## NS2节点

节点  
　　1.创建节点  
　　set ns [new Simulator]  
　　$ns node   
　　  
　　单播节点（unicast node）：  
　　   
　　Tcl对象之一：地址分类器（address classifier），用来判断分组的目标地址，C:\cygwin\home\tchen\ns-allinone-2.27\ns-2.27\classifier\classifier-addr.{h/cc}；  
　 　Tcl 对象之二：端口分类器（port classifier），用来判断分组的目标Agent，C:\cygwin\home\tchen\ns-allinone-2.27\ns-2.27 \classifier\classifier-port.{ h/cc}。  
　　2．配置节点  
　　使用函数simulator::node-config{}配置即将创建的节点属性，这种节点配置方法主要针对移动节点和卫星。  
　　  
　　e.g. example2.tcl  
　　——————————————————————————————————  
　　#设定模拟需要的一些属性，比如mobilenode的Channel、MAC、LL层的类型，天线类型，节点数目，场  
　　#景的长宽尺寸等。  
　　set val(chan) Channel/WirelessChannel ;#信道类型  
　　set val(prop) Propagation/TwoRayGround ;#无线-传播模型  
　　set val(netif) Phy/WirelessPhy ;#网络接口类型  
　　set val(mac) Mac/802\_11 ;#MAC类型  
　　set val(ifq) Queue/DropTail/PriQueue ;#接口队列类型  
　　set val(ll) LL ;#链路层类型  
　　set val(ant) Antenna/OmniAntenna ;#天线模型  
　　set val(ifqlen) 50 ;#ifq中的最大分组   
　　set val(nn) 2 ;#移动节点数  
　　set val(rp) AODV ;#路由协议  
　　set val(x) 500 ;#拓扑结构的X轴范围  
　　set val(y) 500 ;#拓扑结构的Y轴范围  
　　############################################################################################  
　　$set ns [new Simulator]  
　　############################################################################################  
　 　#建立节点（mobilenode）之前，先配置节点的一些参数。agentTrace表示应用层的trace，在trace文件中用#AGT表 示；routerTrace表示路由的trace，在toace文件中用RTR表示；macTrace表示MAC层的trace，  
　　#在trace文件中用MAC表示；movementTrace表示记录节点移动命令的trace，在trace文件中用M表示。  
　　$ns node-config -addressType def\ ;#设定节点地址类型：def(flat) & hierarchical  
　　-adhocRouting $val(rp) \ ;#设定移动节点所使用的路由协议  
　　-llType $val(ll) \ ;#设定移动节点的逻辑链路层  
　　-macType $val(mac) \ ;#设定移动节点的MAC层  
　　-ifqType $val(ifq) \ ;#设定移动节点的队列类型  
　　-ifqLen $val(ifqlen) \ ;#设定移动节点的队列长度  
　　-antType $val(ant) \ ;#设定移动节点的天线类型  
　　-propType $val(prop) \ ;#设定移动节点的无线信号传输模型  
　　-phyType $val(netif) \ ;#设定移动节点物理层类型  
　　-channelType $val(chan) \ ;#设定移动节点的无线信道类型  
　　-topoInstance $topo \ ;#设定移动节点的拓扑对象  
　　-agentTrace ON \ ;#是否打开应用层的trace  
　　-routerTrace ON \ ;#是否打开路由的trace  
　　-macTrace OFF \ ;#是否打开MAC层的trace  
　 　-movementTrace OFF ;#是否打开节点位置和移动信息的trace ############################################################################################ #建立两个节点（mobilenode），关闭节点的随机运动功能，即节点的运动完全由我们指定。   
　　for {set i 0} {$i < $val(nn) } {incr i} {  
　　 set node\_($i) [$ns node]   
　　 $node\_($i) random-motion 0 ;#取消随机移动  
　　}   
　　——————————————————————————————————  
　　  
　　-addressType：设定节点地址类型：def(flat) & hierarchical。  
　　-adhocRouting：设定移动节点所使用的路由协议。  
　　-llType：设定移动节点的逻辑链路层。  
　　-macType：设定移动节点的MAC层。  
　　-ifqType：设定移动节点的队列类型。  
　　-ifqLen：设定移动节点的队列长度。  
　　-antType：设定移动节点的天线类型。  
　　-propType：设定移动节点的无线信号传输模型。  
　　-phyType：设定移动节点物理层类型。  
　　-channelType：设定移动节点的无线信道类型。  
　　-topoInstance：设定移动节点的拓扑对象。  
　　-wiredRouting：是否支持有限网络的路由。  
　　-agentTrace：是否打开应用层的trace。  
　　-routerTrace：是否打开路由的trace。  
　　-macTrace：是否打开MAC层的trace。  
　　-movementTrace：是否打开节点位置和移动信息的trace。  
　　-reset： 重置所有属性为缺省值。  
　　3．分类器（classifier）  
　　e.g. C:\cygwin\home\tchen\ns-allinone-2.27\ns-2.27\classifier\classifier.h节选  
　　——————————————————————————————————  
　　class Classifier : public NsObject {  
　　public:  
　　 Classifier();  
　　 virtual ~Classifier();  
　　 inline int mshift(int val) { return ((val >> shift\_) & mask\_); }  
　　 virtual void recv(Packet\* p, Handler\* h);   
　　//recv()函数的实现参考源文件C:\cygwin\home\tchen\ns-allinone-2.27\ns-2.27\classifier\classifier.cc  
　　 virtual NsObject\* find(Packet\*);  
　　// find()函数的实现参考源文件C:\cygwin\home\tchen\ns-allinone-2.27\ns-2.27\classifier\classifier.cc  
　　 virtual int classify(Packet \*);  
　　 virtual void clear(int slot); //从表中删除对象  
　　 virtual void install(int slot, NsObject\*); //从表中增加对象  
　　protected:  
　　 virtual int command(int argc, const char\*const\* argv);  
　　 void alloc(int); // alloc()函数动态的分配足够的空间来放置slot  
　　 NsObject\*\* slot\_; /\* table that maps slot number to a NsObject \*/  
　　 int nslot\_;  
　　 int maxslot\_;  
　　 int offset\_; // offset for Packet::access()  
　　 int shift\_;  
　　 int mask\_;  
　　 NsObject \*default\_target\_;  
　　 int nsize\_; //what size of nslot\_ should be  
　　};  
　　——————————————————————————————————  
　　  
　　e.g. C:\cygwin\home\tchen\ns-allinone-2.27\ns-2.27\classifier\classifier.cc节选  
　　——————————————————————————————————  
　　void Classifier::recv(Packet\* p, Handler\*h)  
　　{  
　　 NsObject\* node = find(p);  
　　 if (node == NULL) {  
　　 Packet::free(p);  
　　 return;  
　　 }  
　　 node->recv(p,h);  
　　}  
　　  
　　NsObject\* Classifier::find(Packet\* p)  
　　{  
　　 NsObject\* node = NULL;  
　　 int cl = classify(p);  
　　 if (cl < 0 || cl >= nslot\_ || (node = slot\_[cl]) == 0) {   
　　 if (default\_target\_)   
　　 return default\_target\_;  
　　 Tcl::instance().evalf("%s no-slot %ld", name(), cl);  
　　 if (cl == TWICE) {  
　　 cl = classify(p);  
　　 if (cl < 0 || cl >= nslot\_ || (node = slot\_[cl]) == 0)  
　　 return (NULL);  
　　 }  
　　 }  
　　 return (node);  
　　}  
　 　当classifier收到一个分组时（即recv()函数被调用时），recv()函数首先调用find()函数来处理该分组，find()函数会调 用classify()函数，不同类型的classifier对classify()函数定义不同，但各种classify函数都会对分组作检查然后返回 一个slot索引值。如果索引值有效，并指向一个有效的TclObject，recv()函数就会把分组传递给该TclObject，即调用该 TclObject的recv()函数；如果索引无效，就会清除该分组并返回。  
　　——————————————————————————————————  
　　  
　　几种主要的Classifier类：  
　 　1．address classifier：按分组目的地址进行匹配，用来支持单播分组转发，通过对分组的目的地址做位运算来产生一个slot number。该类对应的classify()函数定义参考C:\cygwin\home\tchen\ns-allinone-2.27\ns- 2.27\classifier\classifier-addr.cc。  
　　——————————————————————————————————  
　　int AddressClassifier::classify(Packet \*p)   
　　{  
　　 hdr\_ip\* iph = hdr\_ip::access(p);  
　　 return mshift(iph->daddr());  
　　};  
　 　iph->daddr()函数返回分组IP头中的目的地址，mshift(int val)函数的定义在前面Classifier类的定义中可以找到，其中mask\_和shift\_的值可以通过Otcl来设置，缺省值分别为 0xffffffff和0，即mshift(int val)直接返回val的值。  
　　——————————————————————————————————  
　　  
　 　2．port classifier：按分组目的端口进行匹配，将分组传递给相应的Agent对象。该类对应的classify()函数定义参考C:\cygwin \home\tchen\ns-allinone-2.27\ns-2.27 \classifier\classifier-addr.cc。  
　　——————————————————————————————————  
　　int PortClassifier::classify(Packet \*p)   
　　{  
　　 hdr\_ip\* iph = hdr\_ip::access(p);  
　　 return iph->dport();  
　　};  
　　iph->dport()函数返回分组IP头中的目的端口。  
　　——————————————————————————————————  
　　  
　 　3．replicator：与普通classifier不同，不使用classify()函数，作用是生成一个分组的多份拷贝，并把这些拷贝转发给 slot表中的所有对象。在组播分组转发时，一个分组需要被转发给多个目标对象，生成分组拷贝的工作由replicator完成。Replicator类 的定义及其recv()函数的定义参考C:\cygwin\home\tchen\ns-allinone -2.27\ns-2.27\mcast\replicator.cc。  
　　——————————————————————————————————  
　　class Replicator : public Classifier  
　　{  
　　public:  
　　 Replicator();  
　　 void recv(Packet\*, Handler\* h = 0);  
　　 virtual int classify(Packet\*) {/\*NOTREACHED\*/ return -1;};  
　　protected:  
　　 virtual int command(int argc, const char\*const\* argv);  
　　 int ignore\_;  
　　 int direction\_;  
　　};  
　　  
　　void Replicator::recv(Packet\* p, Handler\*)  
　　{  
　　 hdr\_ip\* iph = hdr\_ip::access(p);  
　　 hdr\_cmn\* ch = hdr\_cmn::access(p);  
　　 if (maxslot\_ < 0) {  
　　 if (!ignore\_)   
　　 Tcl::instance().evalf("%s drop %ld %ld %d", name(),   
　　 iph->saddr(), iph->daddr(), ch->iface());  
　　 Packet::free(p);  
　　 return;  
　　 }  
　　……  
　　 for (int i = 0; i < maxslot\_; ++i) {  
　　 NsObject\* o = slot\_[i];  
　　 if (o != 0)  
　　 o->recv(p->copy());  
　　 }  
　　 /\* we know that maxslot is non-null \*/  
　　 slot\_[maxslot\_]->recv(p);  
　　}   
　　可以看到Replicator类的recv()函数并不调用classify()函数，它只是为slot表中的每一个对象复制一份该分组的拷贝，并把拷贝发送给该对象，表中的最后一个对象收到的是原始的分组，而原始分组和拷贝实际上是完全一样的。  
　　——————————————————————————————————  
　　  
　　Classifier类提供了一些Tcl实例过程，用户可以在Tcl中对Classifier对象进行控制。主要实例过程：  
　　1． alloc-port ：寻找一个空闲的slot。  
　　2． clear ：将号码为index的slot清空。  
　　3． installNext：在最后一个slot后插入一个新的指向object的表项，并返回该表项的slot号码：如果object为空，会返回错误信息。   
　　4． slot ：产旬号码为index的slot对应对象名，如果该slot为空，会返回错误信息。  
　　5． findslot：查询一个object所在的slot号码，如果该object为空，会返回错误信息，如果在表中没有找到该object，会返回-1。  
　　6． install ：将号码为index的slot所指向的对象设为object。  
　　  
　　e.g. Classifier类部分Tcl实例过程的使用方法举例，参见C:\cygwin\home\tchen\ns -allinone-2.27\mimu\example17.tcl。  
　　——————————————————————————————————  
　　set ns [new Simulator] ;#建立一个Simulator对象的实例并把它赋值给变量ns。  
　　set node [$ns node] ;#新建一个节点并赋值给node。  
　　set udp0 [new Agent/UDP] ;#新建一个UDP Agent并赋值给udp0。  
　　set udp1 [new Agent/UDP] ;#新建一个UDP Agent并赋值给udp1。  
　　set null [new Agent/Null] ;#新建一个UDP Agent并赋值给null。  
　　$ns attach-agent $node $udp0; #将UDP Agent udp0绑定到node上。  
　　  
　　puts "[[$node set dmux\_] slot 0]" ;#查询号码为0的slot对应的对象名并将其输出。  
　　puts "$udp0" ;#输出udp0的值。  
　　puts "[[$node set dmux\_] findslot $udp0]" ;#查询udp0所在的slot号码并输出。  
　　puts "[[$node set dmux\_] findslot $null]" ;#查询null所在的slot号码，表中没有找到该对象，返回-1，并输出。  
　　  
　　puts "============================================="  
　　puts "[[$node set dmux\_] installNext $udp1]";#在最后一个slot 0后插入一个新的指向对象udp1的表项，返回  
　　;#该表项的slot号码1，并输出。  
　　puts "[[$node set dmux\_] slot 1]" ;#查询号码为1的slot对应的对象名并将其输出。  
　　puts "$udp1" ;#输出udp1的值。  
　　  
　　puts "============================================="  
　　[$node set dmux\_] install 0 $udp1 ;#将号码为0的slot所指向的对象设为udp1。  
　　puts "[[$node set dmux\_] slot 0]" ;#查询号码为0的slot对应的对象名并将其输出。  
　　  
　　puts "============================================="  
　　puts "[[$node set dmux\_] alloc-port $null]" ;#寻找一个空闲的slot并输出。  
　　[$node set dmux\_] clear 0 ;# 将号码为0的slot清空。  
　　puts "[[$node set dmux\_] alloc-port $null]" ;#寻找一个空闲的slot并输出。  
　　  
　　puts "============================================="  
　　puts "[[$node set dmux\_] slot 0]" ;# 查询号码为0的slot对应的对象名,该object为空，故返回错误信息。  
　　  
　　$ns run  
　　  
　　运行结果：  
　　   
　　——————————————————————————————————  
　　  
　　相关Tcl命令：  
　　1． set $node [$ns node]：建立一个节点实例。  
　　2． $node id：返回该节点id。  
　　3． $node neighbors：返回邻居节点的列表。  
　　4． $node add-neighbor ：增加一个邻居节点（注：这是单向的邻居，即“neighbor node”是node的邻居，但node不是“neighbor node”的邻居）。  
　　5． $node node-addr：返回节点的地址（address）。地址类型为def时，节点地址与节点id相同；地址类型为hierarchical时，节点地址是一个字符串。  
　　6． $node reset：重置连到这个node上的所有agent。  
　　7． $node agent ：返回port\_num端口所指向的agent对象，如果port\_num端口没有指向任何对象，返回null字符串。  
　　8． $node attach ：将agent对象连接到节点上，如果没有指定端口号，节点会自己分配一个空闲的端口，并把agent连接到该端口上；如果指定端口为port\_num，节点会把agent连接到端口port\_num上。   
　　9． $node detach ：将agent与节点分离，并把一个null\_agent连接到agent原来所在的端口上。

## [NS2完整的流程简介](http://www.loosky.net/?p=527)

1、建立脚本文件（只给出两个关键点的代码）  
配置模拟属性：  
set val(chan)  Channel/WirelessChannel   ;# 信道类型  
set val(prop) Propagation/TwoRayGround   ;# 传播模型  
set val(netif)   Phy/WirelessPhy         ;# 物理层  
set val(mac)     Mac/802\_11              ;# MAC类型  
set val(ifq)     Queue/DropTail/PriQueue ;#  
set val(ll)      LL                      ;# 链路层  
set val(ant)     Antenna/OmniAntenna     ;# 天线类型  
set val(ifqlen)   50                     ;# ifq中最大包数  
set val(nn)       30                     ;# 移动节点个数  
set val(rp)       AODV                   ;# 路由协议  
set opt(cp)      “CBR”                   ;# 数据流类型  
set opt(sc)      “scen”                  ;# 场景文件

配置移动节点：  
# 构造节点  
$ns\_ node-config -adhocRouting $val(rp) \  
-llType $val(ll) \  
-macType $val(mac) \  
-ifqType $val(ifq) \  
-ifqLen $val(ifqlen) \  
-antType $val(ant) \  
-propType $val(prop) \  
-phyType $val(netif) \  
-channelType $val(chan) \  
-topoInstance $topo \  
-agentTrace ON \  
-routerTrace OFF \  
-macTrace ON \  
-movementTrace OFF

2、生成场景文件  
./setdest -v 1 -n 30 -p 0.0 -M 40 -t 500 -x 1000 -y 1000 >scen  
注意：该场景文件在目录：ns-2.30\indep-utils\cmu-scen-gen\setdest下生成。

3、生成数据流文件  
ns cbrgen.tcl -type cbr -nn 30 -seed 1 -mc 30 -rate 1.0 >cbr  
注意：该数据流文件在目录：ns-2.30\indep-utils\cmu-scen-gen下生成。

4、把生成的场景文件数据流文件copy到和步骤1的脚本文件同一个目录下。

5、运行NS2软件进入脚本文件所在目录，运行命令：ns \*.tcl。得到一个 \*.tr文件和一个 \*.nam文件。

6、编写awk脚本，对trace文件中数据进行统计。

7、运行命令：gawk –v src=1 –v dst=2 –v outfile=1-2 data –f \*.awk \*.tcl,生成文件 \*.data。

8、编写gnuplot的plot脚本文件 \*.plot。

9、运行命令：gnuplot \*.plot，生成所要图形。

### 相关文章

* 2009-03-03 -- [基于NS2的无线多跳传输仿真](http://www.loosky.net/?p=655)
* 2009-03-01 -- [NS2一些有用的网站](http://www.loosky.net/?p=653)
* 2009-03-01 -- [Hyacinth for ns2.29修改步骤](http://www.loosky.net/?p=651)
* 2009-03-01 -- [Multi-channel and Multi-interface in NS2](http://www.loosky.net/?p=649)
* 2008-12-19 -- [NS2简介与其功能模块](http://www.loosky.net/?p=598)
* 2008-10-23 -- [NS2简介](http://www.loosky.net/?p=528)
* 2008-09-17 -- [NS2中802.11代码深入理解—packet传输的流程(转载)](http://www.loosky.net/?p=473)
* 2008-09-17 -- [NS2中C++与TCL的三种接口](http://www.loosky.net/?p=472)

1. NSME是什么的缩写？

        Network Security Modeling Environment，为什么取这个名字呢？因为Siming最早接触模拟的动机来自网络安全研究，比如网络蠕虫扩散、DDoS等。你可以称呼它为“NS me”，NSme is ME，呵呵。

2. NSME是一个通用的网络模拟器

    尽管最初是为了网络安全研究目的，但NSME也是一个通用的网络模拟器，只不过它在支持下层模拟细节的同时，更侧重于网络上层应用逻辑的表达。  
    其实从某种意义上说，网络安全研究也是一种特殊的应用层研究。当然，严格说来，网络安全的问题来自网络各个层次，并不仅局限在应用层。但是此类问 题（或者说现象）往往都由较为复杂的上层应用逻辑驱动，这和单纯的网络下层协议性能分析是不同的，所以将其归为一种特殊的应用也不为过。可见，NSME的 设计宗旨就是为了满足细致的应用层逻辑模拟，同时最大限度的保持下层模拟的真实性。

3. NSME与NS2的关系

    在 设计NSME之前，Siming研究了1年多的NS2。事实上NSME的早期版是在NS2基础上扩展得到的，那时Siming的研究重点侧重于挖掘NS2 的仿真（emulation）功能。具体说来，是要研究轻量级蜜罐（Honey pot），使多个仿真的虚假节点能够与外界真实计算机自如通信。熟悉NS2的朋友可能知道，它提供简单的仿真能力（emulate模块）。但是这种仿真功 能与各种网络仿真器（Network Emulator）是类似的，即只起到包转发的功能，并不对转发的包做应用层处理。简言之，就是通信会话的源、目的端都不在仿真器内部。这是无法做 Honeypot的，……

    毕竟仿真器支持的网络规模是非常有限的，所以在上述研究成果基础 上，Siming转到了模拟（Simulation）的工作中来，自然而然的，也将详细的上层应用刻画引入到模拟中去。不过此时NS2框架已经不再适用 了。在Siming看来，NS2的Otcl和Agent是两大固有缺陷，要想随心所欲的、简单的实现模拟目标，这种结构必须抛弃。:( 当然，NS2代码中的若干前人贡献（如随机数生成算法）在NSME中仍然在发扬光大。:)

    除此以 外，改动是相当巨大的、和结构性的。比如TCP协议，在NS2中实现了很多不太完整的TCP实现，相对最完整、最常用的是fulltcp了（当 然，fulltcp也不完整），在NSME的早期版本中也是直接在fulltcp agent上做的修改和补充，使它能够处理真实的TCP数据包（带真实负载） 。但是在目前的版本中，Siming根据状态转移图重写了TCP实现（fulltcp agent里面的代码实在是比较乱啊，隔几天不看就容易晕） 。当然也有些简化（如不支持TCP协议头中的可选字段），不过这对于上层应用模拟而言已经足够详细了。 另外在路由模拟方面，NSME从一开始就加入了IP地址和掩码的完全支持，每个路由器的路由表不是简单的数组，而是RadixTree结构，也就是说在路 由查找、寻址等方面完全和真实情况类似。

    其他诸如可视化和并行化就不多说了，对模拟用户而言，这些并不是本质的模拟功能，尽管它们对模拟任务的完成起到本质的支撑作用。

4. 为什么转到Windows平台？

    NSME 的总体结构是由可视化的客户端x-nsme完全配置、控制和回显模拟过程，由用户基于libnsme创建的模拟应用程序是模拟执行的实际承担者。把x- nsme做到windows平台是可以理解的，至于为什么把模拟应用程序也做到windows平台的主要原因是——Siming偷了个懒，:)，在VC下 一起开发x-name和libnsme比较方便嘛，呵呵。不过将来肯定要出Linux版的，这个不难，因为libnsme几乎是用标准C++写的。

## [Hyacinth for ns2.29修改步骤](http://www.loosky.net/?p=651)

1. 下载ns-allinone-2.29.tar.gz：http://www.isi.edu/nsnam/dist/，注意是ns-allinone- 2.29.tar.gz，在ns-allinone-2.29.1.tar.gz和ns-allinone-2.29.2.tar.gz上面进行修改会出 错。  
2．查看http://www.cse.msu.edu/~wangbo1/ns2/nshowto8.html，按照文章中内容对相应文件进行修改。  
3．安装ns-allinone-2.29.tar.gz，此压缩包中的文件有错，安装之前修改\ns-allinone-2.29\nam-1.11\agent.h文件第73行NULL为0，否则会出现“找不到NAM命令”的错误信息。  
4. 安装成功后，编译测试文件。  
  
Hyacinth的手动routing table的设定：  
0—>1—>2  如果是这样的拓扑,0如果要routing给2就要在0上加去2的route tag,以及1去2的tag,相同的tag.其实也就是和实际的routing table上的标识一样. 然后做好tag后,增加nexthop. 0去2要走1通过nic2,1去2要走2通过nic2. nic可改。  
正确的设置为：  
# Add routing tag  
$node\_(0) add-route-tag 2 02  
$node\_(1) add-route-tag 2 02  
# Add next hop  
$node\_(0) add-next-hop 2 02 1 2  
$node\_(1) add-next-hop 2 02 2 1

# Add routing tag 这个function要有4个input 分別为 New\_entry, add-route-tag, entry.dsy, entry.tag  
如：$node\_(0) add-route-tag 1 999 ，增加一条route table是从node\_(0) 传到 node\_(1) 并且tag 设为 999

#Add next hop 这个function要有6个input 分別为 fixed\_rtable\_ent, add-next-hop, dst, tag, next hop, nic，其中 nic 代表是使用的频道数，已经与对于的接口绑定了。  
如：$node\_(0) add-next-hop 1 999 1 2，就是node\_(0)通过一跳传输, 传给node\_(1), tag为999, 使用第二频道。

Hyacinth修改方法，使用了一种专有的路由，并且接口和频道都是固定的，支持NS2 2.29可参考以下文章进一步修改：  
（1）Adding Multiple Interface Support in NS-2：http://personales.unican.es/aguerocr/files/ucMultiIfacesSupport.pdf  
（2）NOAH & Hyacinth : Multi-Channel & Multi-Interface Extension in NS2：http://dobbymmlab.blogspot.com/2008/07/multi-channel-multi- interface-extension.html  
（3）Make “hyacinth” run on Debian NS-2.29.2 ：http://my.opera.com/HenryFD/blog/