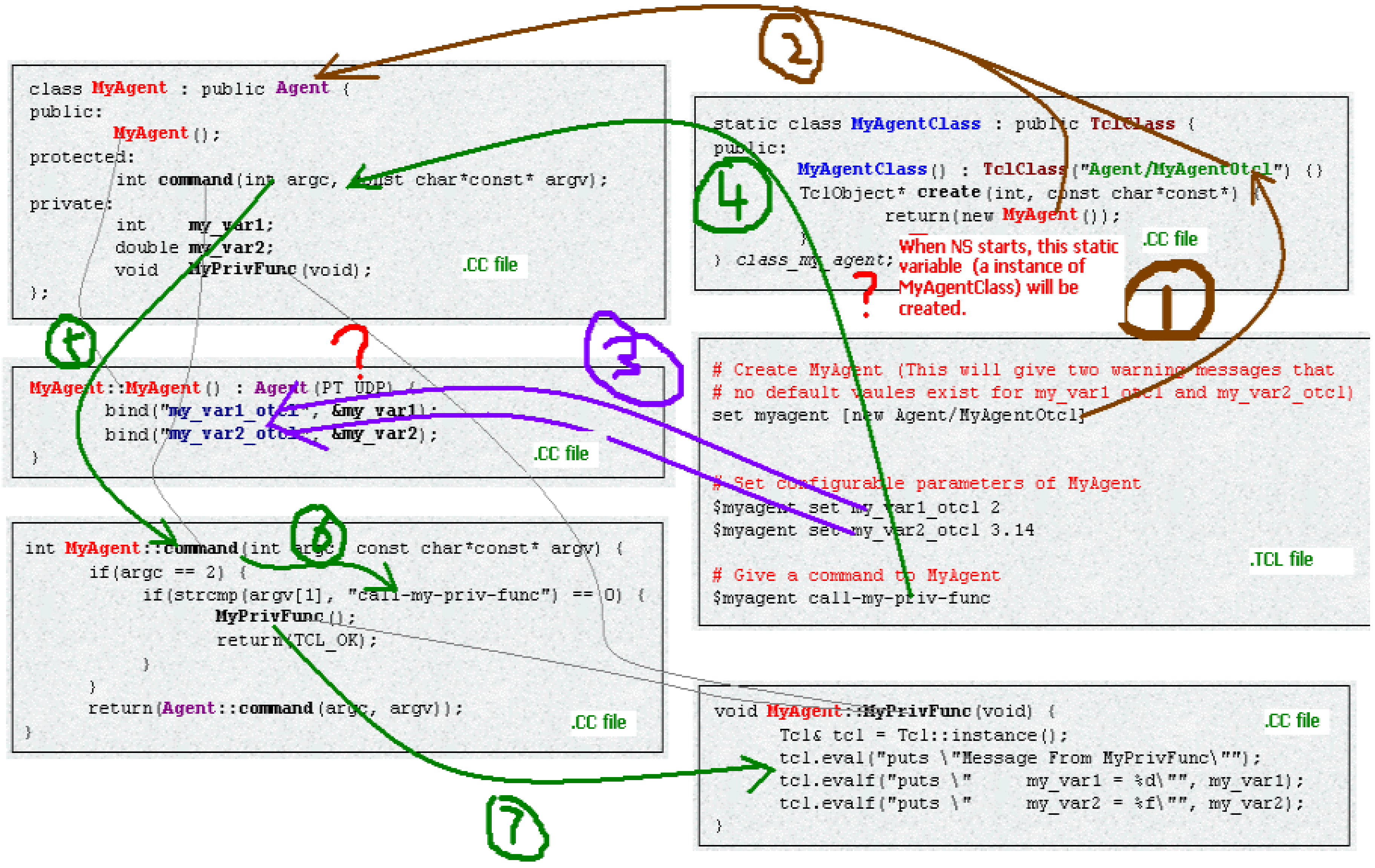
7.2.1. 第一步:Export C++ Class to Tcl



```
class MyAgent : public Agent {
public:
        MyAgent();
protected:
        int command(int argc, const char*const* argv);
private:
        int     my_var1;
        double  my_var2;
        void    MyPrivFunc(void);
};
```

```
static class MyAgentClass : public TclClass {
public:
        MyAgentClass() : TclClass("Agent/MyAgentOtcl") {}
        TclObject* create(int, const char*const*) {
                return(new MyAgent());
        }
} class_my_agent;
```

Figure 18. Example C++ Network Component and The Linkage Object



```
class MyAgent : public Agent {
public:
        MyAgent();
protected:
        int command(int argc, const char*const* argv);
private:
        int     my_var1;
        double  my_var2;
        void    MyPrivFunc(void);
};
```

```
MyAgent::MyAgent() : Agent(PT_UDP) {
       bind("my_var1_otcl", &my_var1);
       bind("my_var2_otcl", &my_var2);
}
```

```
int MyAgent::command(int argc, const char*const* argv) {
      if(argc == 2) {
           if(strcmp(argv[1], "call-my-priv-func") == 0) {
                  MyPrivFunc();
                  return(TCL_OK);
           }
      }
      return(Agent::command(argc, argv));
}
```

```
static class MyAgentClass : public TclClass {
public:
        MyAgentClass() : TclClass("Agent/MyAgentOtcl") {}
        TclObject* create(int, const char*const*) {
                return(new MyAgent());
        }
} class_my_agent;
```

When NS starts, this static variable (a instance of MyAgentClass) will be created.

```
# Create MyAgent (This will give two warning messages that
# no default vaules exist for my_var1_otcl and my_var2_otcl)
set myagent [new Agent/MyAgentOtcl]

# Set configurable parameters of MyAgent
$myagent set my_var1_otcl 2
$myagent set my_var2_otcl 3.14

# Give a command to MyAgent
$myagent call-my-priv-func
```

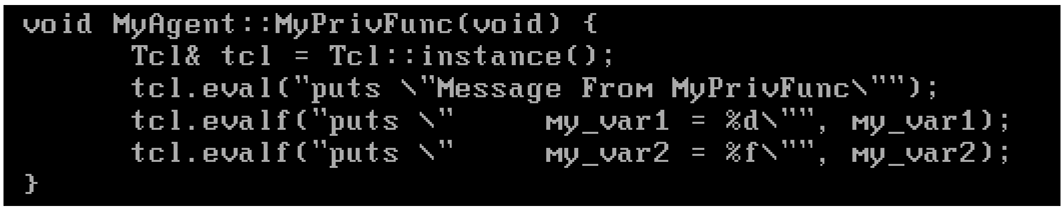
```
void MyAgent::MyPrivFunc(void) {
     Tcl& tcl = Tcl::instance();
     tcl.eval("puts \"Message From MyPrivFunc\"");
     tcl.evalf("puts \"     my_var1 = %d\"", my_var1);
     tcl.evalf("puts \"     my_var2 = %f\"", my_var2);
}
```

Class Myagent 是一个新的类，是从 NS 中的 C＋＋类中 Agent 继承而来的。

为了能让 OTcl 使用这个类定义的对象，我们又定义了一个从 TclClass 中继承而来的新类 MyAgentClass.这个类的作用就是让 OTCL 可以使用类 MyAgent，作用相当于一个桥梁。它将定义一个名为 Agent/MyAgent0tcl 的对象。Create 成员函数将完成桥梁的搭建。

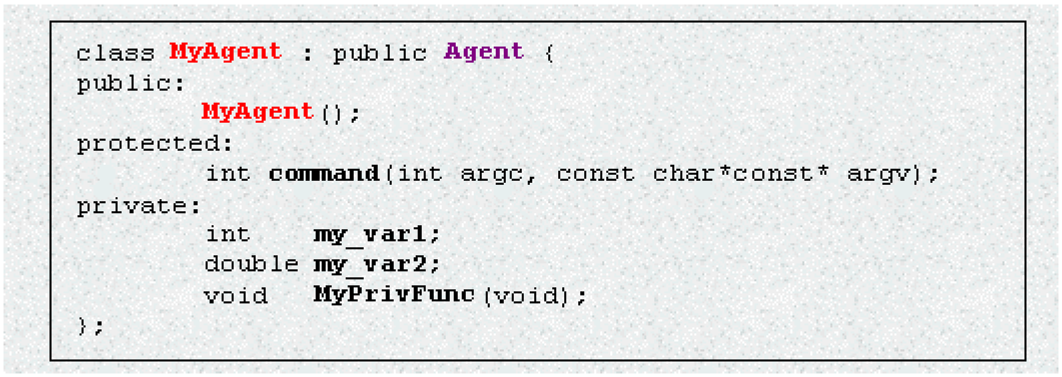
当 NS 一开始运行，当 class_my_agent 对象的实例创建时，构建函数 **myAgentClass( )**，当会运行。在上例中，一个类名的 Agent/MyAgent0tcl 的类及其成员函数会在 OTcl 的空间产生。

当使用命令 New Agent/MyAgent0tcl 时，成员函数 **create( )** 将会创建 MyAgent 的类的一个实例，并返回它的函数指针。


```
void MyAgent::MyPrivFunc(void) {
       Tcl& tcl = Tcl::instance();
       tcl.eval("puts \"Message From MyPrivFunc\"");
       tcl.evalf("puts \"     my_var1 = %d\"", my_var1);
       tcl.evalf("puts \"     my_var2 = %f\"", my_var2);
}
```

图表 10

7.2.2.Export C++ class variables to OTcl


```
class MyAgent : public Agent {
public:
        MyAgent();
protected:
        int command(int argc, const char*const* argv);
private:
        int    my_var1;
        double my_var2;
        void   MyPrivFunc(void);
};
```

MyAgent 有两个变量和一个成员函数。

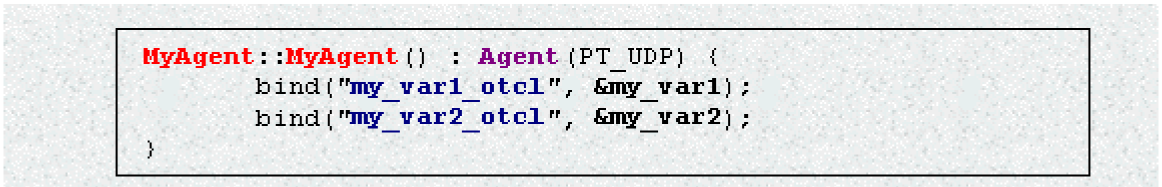
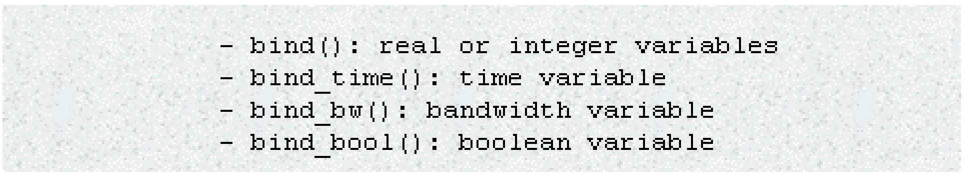

```
MyAgent::MyAgent() : Agent(PT_UDP) {
       bind("my_var1_otcl", &my_var1);
       bind("my_var2_otcl", &my_var2);
}
```

Figure 19. Variable Binding Creation Example

为了能让 OTCL 使用 C++的变量，我们使用绑定函数 bind( ) 。这个函数的第一个参数是在 OTCL 空间中的变量，第二个参数是 C++类中的成员变量。这是一个双向绑定过程，指向同一个地址空间。



```
- bind(): real or integer variables
- bind_time(): time variable
- bind_bw(): bandwidth variable
- bind_bool(): boolean variable
```

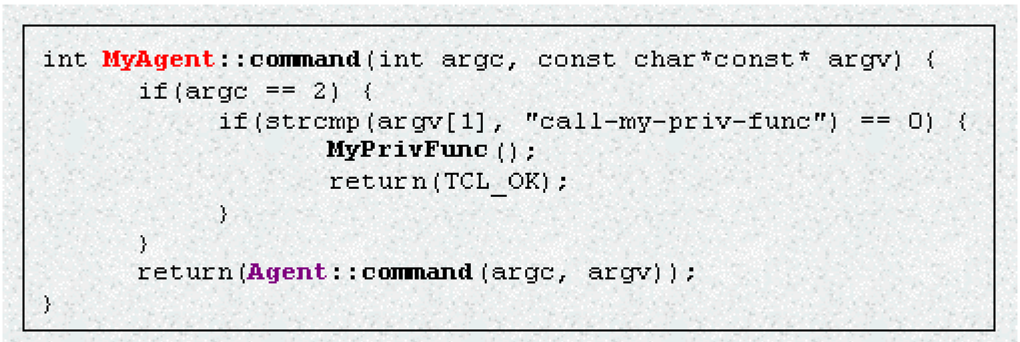
如上所示，总体上，Bind ( )共有四种不同的用法。

注意：要对变量进行初始化，在文件 ns-2/tcl/lib/ns-lib.tcl 中

Note that whenever you export a C++ variable, it is recommended that you also set the default value for that variable in the "ns-2/tcl/lib/ns-lib.tcl " file.

7.2.3. Export C++ Object Control Commands to OTcl.

为了能让 OTCL 对 C++对象进行控制，我们要对 command 函数进行编写。



```
int MyAgent::command(int argc, const char*const* argv) {
        if(argc == 2) {
              if(strcmp(argv[1], "call-my-priv-func") == 0) {
                       MyPrivFunc();
                       return(TCL_OK);
              }
        }
        return(Agent::command(argc, argv));
}
```

Figure 20. Example OTcl command interpreter

Command 函数相当于 OTCL 的接口函数。虽然是在 C++中定义的成员函数，但对用户来说，可以认为是在 OTCL 中的一个成员函数。

当程序执行

`set myagent [new Agent/MyAgentOtcl]`),

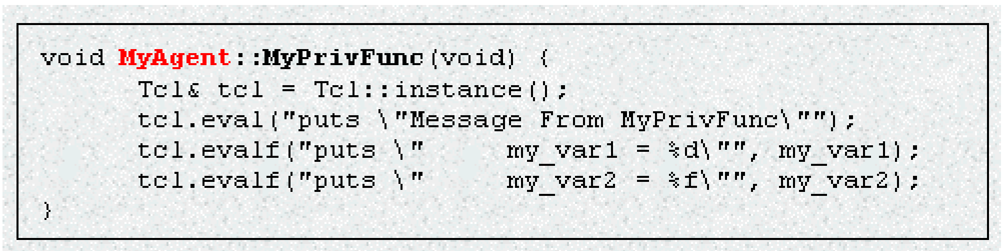
的时候，一个 myagent 类的对象实例就被建立起来了。当执行

`$myagent call-my-priv-func`

时，NS 首先在 OTCL 自身空间中寻找这个 “$myagent call-my-priv-func ” 函数，如果没有。则进入 C++空间中的对象中找 MyAgent::command( )函数，找到后；将函数的名字和参数传递过去。即是：int argc, const char * const * argv 。这样，在 command 函数中，如果函数名称匹配，将会调用 C++++空间中的成员函数。如果因某种原因，这个函数的名称也没有匹配的，则一个缺省的函数 Agent::command(argc, argv)会返回错误的信息给用户。至此，用户就可以间接地使用在 C++中定义的函数。

7.2.4.Execute an OTcl command from C++.

如果，想在 C++的类中，使用 OTCL 空间的命令，则需要使用下面的方法：



```
void MyAgent::MyPrivFunc(void) {
        Tcl& tcl = Tcl::instance();
        tcl.eval("puts \"Message From MyPrivFunc\"");
        tcl.evalf("puts \"     my_var1 = %d\"", my_var1);
        tcl.evalf("puts \"     my_var2 = %f\"", my_var2);
}
```

Figure 21. Execute OTcl command from a C++ Object

可以看到，我们先得到一个 tcl 的实例。

7.2.5.编译运行

A．编写好“ex-linkage.CC”文件，并放在 NS-2 的目录下面。

B．打开 NS-2 目录下的文件名为 Makefile 文件，把“ex-linkage.o ”加到文件最后。

C．使用 Make 命令，进行重新编译

D．可以运行使用新的命令的 TCL 程序了"ns ex-linkage.tcl". 。

8. NS 学习-增加一个新的应用(上)

8.1.目标

建立一个影像媒体应用数据流，可以根据当前网络的拥塞状况，可以自适应的 (分为 0-4，共五个级别)调整发送的数据量的大小。

8.2.应用描述

发送方与接收方，共同定义了五组(0-4) 方案，每一种方案有不同的编码与传输策略。在每一组方案和不考虑编码方案的下，传输的速率是不变的，且每个数据包的大小是固定的。

工作方式如下：(发送方为 A，接收方为 B)

例如：初始时，发送方 A 以组 0 方案进行发送数据，当接收方 B 认为网络的拥塞状况严重时，发送给 A 一个数据包，要求以组 2 方案发送数据，即是降低一倍。如果 B 认为网络又拥塞状况改善时，发给 A 一个数据包，要求以组 1 方案发送数据，即是提高一档。

每隔一段时间，B 会发送 A 一个数据包，要求 A 以某个方案发送影像数据。

8.3.问题分析

因为在实际的应用中，UDP 数据流是和应用密切相关的。我们应当让 UDP 端可以应用中接收数据，并把数据打包，然后进行发送。而在 NS 中缺省的 UDP 只使用仅有包头的数据

包。这一点是不满足我们的要求的。所以，要对 UDP 部分进行一定的修改，让其有这个机制能够处理这个问题。

而且，这个实验对于以后对 IP 路由排队研究的深入做一定的基础学习。这样的话，我们就要在 IP 头部分记录数据的类型。

8.4. NS 中所需文件的位置

packet.h	in	"common/packet.h",
agent.h	in	"common/agent.h"
app.h	in	"apps/app.h".

the locations of OTcl files :

"tcl/lib/ns-packet.tcl", "tcl/lib/ns-default.tcl" and etc

8.5. 设计与实现：

对于这个例子，我们修改一个 CBR 的例子使其具有 0-4 组方案的传输。

1、先构造一个 C++类，MmApp，从 Application 继承而来。应用层的发送和接收均在此类中完成

2、在 OTCL 空间中的映射为 Application/MmApp

3、在构造一个 C++类名 UdpMmAgent 从 UdpAgent 中继承，传输层的发送和接收在此类中完成。

4、在 OTCL 空间中的映射为 Agent/UDP/UDPmm

8.5.1. 定义 MmApp Header， 一个新的 UDP 数据包

1、应用层有信息传送时，应用层把信息按“hdr_mm”的数据结构，组织好数据，传递给 UdpMmAgent。

2、UdpMmAgent 把分配一个或多个数据包，然后把数据写入每个数据包的多媒体头 hdr_mm， 这个是在传输层中使用的。

3、在 C++ 类中叫 hdr_mm， 在 OTCL 空间中叫 MulitmediaHeaderClass ， 从 PacketHeaderClass 中继承而来。

```
// Multimedia Header Structure
struct hdr_mm {
        int ack;      // is it ack packet?
        int seq;      // mm sequence number
        int nbytes;   // bytes for mm pkt
        double time;  // current time
        int scale;    // scale (0-4) associated with data rates

        // Packet header access functions
        static int offset_;
        inline static int& offset() { return offset_; }
        inline static hdr_mm* access(const Packet* p) {
                return (hdr_mm*) p->access(offset_);
        }
};
```

```
// Multimedia Header Class
static class MultimediaHeaderClass : public PacketHeaderClass {
public:
 MultimediaHeaderClass() : PacketHeaderClass("PacketHeader/Multimedia",
                                              sizeof(hdr_mm)) {
                bind_offset(&hdr_mm::offset_);
        }
} class_mmhdr;
```

Figure 23. MM Header Structure & Class (in "udp-mm.h" & "udp-mm.cc")

8.5.2.在 NS 中注册

```
enum packet_t {
        PT_TCP,
        ...
        PT_Multimedia,
        PT_NTYPE // This MUST be the LAST one
};

class p_info {
public:
        p_info() {
                name_[PT_TCP]= "tcp";
                ...
                name_[PT_Multimedia] = "Multimedia";
                name_[PT_NTYPE]= "undefined";
        }
        ...
};
```

Figure 24 (a). Add to the "packet.h" file (C++)

```
foreach prot {
        AODV
        ...
        Multimedia
} {
        add-packet-header $prot
}
```

Figure 24 (b). Add to the "ns-packet.tcl" file (Otcl)

这样，我们就可通过 hdr_mm::acess( )成员函数来访问这个数据包的头，这个成员函数是从上级的类中继承来的。

8.5.3.MmApp Sender 发送端

发送端使用一个定时器来调度下一个应用数据包的传输。

```
class SendTimer : public TimerHandler {
 public:
        SendTimer(MmApp* t) : TimerHandler(), t_(t) {}
        inline virtual void expire(Event*);
 protected:
        MmApp* t_;
};

void SendTimer::expire(Event*)
{
  t_->send_mm_pkt();
}
```

```
class MmApp : public Application {
public:
        MmApp();
        ...
private:
        ...
        SendTimer snd_timer_;
        ...
};
```

从上面的程序中，可以看到 timer 中 expire 函数会调用发送函数。

Before setting this timer, "MmApp" re-calculates the next transmission time using the transmission rate associated with the current scale value and the size of application data packet that is given in the input simulation script (or use the default size). The "MmApp" sender, when an ACK application packet arrives from the receiver side, updates the scale parameter.

也就是说，在定时器启动前，下一个发送的时刻，是由传送速率，当前方案值 0-4，以及应用数据包的大小来决定的。当第一个接收端的 ACK 包回来后，发送端会根据返回的值来决定当前方案值 0-4。

8.5.4.MmApp Receiver 接收端

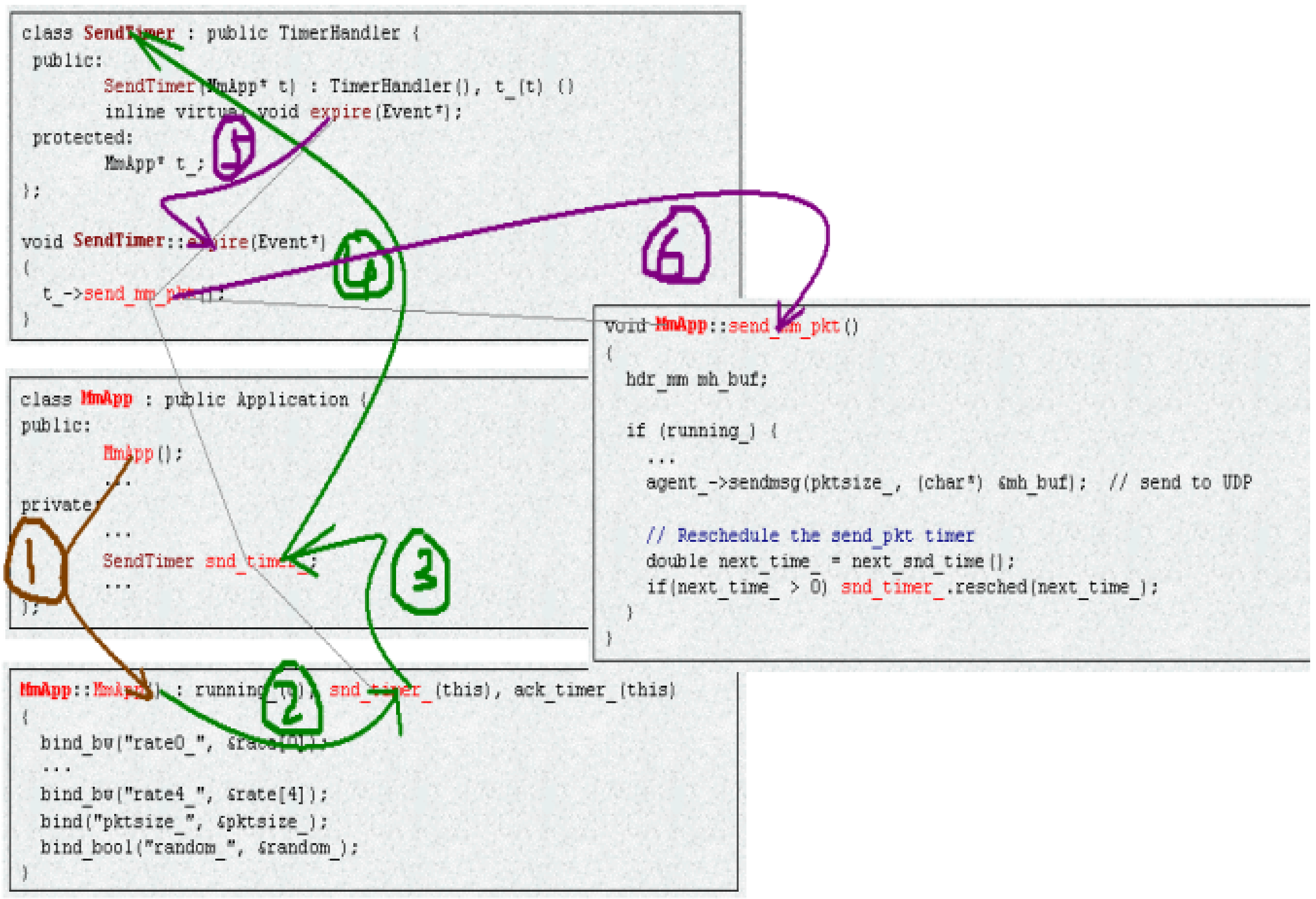
接收端也有一个定时器，来调度下一个 ACK 数据包的发送时间。通常是根据 mean RTT 也就是平均时间来确定。

同时接收端在应用层要使用两个计数器来统计**接到数据**包个数和**丢失的数据**包个数。

根据上面计数器的信息来决定发送给发送端哪一种方案。

当定时到，发送 ACK 数据包；同时，到计数器清零。

由于没有关闭信息，接收到会每隔一段时间就发送一个 ACK，不会停止。(这是个 Bug)



```
class SendTimer : public TimerHandler {
 public:
        SendTimer(MmApp* t) : TimerHandler(), t_(t) {}
        inline virtual void expire(Event*);
 protected:
        MmApp* t_;
};

void SendTimer::expire(Event*)
{
  t_->send_mm_pkt();
}
```

```
class MmApp : public Application {
public:
        MmApp();
        ...
private:
        ...
        SendTimer snd_timer_;
        ...
};
```

```
MmApp::MmApp() : running_(0), snd_timer_(this), ack_timer_(this)
{
  bind_bw("rate0_", &rate[0]);
  ...
  bind_bw("rate4_", &rate[4]);
  bind("pktsize_", &pktsize_);
  bind_bool("random_", &random_);
}
```

```
void MmApp::send_mm_pkt()
{
  hdr_mm mh_buf;

  if (running_) {
    ...
    agent_->sendmsg(pktsize_, (char*) &mh_buf);  // send to UDP

    // Reschedule the send_pkt timer
    double next_time_ = next_snd_time();
    if(next_time_ > 0) snd_timer_.resched(next_time_);
  }
}
```

8.5.5.UdpMmAgent 的工作

"UdpMmAgent" 是由"UdpAgent"修改而来的。并有以下特征：

(1) writing to the sending data packet MM header the information received from a "MmApp" (or reading information from the received data packet MM header and handing it to the "MmApp"),

把从 MmApp 得到的信息写入将要发送的 MM 头中(或者是从收到的 MM 头中，取得数据传递给 MmApp

(2) segmentation and re-assembly ("UdpAgent" only implements segmentation), and

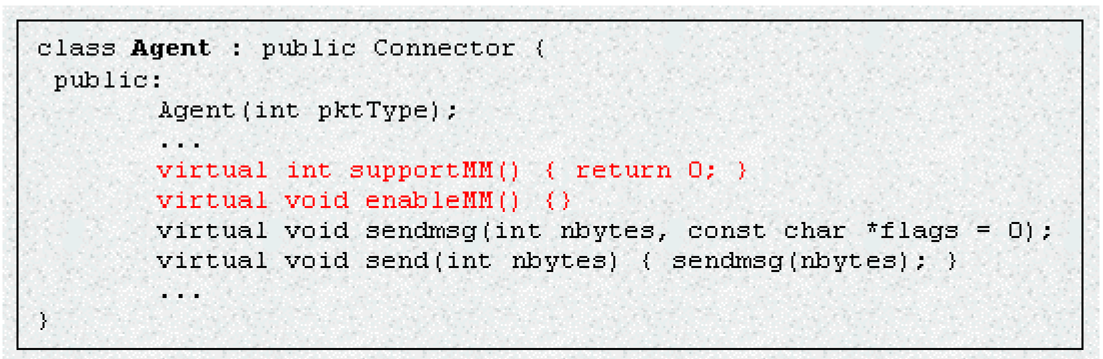
重新分割和打包，发送。

(3) setting the priority bit (IPv6) to 15 (max priority) for the "MmApp" packets.

设置优先级给 MmApp 包。

9. NS 学习-增加一个新的应用(下)

9.1. 修改 agent.h at Ns-2.x/common/agent.h



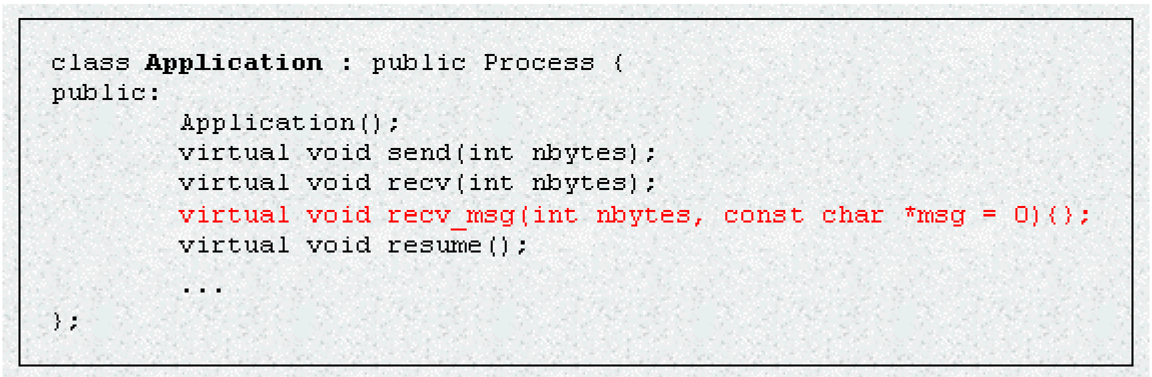
```
class Agent : public Connector {
 public:
        Agent(int pktType);
        ...
        virtual int supportMM() { return 0; }
        virtual void enableMM() {}
        virtual void sendmsg(int nbytes, const char *flags = 0);
        virtual void send(int nbytes) { sendmsg(nbytes); }
        ...
}
```

Figure 26 (a). Add two member functions to "Agent" class.

在 Class Agent 增加两个成员函数 supportMM( )和 enableMM( )来检查 Agent 是否支持 MM。在 class MmApp 中的 command 成员函数中，定义 attach agent 的 OTCL 命令。这个命令执行前，会调用 class Agent 中的成员函数 supportMM( )和 enableMM( )来检查 Agent 是否支持 MM。

注意，仅在 Class UdpMm Agent 中定义 supportMM( )和 enableMM( )是不够的，会出现编译的错误。

9.2. 修改 App.h in " Ns-2.x/apps/app.h”



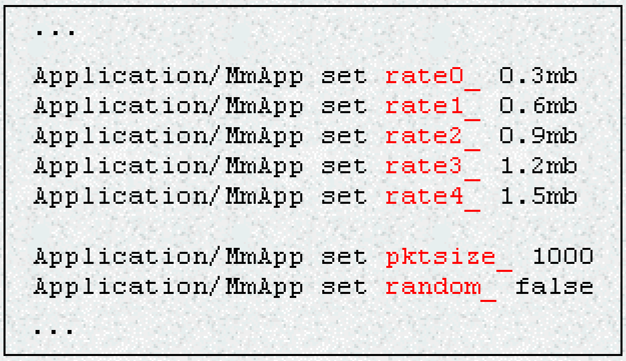
```
class Application : public Process {
public:
        Application();
        virtual void send(int nbytes);
        virtual void recv(int nbytes);
        virtual void recv_msg(int nbytes, const char *msg = 0){};
        virtual void resume();
        ...
};
```

Figure 26 (b). Add a member function to "Application" class.

这个函数 recv_msg(int nbytes, const char *msg = 0){}在早期的 NS 中是有的。后来又被去掉了。这个要根据你具体使用的 NS 而定。我们需要去检查一下 app.h 即可。

9.3. 在 OCTL 中设置缺省参数 "ns-default.tcl":

tcl/lib/ns-default.tcl



```
...
Application/MmApp set rate0_  0.3mb
Application/MmApp set rate1_  0.6mb
Application/MmApp set rate2_  0.9mb
Application/MmApp set rate3_  1.2mb
Application/MmApp set rate4_  1.5mb

Application/MmApp set pktsize_  1000
Application/MmApp set random_  false
...
```

Figure 27. Default parameter value settings

9.4. 编译 Compile

1. 把 "mm-app.h", "mm-app.cc", "udp-mm.h" and "udp-mm.cc" 放到 "ns-2" 目录下。
2. Make sure you registered the new application header by modifying "packet.h" and "ns-packet.tcl" as shown in Figure 24 (a) and (b).
3. Add supportMM() and enableMM() methods to the "Agent" class in "agent.h" as shown in Figure 26 (a).
4. Add recv_msg() method to the "Application" class in "app.h" as shown in Figure 26 (b).
5. Set default values for the newly introduced configurable parameters in "ns-default.tcl" as described in Figure 27. Be SURE to complete this last step. Otherwise, all five-scale rates are initialized to zero unless specified in the input simulation script (i.e., the test simulation script given below will not transmit any frames).

上面五步做好后，修改 Makefile 文件，把"mm-app.o" and "udp-mm.o" 放到 OBJ 文件名的最后，使用 make 重新编译 NS。

注意，在重新编译 NS 之前要运行 "make clean" 和 "make depend" 。否则，新应用不会传输任何数据包。所以，在修改 Makefile 后，运行 "make depend" 命令。

9.5. 测试

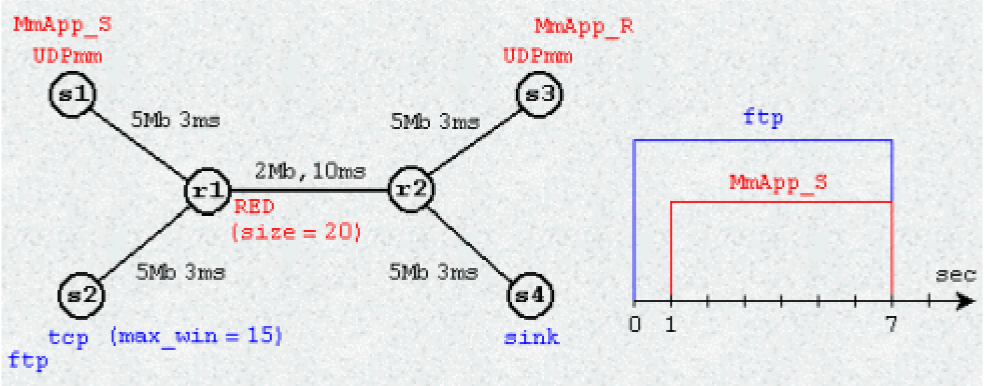
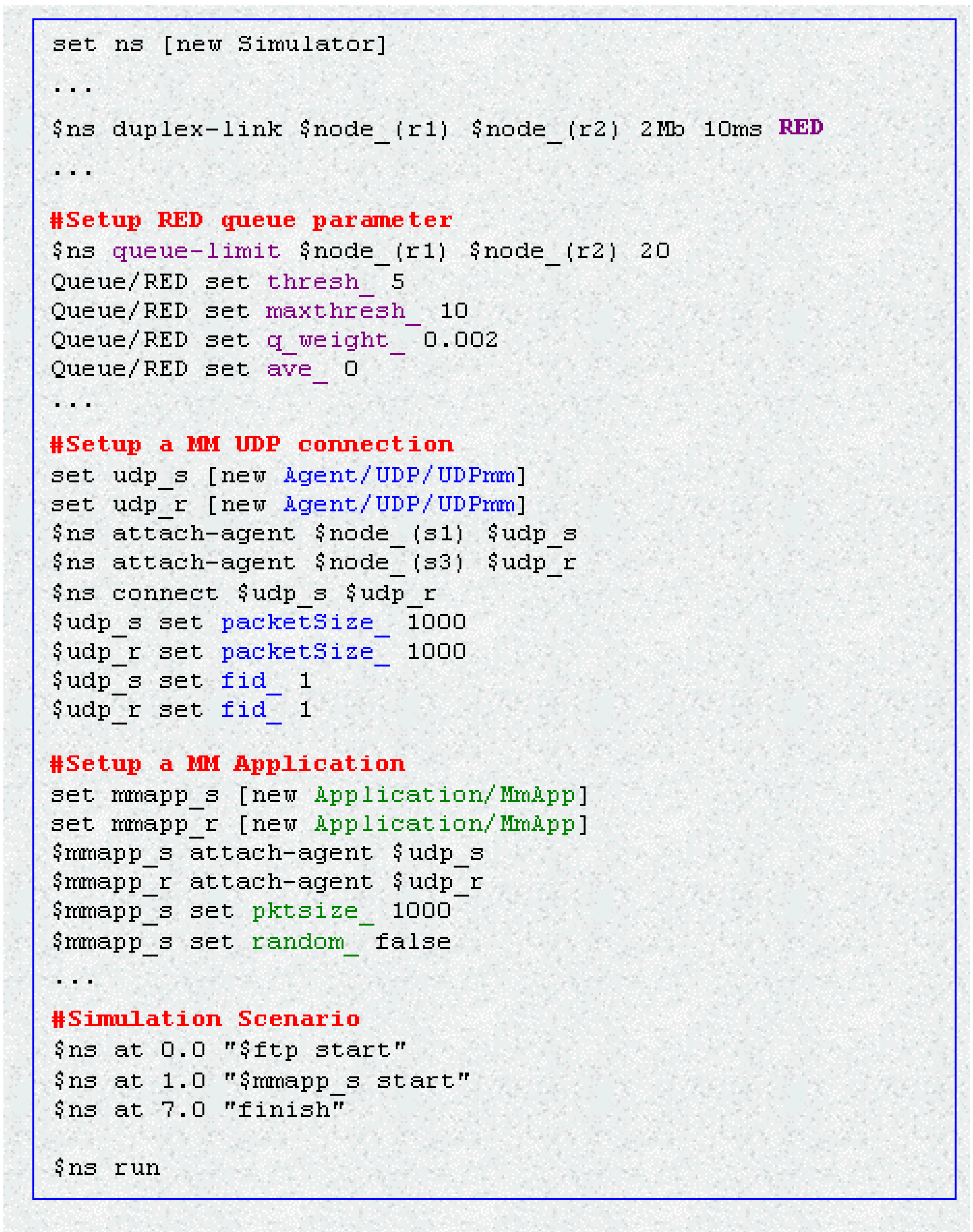


Figure 28. Test Simulation Topology and Scenario



```
set ns [new Simulator]
...
$ns duplex-link $node_(r1) $node_(r2) 2Mb 10ms RED
...
#Setup RED queue parameter
$ns queue-limit $node_(r1) $node_(r2) 20
Queue/RED set thresh_ 5
Queue/RED set maxthresh_ 10
Queue/RED set q_weight_ 0.002
Queue/RED set ave_ 0
...
#Setup a MM UDP connection
set udp_s [new Agent/UDP/UDPmm]
set udp_r [new Agent/UDP/UDPmm]
$ns attach-agent $node_(s1) $udp_s
$ns attach-agent $node_(s3) $udp_r
$ns connect $udp_s $udp_r
$udp_s set packetSize_ 1000
$udp_r set packetSize_ 1000
$udp_s set fid_ 1
$udp_r set fid_ 1

#Setup a MM Application
set mmapp_s [new Application/MmApp]
set mmapp_r [new Application/MmApp]
$mmapp_s attach-agent $udp_s
$mmapp_r attach-agent $udp_r
$mmapp_s set pktsize_ 1000
$mmapp_s set random_ false
...
#Simulation Scenario
$ns at 0.0 "$ftp start"
$ns at 1.0 "$mmapp_s start"
$ns at 7.0 "finish"

$ns run
```

Figure 29. "MmApp" Test Simulation Script

9.6.源文件

9.6.1.mm-App.h

```
//
// Author:    Jae Chung
// File:      mm-app.h
// Written:   07/17/99 (for ns-2.1b4a)
// Modifed:   10/14/01 (for ns-2.1b8a)
//

#include "timer-handler.h"
#include "packet.h"
#include "app.h"
#include "udp-mm.h"

// This is used for receiver's received packet accounting
```

```
//接收方定义的 ACK 返回信息
struct pkt_accounting {
       int last_seq;   // sequence number of last received MM pkt
       int last_scale; // rate (0-4) of last acked
       int lost_pkts;  // number of lost pkts since last ack
       int recv_pkts;  // number of received pkts since last ack
       double rtt;     // round trip time
};

//为何在此处就把 MmApp 写出来了？？？
class MmApp;


// 发送方的：发送定时器 Sender uses this timer to
// schedule next app data packet transmission time
class SendTimer : public TimerHandler {
 public:
    SendTimer(MmApp* t) : TimerHandler(), t_(t) {}
    inline virtual void expire(Event*);
 protected:
    MmApp* t_;
};


// 接收方的：定时器 Reciver uses this timer to schedule
// next ack packet transmission time
class AckTimer : public TimerHandler {
 public:
    AckTimer(MmApp* t) : TimerHandler(), t_(t) {}
    inline virtual void expire(Event*);
 protected:
    MmApp* t_;
};


// 可以理解为在应用层的工作：
//Mulitmedia Application Class Definition
class MmApp : public Application {
 public:
    MmApp();
    void send_mm_pkt();  // called by SendTimer:expire (Sender)
    void send_ack_pkt(); // called by AckTimer:expire (Receiver)
 protected:
    int command(int argc, const char*const* argv);
    void start();       // Start sending data packets (Sender)
    void stop();        // Stop sending data packets (Sender)
 private:
    void init();
    inline double next_snd_time();                          // (Sender)
    virtual void recv_msg(int nbytes, const char *msg = 0); // (Sender/Receiver)
    void set_scale(const hdr_mm *mh_buf);                   // (Sender)
    void adjust_scale(void);                                // (Receiver)
    void account_recv_pkt(const hdr_mm *mh_buf);            // (Receiver)
    void init_recv_pkt_accounting();                        // (Receiver)

    double rate[5];       // Transmission rates associated to scale values
    double interval_;     // Application data packet transmission interval
    int pktsize_;         // Application data packet size
    int random_;          // If 1 add randomness to the interval
    int running_;         // If 1 application is running
    int seq_;             // Application data packet sequence number
    int scale_;           // Media scale parameter
    pkt_accounting p_accnt;
    SendTimer snd_timer_;  // SendTimer
    AckTimer  ack_timer_;  // AckTimer
};
```

9.6.2.mm-app.cc

```
//
// Author:    Jae Chung
// File:      mm-app.cc
// Written:   07/17/99 (for ns-2.1b4a)
// Modifed:   10/14/01 (for ns-2.1b8a)
//

#include "random.h"
#include "mm-app.h"


// MmApp OTcl linkage class
static class MmAppClass : public TclClass {
 public:
  MmAppClass() : TclClass("Application/MmApp") {}
  TclObject* create(int, const char*const*) {
    return (new MmApp);
  }
} class_app_mm;


// When snd_timer_ expires call MmApp:send_mm_pkt()
void SendTimer::expire(Event*)
{
  t_->send_mm_pkt();
}


// When ack_timer_ expires call MmApp:send_ack_pkt()
void AckTimer::expire(Event*)
{
  t_->send_ack_pkt();
}


// Constructor (also initialize instances of timers)
MmApp::MmApp() : running_(0), snd_timer_(this), ack_timer_(this)
{
  bind_bw("rate0_", &rate[0]);
```

//绑定 bandwidth 变量

```
  bind_bw("rate1_", &rate[1]);
  bind_bw("rate2_", &rate[2]);
  bind_bw("rate3_", &rate[3]);
  bind_bw("rate4_", &rate[4]);
  bind("pktsize_", &pktsize_);
  bind_bool("random_", &random_);
}


// OTcl command interpreter
int MmApp::command(int argc, const char*const* argv)
{
  Tcl& tcl = Tcl::instance();

  if (argc == 3) {
    if (strcmp(argv[1], "attach-agent") == 0) {
      agent_ = (Agent*) TclObject::lookup(argv[2]);
      if (agent_ == 0) {
    tcl.resultf("no such agent %s", argv[2]);
    return(TCL_ERROR);
      }

      // Make sure the underlying agent support MM
      if(agent_->supportMM()) {
    agent_->enableMM();
      }
      else {
    tcl.resultf("agent \"%s\" does not support MM Application", argv[2]);
    return(TCL_ERROR);
      }

      agent_->attachApp(this);
      return(TCL_OK);
    }
  }
  return (Application::command(argc, argv));
}



void MmApp::init()
{
  scale_ = 0; // Start at minimum rate
  seq_ = 0;   // MM sequence number (start from 0)
  interval_ = (double)(pktsize_ << 3)/(double)rate[scale_];
}


void MmApp::start()
{
  init();
  running_ = 1;
  send_mm_pkt();
}


void MmApp::stop()
{
  running_ = 0;
}


// Send application data packet
void MmApp::send_mm_pkt()
{
  hdr_mm mh_buf;

  if (running_) {
    // the below info is passed to UDPmm agent, which will write it
    // to MM header after packet creation.
    mh_buf.ack = 0;           // This is a MM packet
    mh_buf.seq = seq_++;        // MM sequece number
    mh_buf.nbytes = pktsize_;  // Size of MM packet (NOT UDP packet size)
    mh_buf.time = Scheduler::instance().clock();  // Current time
    mh_buf.scale = scale_;                    // Current scale value
    agent_->sendmsg(pktsize_, (char*) &mh_buf);  // send to UDP

    // Reschedule the send_pkt timer
    double next_time_ = next_snd_time();
    if(next_time_ > 0) snd_timer_.resched(next_time_);
  }
}



// Schedule next data packet transmission time
double MmApp::next_snd_time()
{
  // Recompute interval in case rate or size chages
  interval_ = (double)(pktsize_ << 3)/(double)rate[scale_];
  double next_time_ = interval_;
  if(random_)
    next_time_ += interval_ * Random::uniform(-0.5, 0.5);
  return next_time_;
}


// Receive message from underlying agent
void MmApp::recv_msg(int nbytes, const char *msg)
{
  if(msg) {
    hdr_mm* mh_buf = (hdr_mm*) msg;

    if(mh_buf->ack == 1) {
      // If received packet is ACK packet
      set_scale(mh_buf);
    }
    else {
      // If received packet is MM packet
      account_recv_pkt(mh_buf);
      if(mh_buf->seq == 0) send_ack_pkt();
    }
  }
}


// Sender sets its scale to what reciver notifies
void MmApp::set_scale(const hdr_mm *mh_buf)
{
  scale_ = mh_buf->scale;
}


void MmApp::account_recv_pkt(const hdr_mm *mh_buf)
{
  double local_time = Scheduler::instance().clock();

  // Calculate RTT
  if(mh_buf->seq == 0) {
    init_recv_pkt_accounting();
    p_accnt.rtt = 2*(local_time - mh_buf->time);
  }
  else
    p_accnt.rtt = 0.9 * p_accnt.rtt + 0.1 * 2*(local_time - mh_buf->time);

  // Count Received packets and Calculate Packet Loss
  p_accnt.recv_pkts ++;
  p_accnt.lost_pkts += (mh_buf->seq - p_accnt.last_seq - 1);
  p_accnt.last_seq = mh_buf->seq;
}
```

```
void MmApp::init_recv_pkt_accounting()
{
  p_accnt.last_seq = -1;
  p_accnt.last_scale = 0;
  p_accnt.lost_pkts = 0;
  p_accnt.recv_pkts = 0;
}


void MmApp::send_ack_pkt(void)
{
  double local_time = Scheduler::instance().clock();

  adjust_scale();

  // send ack message
  hdr_mm ack_buf;
  ack_buf.ack = 1;  // this packet is ack packet
  ack_buf.time = local_time;
  ack_buf.nbytes = 40;  // Ack packet size is 40 Bytes
  ack_buf.scale = p_accnt.last_scale;
  agent_->sendmsg(ack_buf.nbytes, (char*) &ack_buf);

  // schedul next ACK time
  ack_timer_.resched(p_accnt.rtt);
}


void MmApp::adjust_scale(void)
{
  if(p_accnt.recv_pkts > 0) {
    if(p_accnt.lost_pkts > 0)
      p_accnt.last_scale = (int)(p_accnt.last_scale / 2);
    else {
      p_accnt.last_scale++;
      if(p_accnt.last_scale > 4) p_accnt.last_scale = 4;
    }
  }
  p_accnt.recv_pkts = 0;
  p_accnt.lost_pkts = 0;
}
```

9.6.3.udp-mm.h

```
//
// Author:    Jae Chung
// File:      udp-mm.h
// Written:   07/17/99 (for ns-2.1b4a)
// Modifed:   10/14/01 (for ns-2.1b8a)
//

#ifndef ns_udp_mm_h
#define ns_udp_mm_h

#include "udp.h"
#include "ip.h"

// Multimedia Header Structure
struct hdr_mm {
    int ack;    // is it ack packet?
    int seq;     // mm sequence number
    int nbytes;  // bytes for mm pkt
    double time; // current time
    int scale;   // scale (0-4) associated with data rates

    // Packet header access functions
       static int offset_;
       inline static int& offset() { return offset_; }
       inline static hdr_mm* access(const Packet* p) {
              return (hdr_mm*) p->access(offset_);
       }
};


// Used for Re-assemble segmented (by UDP) MM packet
struct asm_mm {
    int seq;     // mm sequence number
    int rbytes;  // currently received bytes
    int tbytes;  // total bytes to receive for MM packet
};


// UdpMmAgent Class definition
class UdpMmAgent : public UdpAgent {
public:
    UdpMmAgent();
    UdpMmAgent(packet_t);
    virtual int supportMM() { return 1; }
    virtual void enableMM() { support_mm_ = 1; }
    virtual void sendmsg(int nbytes, const char *flags = 0);
    void recv(Packet*, Handler*);
protected:
    int support_mm_; // set to 1 if above is MmApp
private:
    asm_mm asm_info; // packet re-assembly information
};

#endif
```

9.6.4. udp-mm.cc

```
//
// Author:   Jae Chung
// File:     udp-mm.cc
// Written:  07/17/99 (for ns-2.1b4a)
// Modifed:  10/14/01 (for ns-2.1b8a)
//

#include "udp-mm.h"
#include "rtp.h"
#include "random.h"
#include <string.h>


int hdr_mm::offset_;

// Mulitmedia Header Class
static class MultimediaHeaderClass : public PacketHeaderClass {
public:
    MultimediaHeaderClass() : PacketHeaderClass("PacketHeader/Multimedia",
                             sizeof(hdr_mm)) {
```

```
        bind_offset(&hdr_mm::offset_);
    }
} class_mmhdr;


// UdpMmAgent OTcl linkage class
static class UdpMmAgentClass : public TclClass {
public:
    UdpMmAgentClass() : TclClass("Agent/UDP/UDPmm") {}
    TclObject* create(int, const char*const*) {
        return (new UdpMmAgent());
    }
} class_udpmm_agent;


// Constructor (with no arg)
UdpMmAgent::UdpMmAgent() : UdpAgent()
{
    support_mm_ = 0;
    asm_info.seq = -1;
}

UdpMmAgent::UdpMmAgent(packet_t  type) : UdpAgent(type)
{
    support_mm_ = 0;
    asm_info.seq = -1;
}


// Add Support of Multimedia Application to UdpAgent::sendmsg
void UdpMmAgent::sendmsg(int  nbytes, const char* flags)
{
    Packet *p;
    int n, remain;


    if (size_) {
        n = (nbytes/size_ + (nbytes%size_ ? 1 : 0));
        remain = nbytes%size_;
    }
    else
        printf("Error: UDPmm size = 0\n");

    if (nbytes == -1) {
        printf("Error:  sendmsg() for UDPmm should not be -1\n");
        return;
    }
    double local_time =Scheduler::instance().clock();
    while (n-- > 0) {
        p = allocpkt();
        if(n==0 && remain>0) hdr_cmn::access(p)->size() = remain;
        else hdr_cmn::access(p)->size() = size_;
        hdr_rtp* rh = hdr_rtp::access(p);
        rh->flags() = 0;
        rh->seqno() = ++seqno_;
        hdr_cmn::access(p)->timestamp() =
            (u_int32_t)(SAMPLERATE*local_time);
        // to eliminate recv to use MM fields for non MM packets
        hdr_mm* mh = hdr_mm::access(p);
        mh->ack = 0;
        mh->seq = 0;
        mh->nbytes = 0;
        mh->time = 0;
        mh->scale = 0;
        // mm udp packets are distinguished by setting the ip
        // priority bit to 15 (Max Priority).
        if(support_mm_) {
            hdr_ip* ih = hdr_ip::access(p);
            ih->prio_ = 15;
            if(flags) // MM Seq Num is passed as flags
                memcpy(mh, flags, sizeof(hdr_mm));
        }
        // add "beginning of talkspurt" labels (tcl/ex/test-rcvr.tcl)
        if (flags && (0 ==strcmp(flags, "NEW_BURST")))
            rh->flags() |= RTP_M;
        target_->recv(p);
    }
    idle();
}


// Support Packet Re-Assembly and Multimedia Application
void UdpMmAgent::recv(Packet* p, Handler*)
{
    hdr_ip* ih = hdr_ip::access(p);
    int bytes_to_deliver = hdr_cmn::access(p)->size();

    // if it is a MM packet (data or ack)
    if(ih->prio_ == 15) {
        if(app_) {  // if MM Application exists
            // re-assemble MM Application packet if segmented
            hdr_mm* mh = hdr_mm::access(p);
            if(mh->seq == asm_info.seq)
                asm_info.rbytes += hdr_cmn::access(p)->size();
            else {
                asm_info.seq = mh->seq;
                asm_info.tbytes = mh->nbytes;
                asm_info.rbytes = hdr_cmn::access(p)->size();
            }
            // if fully reassembled, pass the packet to application
            if(asm_info.tbytes == asm_info.rbytes) {
                hdr_mm mh_buf;
                memcpy(&mh_buf, mh, sizeof(hdr_mm));
                app_->recv_msg(mh_buf.nbytes, (char*) &mh_buf);
            }
        }
        Packet::free(p);
    }
    // if it is a normal data packet (not MM data or ack packet)
    else {
        if (app_) app_->recv(bytes_to_deliver);
        Packet::free(p);
    }
}
```

9.6.5. Test.tcl

```
# Author: Jae Chung
# Date:   7/17/99
#
#
#       s1                s3
#        \               /
# 5Mb,3ms \    2Mb,10ms   / 5Mb,3ms
#          r1 --------- r2
# 5Mb,3ms /             \ 5Mb,3ms
#       /               \
#      s2               s4
#

set ns [new Simulator]

#Define different colors for data flows
$ns color 1 Red
$ns color 2 Blue


#Open the nam trace file
set nf [open out.nam w]
set tf [open out.tr w]
$ns namtrace-all $nf
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
       global ns nf tf
       $ns flush-trace
       #Close the trace file
       close $nf
       close $tf
       #Execute nam on the trace file
       exec nam out.nam &
       exit 0
}

set node_(s1) [$ns node]
set node_(s2) [$ns node]
set node_(r1) [$ns node]
set node_(r2) [$ns node]
set node_(s3) [$ns node]
set node_(s4) [$ns node]

$ns duplex-link $node_(s1) $node_(r1) 5Mb 3ms DropTail
$ns duplex-link $node_(s2) $node_(r1) 5Mb 3ms DropTail
$ns duplex-link $node_(r1) $node_(r2) 2Mb 10ms RED
$ns duplex-link $node_(s3) $node_(r2) 5Mb 3ms DropTail
$ns duplex-link $node_(s4) $node_(r2) 5Mb 3ms DropTail

#Setup RED queue parameter
$ns queue-limit $node_(r1) $node_(r2) 20
Queue/RED set thresh_ 5
Queue/RED set maxthresh_ 10
Queue/RED set q_weight_ 0.002
Queue/RED set ave_ 0

$ns duplex-link-op $node_(r1) $node_(r2) queuePos 0.5

$ns duplex-link-op $node_(s1) $node_(r1) orient right-down
$ns duplex-link-op $node_(s2) $node_(r1) orient right-up
$ns duplex-link-op $node_(r1) $node_(r2) orient right
$ns duplex-link-op $node_(s3) $node_(r2) orient left-down
$ns duplex-link-op $node_(s4) $node_(r2) orient left-up


#Setup a MM UDP connection
set udp_s [new Agent/UDP/UDPmm]
set udp_r [new Agent/UDP/UDPmm]
$ns attach-agent $node_(s1) $udp_s
$ns attach-agent $node_(s3) $udp_r
```

```
$ns connect $udp_s $udp_r
$udp_s set packetSize_ 1000
$udp_r set packetSize_ 1000
$udp_s set fid_ 1
$udp_r set fid_ 1

#Setup a MM Application
set mmapp_s [new Application/MmApp]
set mmapp_r [new Application/MmApp]
$mmapp_s attach-agent $udp_s
$mmapp_r attach-agent $udp_r
$mmapp_s set pktsize_ 1000
$mmapp_s set random_ false

#Setup a TCP connection
set tcp [$ns create-connection TCP/Reno $node_(s2) TCPSink $node_(s4) 0]
$tcp set window_ 15
$tcp set fid_ 2

#Setup a FTP Application
set ftp [$tcp attach-source FTP]

#Simulation Scenario
$ns at 0.0 "$ftp start"
$ns at 1.0 "$mmapp_s start"
$ns at 7.0 "finish"

$ns run
```

10. NS 学习－定义一种新的协议(另外一例)

10.1. 例子的说明

One node will be able to send a packet to another node which will return it immediately, so that the round-trip-time can be calculated.

即是一个 Ping 的例子，一个结点送出一个数据包到达另一个结点，然后会立刻返回；同时可以计算往返的时间。

10.2. 头文件 ping.h

Ping 功能的数据包的数据结构

```
struct hdr_ping {
  char ret;  //去时，为 0；回来时，为 1
  double send_time;  //出发时的时刻
};
```

Ping 功能的类名

```
class PingAgent : public Agent {
 public:
  PingAgent();
  int command(int argc, const char*const* argv);
  void recv(Packet*, Handler*);
 protected:
  int off_ping_; //用来访问数据包的头，最后的下划线表示是局部变量
};
```

10.3. C++代码 ping.cc

描述了 C＋＋与 TCL 的关系

```
static class PingHeaderClass : public PacketHeaderClass {
public:
  PingHeaderClass() : PacketHeaderClass("PacketHeader/Ping",
                      sizeof(hdr_ping)) {}
} class_pinghdr;

static class PingClass : public TclClass {
public:
  PingClass() : TclClass("Agent/Ping") {}
  TclObject* create(int, const char*const*) {
    return (new PingAgent());
  }
} class_ping;
```

构造函数

```
PingAgent::PingAgent() : Agent(PT_PING) //应该在 pack.h 中手工 PT_PING 注册
{
  bind("packetSize_", &size_);
  bind("off_ping_", &off_ping_);
}//绑定了 C++和 Otcl 可以访问的数据
```

当一个 Ping Agent 的 TCL 命令执行时，将调用command 函数

```
int PingAgent::command(int argc, const char*const* argv)
{
  if (argc == 2) {
    if (strcmp(argv[1], "send") == 0) {
      // Create a new packet
      Packet* pkt = allocpkt();
      // Access the Ping header for the new packet:
      hdr_ping* hdr = (hdr_ping*)pkt->access(off_ping_);
      // Set the 'ret' field to 0, so the receiving node knows
      // that it has to generate an echo packet
      hdr->ret = 0;
      // Store the current time in the 'send_time' field
      hdr->send_time = Scheduler::instance().clock();
      // Send the packet
      send(pkt, 0);
      // return TCL_OK, so the calling function knows that the
      // command has been processed
      return (TCL_OK);
    }
  }
  // If the command hasn't been processed by PingAgent()::command,
  // call the command() function for the base class
  return (Agent::command(argc, argv));
}
```

Agent 接收到一个PING 的数据包后的处理，将ret 置 1，并返回

```
void PingAgent::recv(Packet* pkt, Handler*)
{
  // Access the IP header for the received packet:
  hdr_ip* hdrip = (hdr_ip*)pkt->access(off_ip_);
  // Access the Ping header for the received packet:
  hdr_ping* hdr = (hdr_ping*)pkt->access(off_ping_);
  // Is the 'ret' field = 0 (i.e. the receiving node is being pinged)?
  if (hdr->ret == 0) {
    // Send an 'echo'. First save the old packet's send_time
    double stime = hdr->send_time;
    // Discard the packet
    Packet::free(pkt);
    // Create a new packet
    Packet* pktret = allocpkt();
    // Access the Ping header for the new packet:
    hdr_ping* hdrret = (hdr_ping*)pktret->access(off_ping_);
    // Set the 'ret' field to 1, so the receiver won't send another echo
    hdrret->ret = 1;
    // Set the send_time field to the correct value
    hdrret->send_time = stime;
    // Send the packet
    send(pktret, 0);
  } else {
    // A packet was received. Use tcl.eval to call the Tcl
    // interpreter with the ping results.
    // Note: In the Tcl code, a procedure 'Agent/Ping recv {from rtt}'
    // has to be defined which allows the user to react to the ping
    // result.
    char out[100];
    // Prepare the output to the Tcl interpreter. Calculate the round
    // trip time
    sprintf(out, "%s recv %d %3.1f", name(),
            hdrip->src_ >> Address::instance().NodeShift_[1],
         (Scheduler::instance().clock()-hdr->send_time) * 1000);
    Tcl& tcl = Tcl::instance();
    tcl.eval(out);
    // Discard the packet
Packet::free(pkt);
  }
}
```

The most interesting part should be the 'tcl.eval()' function where a Tcl function 'recv' is called, with the id of the pinged node and the round-trip-time (in miliseconds) as parameters.

10.4. 必要的修改

要让 NS 知道新增加了一种协议

```
修改 packet.h 文件
enum packet_t {
    PT_TCP,
    PT_UDP,
       ......
    // insert new packet types here
    PT_TFRC,
    PT_TFRC_ACK,
       PT_PING,    //  packet protocol ID for our ping-agent
    PT_NTYPE // This MUST be the LAST one
};
修改 p_info()
class p_info {
public:
    p_info() {
        name_[PT_TCP]= "tcp";
        name_[PT_UDP]= "udp";
              ...........
        name_[PT_TFRC]= "tcpFriend";
        name_[PT_TFRC_ACK]= "tcpFriendCtl";

              name_[PT_PING]="Ping";

        name_[PT_NTYPE]= "undefined";
    }
       .....
 }
```

注意

在做 make 之前，一定要做 make depend，保证编译的正确性。

修改'tcl/lib/ns-default.tcl'

这个文件是用来定义 TCL 中所有对象的缺省值。增加下面一行：

Agent/Ping set packetSize_ 64

修改'tcl/lib/ns-packet.tcl'

增加一条新的数据包条目

{ SRMEXT off_srm_ext_}

{ Ping off_ping_ }} {

set cl PacketHeader/[lindex $pair 0]

增加 ping.o 到 NS 的 makefile 文件列表中

sessionhelper.o delaymodel.o srm-ssm.o \

srm-topo.o \

```
ping.o \

$(LIB_DIR)int.Vec.o $(LIB_DIR)int.RVec.o \

$(LIB_DIR)dmalloc_support.o  \
```

10.5. 现在可以用 make 命令了

10.6. 例子 ping.tcl

```
#Create a simulator object
set ns [new Simulator]

#Open a trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam &
        exit 0
}

#Create three nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

#Connect the nodes with two links
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail

#Define a 'recv' function for the class 'Agent/Ping'
Agent/Ping instproc recv {from rtt} {
    $self instvar node_
    puts "node [$node_ id] received ping answer from \
            $from with round-trip-time $rtt ms."
}

#Create two ping agents and attach them to the nodes n0 and n2
set p0 [new Agent/Ping]
$ns attach-agent $n0 $p0

set p1 [new Agent/Ping]
$ns attach-agent $n2 $p1

#Connect the two agents
$ns connect $p0 $p1

#Schedule events
$ns at 0.2 "$p0 send"
$ns at 0.4 "$p1 send"
$ns at 0.6 "$p0 send"
$ns at 0.6 "$p1 send"
$ns at 1.0 "finish"

#Run the simulation
$ns run
```

But I will show you how to write the 'recv' procedure that is called from the 'recv()' function in the C++ code when a ping 'echo' packet is received.
This code should be fairly easy to understand. The only new thing is that it accesses the member variable 'node_' of the base class 'Agent' to get the node id for the node the agent is attached to.

一定要牢记头文件和 C＋＋文件放在哪个目录

[参考文献]
[1]一个较好的教程，网址是：http://ns2.netlab.cse.yzu.edu.tw/Example/