# NS2 Training Course _NS Modules

Reporter：Chau-Chi Wang

Date：2008/7/9



Wireless Network Lab., NC, CYUT

# Outline

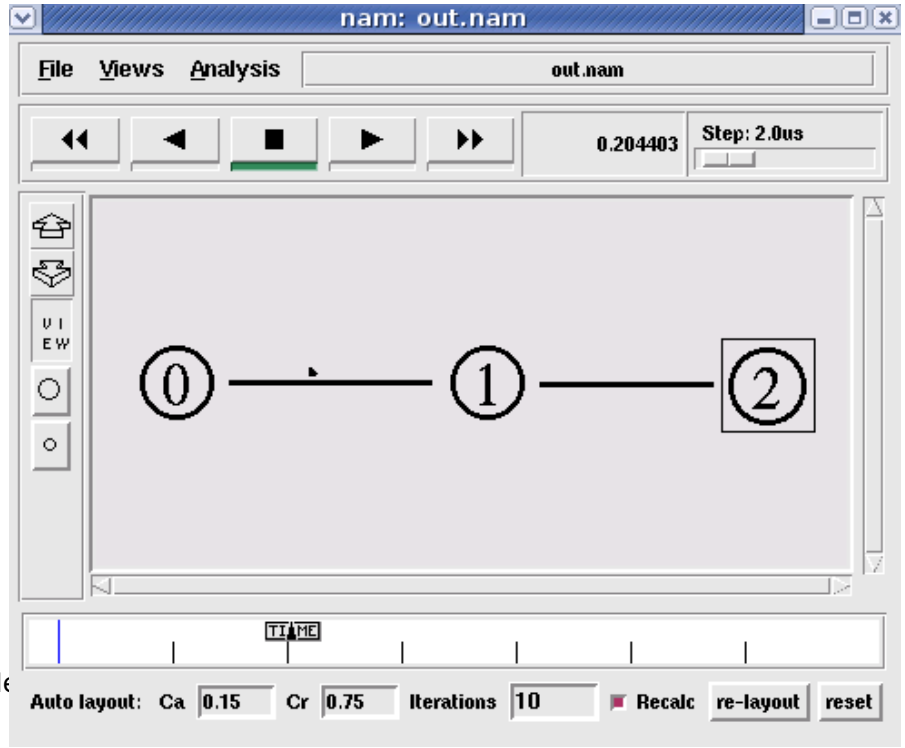
- Introduction
- PING
- Add NS2 module
- Modify NS2 module
- Reference

# Introduction

- Modules ?
- Protocols ?

# NS-2 Modules

PING



# PING _ ping.tcl (1/2)

```
#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."
}
```

# PING _ ping.tcl (2/2)

```
#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
```

Result :

```
.....................................
node 0 receved ping answer form 2 with round-trip-time 42.0 ms.
node 2 receved ping answer form 0 with round-trip-time 42.0 ms.
node 0 receved ping answer form 2 with round-trip-time 42.0 ms.
node 2 receved ping answer form 0 with round-trip-time 42.0 ms.
```

# PING _ ping.cc (1/4)

```
#include "ping.h"

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;
```

```
PingAgent::PingAgent() : Agent(PT_PING)
{
  bind("packetSize_", &size_);
  bind("off_ping_", &off_ping_);
}
```

# PING _ ping.cc (2/4) _ command

```
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));
}
```

# PING _ ping.cc (3/4)
# _ recv

```
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.
```

# PING _ ping.cc (4/4)
# _ recv (cont.)

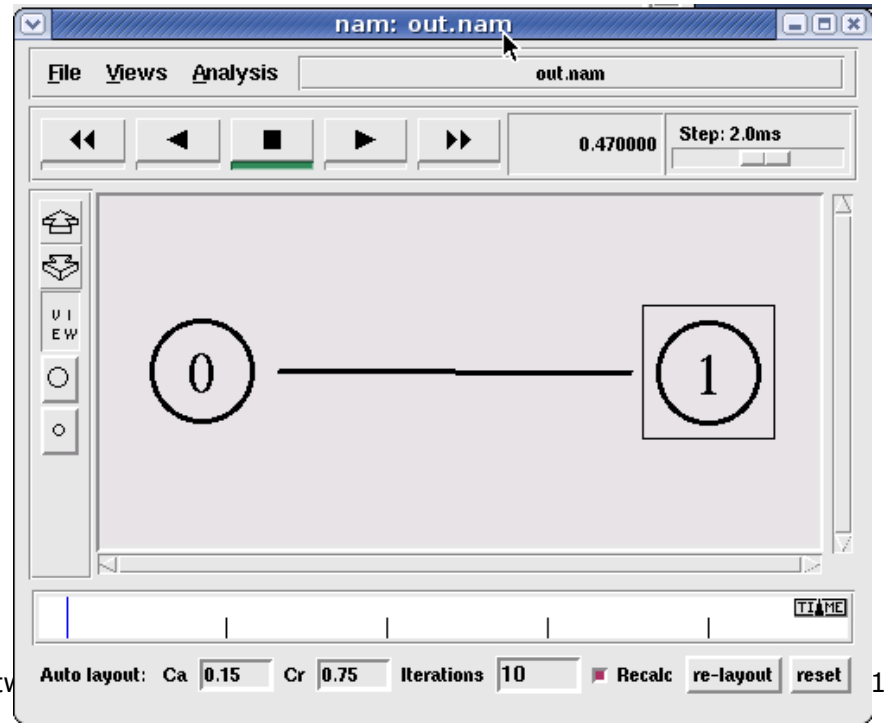
```
char out[100];
    // Prepare the output to the Tcl interpreter.
Calculate the round trip time
    sprintf(out, "%s recv %d %3.1f", name(),
            hdrip->src_ >> Address::instance().
NodeShift_[1],
                    (Scheduler::instance().clock()-hdr-
>send_time) * 1000);
    Tcl& tcl = Tcl::instance();
    tcl.eval(out);
    // Discard the packet
    Packet::free(pkt);
  }
}
```

# NS-2 Modules

## NS-2 user define protocol

# NS-2 Modules

## ECHO



# ECHO_ echo.tcl

```
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 1Mb 10ms DropTail

set p0 [new Agent/ECHO]
$ns attach-agent $n0 $p0
set p1 [new Agent/ECHO]
$ns attach-agent $n1 $p1
$ns connect $p0 $p1

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

```
$ns at 0.2 "$p0 send"
$ns at 0.4 "$p1 send"
$ns run
```

Result:

```
[gregg@sctp apps]$ ns echo.tcl
node 0 received ping answer from  1 with
round-trip-time 20.0 ms.
node 1 received ping answer from  0 with
round-trip-time 20.0 ms.
```

# ECHO _ echo.h

- 建立 **EchoAgent**所需之封包格式

```
struct hdr_echo {
        // send_time為紀錄發送端傳送要求回應的時間
        double send_time;
        // state為紀錄封包狀態的資訊，若state = 0時，則表示該封包為要求回應之封包，state = 1時，則表示該封包為目的端回應封包
        char state;
       //了提供NS2其他元件可以使用此自訂封包struct hdr_echo{}，因此必須給予一個可供PacketManager存取的地方。因此必須於自訂封包內加入offset_參數，並經由Packet::access()函式取得struct hdr_echo內的資料。
        static int offset_;
        inline static int& offset() { return offset_; }
        inline static hdr_echo* access(const Packet* p) {
                return (hdr_echo*) p->access(offset_);
        }
};
```

# ECHO _ echo.h

- 繼承 **Agent**建立 **EchoAgent**類別

```
// class Agent{}為建立協定的base class,
   因此欲建立新的協定時, 必須繼承該物件

class EchoAgent : public Agent {
public:
EchoAgent();
        int command(int argc, const char *const *argv);
        void recv(Packet *, Handler *);
};
```

# ECHO _ echo.cc (1/6)

- 建立 **C++**與**TCL**間的接口模組

```
// PacketHeaderClass{}為管理封包表頭之用的類別，因此我們必須給予該封包表頭一個名稱，並標明其繼承階層
int hdr_echo::offset_;
static class EchoHeaderClass : public PacketHeaderClass {
      public:
      EchoHeaderClass() : PacketHeaderClass("PacketHeader/ECHO",sizeof(hdr_echo)) {
            bind_offset(&hdr_echo::offset_);
       }
}class_echohdr;
```

# ECHO _ echo.cc (2/6)

- 建立 **C++**與**TCL**間的接口模組 **(cont.)**

```
// TclClass{}所扮演的角色為提供Tcl取得C++物件的窗口，因此也需給予自訂協定一名稱以及繼承階層

static class EchoClass : public TclClass {
        public:
        EchoClass() : TclClass("Agent/ECHO") {}
        TclObject * create( int, const char *const *) {
                return (new EchoAgent());
        }
}class_echo;
```

# ECHO _ echo.cc (3/6)

- 給予 **EchoAgent**封包名稱

```
// Agent之建構子，其目的為給予此Agent一個固定名稱，以提供NS2辨別封包型態。如上圖所示，我們給予此Agent的封包型態名稱為PT_ECHO。可經由struct hdr_cmn::ptype()取得該名稱以作為判斷之用。

EchoAgent::EchoAgent() : Agent(PT_ECHO) {

}
```

# ECHO _ echo.cc (4/6)

- **EchoAgent::command()**

```
//當我們對TCL物件下達send指令時，該send指令會送到物件的command此函式中進行處理
int EchoAgent::command(int argc, const char*const* argv) {
    //其參數有兩個的時候，第二個為指令參數。當接收到send指令時，會先配置記憶體位置給封包，並將各種所需資訊寫入自訂的封包格式中，然後再將封包透過Agent::send()傳送出去
    if (argc == 2) {
        if (strcmp(argv[1], "send") == 0) {
            Packet* pkt = allocpkt();
            struct hdr_echo* hdrcom = (hdr_echo*)hdr_echo::access(pkt);
            hdrcom->state = 0;
            //系統時間可由Scheduler::instance().clock()函式取得
            hdrcom->send_time = Scheduler::instance().clock();
            send(pkt, 0);
            return (TCL_OK);  //回傳一個TCL_OK訊息回去給上一層呼叫者
        }
    }
    return (Agent::command(argc, argv));
}
```

# ECHO _ echo.cc (5/6)

- **EchoAgent::recv()**

```
void EchoAgent::recv(Packet* pkt, Handler*){
                struct hdr_ip *hdrip = (hdr_ip*)hdr_ip::access(pkt);
                struct hdr_echo *hdrecho = (hdr_echo*)hdr_echo::access(pkt);
      // //表示接收到ping要求.
                if (hdrecho->state == 0) {
          double stime = hdrecho->send_time;
                    Packet::free(pkt);
                    Packet* pktret = allocpkt();
                    struct hdr_echo *hdrret = (hdr_echo*)hdr_echo::access(pkt);
                    hdrecho->state = 1;
                    hdrecho->send_time = stime;
                    send(pktret, 0);
                } else {
            ……..
```

# ECHO _ echo.cc (6/6)

- **EchoAgent::recv()**

```
        } else {
    //收到ping要求的回應
            char out[100];
    //下面的" %s showecho %d %3.1f"是對應到tcl程式中的 gotit函式
    sprintf(out, "%s showecho %d %3.1f", name(),
                    hdrip->src_.addr_ >> Address::instance().NodeShift_[1],
              (Scheduler::instance().clock()- hdrecho->send_time) * 1000);
    //形成一個實體
    Tcl& tcl = Tcl::instance();
            tcl.eval(out);
            Packet::free(pkt);
        }
}
```

# Insert Module

- packet.h
  - ../ns-allinone-2.31/ns-2.31 /common/packet.h
- ns-default.tcl
  - ../ns-allinone-2.31/ns-2.31/tcl/lib/ns-default.tcl
- Makefile
  - ../ns-allinone-2.31/ns-2.31/Makefile
- make clean; make

```
.....
        PT_PING,
        PT_ECHO,
.....
        name_[PT_PING]="ping";
        name_[PT_ECHO]="echo";
.....
```

```
Agent/Ping set packetSize_ 64
```

```
..... sink.o mobile/energy-model.o apps/ping.o
apps/echo.o tcp/tcp-rfc793
```

# NS-2 Modules

## Modify Drop-tail module into myfifo

# Insert Module

- 把路徑切換到queue的目錄下。

  cd  ns-allinone-2.31/ns-2.31/queue
- 拷貝drop-tail.[cc, h]到myfifo.[cc.h]。

  cp  drop-tail.cc  myfifo.cc

  cp  drop-tail.h   myfifo.h
- 使用文字編輯軟體去修改myfifo.h和myfifo.cc，先修改myfifo.h，所有DropTail改成myfifo，另外，把drop_tail改成myfifo

# Insert Module

- 再修改myfifo.cc, 使用取代的功能把所有DropTail改成myfifo, 另外, 把drop_tail改成myfifo和drop-tail改成myfifo。

- 修改ns-default.tcl檔, 設定初始內定值。
  - cd ns-allinone-2.31/ns-2.31/tcl/lib/
  - 使用文字編輯軟體打開ns-default.tcl
  - 使用搜尋的功能找到Queue/DropTail
  - 把每個初始設定值都再設一份給Queue/myfifo

# Insert Module

```
Queue/DropTail set drop_front_ false
Queue/DropTail set summarystats_ false
Queue/DropTail set queue_in_bytes_ false
Queue/DropTail set mean_pktsize_ 500
Queue/myfifo set drop_front_ false
Queue/myfifo set summarystats_ false
Queue/myfifo set queue_in_bytes_ false
Queue/myfifo set mean_pktsize_ 500
```

# Insert Module

- 修改Makefile, 把myfifo.o加入OBJ_CC內，並重新編譯。
  - 使用文字編輯軟體打開ns-allinone-2.31/ns-2.31目錄下的Makefile
  - 使用搜尋找到drop-tail.o。
  - 在drop-tail.o後面加上queue/myfifo。

```
.....
tools/flowmon.o tools/loss-monitor.o \
queue/queue.o queue/drop-tail.o  queue/myfifo.o \
adc/simple-intserv-sched.o queue/red.o \
.....
```

    - 重新編譯。
  - make

# Summary

- 如何新增模組到ns2的核心步驟如下:
  - 1.準備好模組檔(例如, a.cc 和 a.h)。
  - 2.若有需要做初始設定的話, 修改ns-default.tcl檔。
  - 3.修改Makefile(把a.o加到OBJ_CC內)
  - 4.重新編譯
  - 5.測試模組

# Reference

- http://140.116.72.80/~smallko/ns2/BuildEcho.htm
- http://140.116.72.80/~smallko/ns2/module.htm