فصل چهارم

پروتکل‌ها

۴-۱ مقدمه

پروتکل عبارت است از مجموعه قوانین لازم به منظور قانونمند نمودن نحوه ارتباط در سیستم‌های مختلف. در این میان شبکه‌های کامپیوتری نیز از این مطلب مستثنا نشده‌اند. در این فصل ابتدا برخی از پروتکل‌های شبیه‌سازی را معرفی و در ادامه به شبیه‌سازی چند نمونه از آن‌ها می‌پردازیم.

۴-۲ معرفی برخی از پروتکل‌ها

MAC Layer ۴-۲-۱

این لایه در واقع پروتکل MAC با الگوی DCF[1] است که توسط CMU پیاده‌سازی شده‌است. این پروتکل از الگوی RTS/CTS/DATA/ACK برای ارسال‌های تک پخشی و از الگوی DATA برای ارسال‌های broadcast استفاده می‌کند. در پیاده‌سازی این پروتکل از هر دوی روش گوش کردن به کانال فیزیکی و مجازی استفاده می‌کند. کلاس Mac802_11 در شاخه ns/mac802_11~ قرار دارد.

Tap Agent ۴-۲-۲

Agentهایی که زیر کلاسی از کلاس Tap باشند پروتکل MAC پیش از این که عمل فیلترینگ را روی آنها انجام دهد همه آنها را دریافت خواهد کرد. کلاس Tap در ns/mac~ قرار دارد.

Network Interfaces ۴-۲-۳

لایه میانی شبکه که یک رابط سخت‌افزاری به گره‌های متحرک برای دسترسی به کانال می‌دهد. رابط رسانه اشتراکی بی‌سیم با دو کلاس Phy/wirelessPhy پیاده‌سازی شده است. این رابط از مدل تصادم و

---

1. Distributed Coordination Function

انتشار برای دریافت و ارسال بسته‌ها استفاده می‌کند. برچسب‌های داخل هر سرآیند اطلاعات meta data هستند که در شبکه‌های بی‌سیم استفاده می‌شوند (مثل قدرت و بازه انتقال سیگنال یا طول موج و غیره). این داده‌ها در سرآیند توسط مدل انتشار در رابط شبکه گیرنده به منظور تعیین مد ارسال استفاده می‌شود. در شبکه‌های بی‌سیم مد ارسال به سه روش گوش کردن به کانال، ذخیره و ارسال و مد حفظ توان مصرفی است. این مدل رابط رادیویی DSSS[1] را تخمین می‌زند. این دو کلاس در فهرست ns/wireless-phy~ موجود هستند.

۴-۲-۴ لایه‌های MAC مجزا برای شبکه‌های متحرک

دو پروتکل برای لایه Mac و NS برای گره‌های متحرک پیاده‌سازی شده که یکی 802.11 و دومی TDMA است.

۴-۲-۴-۱ پروتکل MAC802.11

در فصل‌های گذشته راجع به این پروتکل کمی توضیح دادیم و برای دیدن اطلاعات بیشتر به ns/mac_802.11~ مراجعه کنید.

۴-۲-۴-۲ پروتکل TDMA[2] پایه‌ای

همان‌طورکه از مخفف این پروتکل پیداست از تقسیم بازه زمانی برای دسترسی چندگانه استفاده می‌کند. نکته‌ای که باید در این جا در نظر داشته باشید این است که این روش هنوز در حال پیاده‌سازی اولیه است و محیط multi hop در آن در نظر نگرفته شده است.

برعکس پروتکل 802.11 در TDMA از time slotهای مختلف برای شروع رقابت ارسال و دریافت بسته‌ها استفاده می‌کند. به هر کدام از این time slotها یک FDMA frame می‌گویند. در مورد پروتکل TDMA در شبکه‌های بی‌سیم می‌توانید کدهای ns/mac-tdma~ را بررسی کنید.

۴-۲-۵ انواع مختلف پروتکل‌های مسیریابی در شبکه‌های محرک

در زیر چهار نوع پروتکل مجزایی که در شبکه‌های متحرک در ns استفاده می‌شود را بیان کرده‌ایم. این پروتکل‌ها dsdv,dsr,aodv و tora هستند.

۴-۲-۵-۱ DSDV

در این پروتکل پیغام‌های مسیریابی بین گره‌های متحرک همسایه تبادل می‌شود (همسایه‌هایی که در بازه رادیویی هم هستند). به روزرسانی جدول مسیریابی وقتی که تغییری در مسیرهای شبکه ایجاد می‌شود اتفاق خواهد افتاد. بسته به روزرسانی جدول تا زمانی که در جدول ثبت نشوند در cache باقی خواهند ماند. در این روش مسیریابی مسیرهایی که شماره ترتیب بیشتری دارند استفاده خواهند شد.

---

1. Direct-Sequence Spread-Spectrum
2. Time Division Multiple Access

شماره ترتیب در واقع مشخص می‌کند که به مسیر موردنظر آخرین و بهترین مسیر ثبت شده کدام است. اگر شماره دو مسیر به یک مقصد یکسان باشند آن مسیری که تعداد گام کمتری دارد انتخاب خواهد شد. اگر گره کشف کند که مسیر به یک مقصد از بین رفته تعداد گام را بی‌نهایت خواهد کرد و شماره ترتیب آن را به روز خواهد کرد (می‌افزاید) اما کمتر از مقدار مسیرهای دیگر به آن مقصد است.

از پورت 255 برای ارسال و دریافت داده‌های به روزرسانی استفاده می‌شود (پورت dmux در برنامه‌نویسی C++). در لایه بالاتر بعد از به روزرسانی جدول بسته به همسایه ارسال رو به جلو خواهد شد. (این پروتکل در ns/tcl/movility/dsdv.tcl~ پیاده‌سازی شده است).

DSR ۲-۵ -۲-۴

این پروتکل مخفف کلمه Dynamic Routing Protocol است. توجه داشته باشید که گره‌های متحرک یا ثابت متفاوت بوده و در گره‌های متحرک باید تمام بسته‌های دریافت شده توسط گره به وسیله routing agent اداره شوند. در DSR agent هر بسته داده برای اطلاعات مسیر منبع باید چک شود اگر بسته برای اطلاعات مسیریابی باشد بسته را رو به جلو ارسال خواهد کرد. اگر بسته داده‌ای رسیده دارای آدرس مقصد نباشد این پروتکل آن را به تمام همسایه‌های خود broadcast خواهد کرد. اگر همسایه اطلاعات مسیریابی درخواست کند گره این اطلاعات را به آن ارسال خواهد کرد. در اینجا DSR از پورت 255 برای دریافت و پردازش بسته‌های مسیریابی (routing agent) استفاده می‌کند. (اطلاعات بیشتر درباره DSR در فایل‌های ns/dsr~ و ns/tcl/mobility/dsr.tcl~ قرار دارند).

TORA ۳-۵-۲-۴

این پروتکل بر پایه الگوریتم link reversal یا لینک معکوس‌ساز است. در هر گره یک کپی از tora برای هر مقصد اجرا می‌شود. وقتی یک گره نیاز به تشخیص مسیر به یک مقصد دارد یک پیغام QUERY شامل آدرس مقصد را broadcast می‌کند که اعلان کند به آدرس مقصد نیاز دارد. این بسته تا زمانی که به دست مقصد یا یک گره آدرس مقصد که می‌داند برسد شبکه را طی خواهد کرد. سپس دارندهٔ آدرس مقصد یا خود مقصد بسته UPDATE را به فرستنده ارسال خواهد کرد. در این بین گره‌های میانی نیز جداول خود را به روز خواهند کرد. توجه داشته باشید که در این جداول آدرسها براساس وزن آن‌ها که همان طول کمترین مسیر به مقصد است نگهداری می‌شود. اگر آدرسی از مقصد یافت نشود گره درخواست کننده آدرس مقصد بسته CLEAR را به سایرین برای پاک کردن محتوای آدرس مقصد در جدول ارسال خواهد کرد.

روش tora در IMEP[1] برای تحویل مطمئن بسته‌های مسیریابی و تغییرات اتصال همسایه‌ها استفاده می‌شود. در این پروتکل از یک بسته ویژه استفاده شده که سربار را کاهش می‌دهد و به صورت دوره‌ای پیغام‌هایی از گره‌های لنگر (مکان مشخص و ثابت در ah hoc) به سایر گره‌ها با برچسب *HELLO* می‌دهد. (اطلاعات بیشتر در ns/tora~ و ns/tcl/mobility/tora/.tcl~ قرار دارد).

---

1. Internet MANET Encapsulation Protocol

AODV ۴-۵-۲-۴

این نوع مسیریاب در واقع از نوع بردار فاصله است و لازم نیست که گره‌ها مسیر به مقصد را نگه دارند (باید توجه داشت که همه آنها مورد استفاده قرار نمی‌گیرد). در نقاط انتهای یک اتصال که مسیرهای معتبر را دارند AODV نقشی را بازی نمی‌کند. این پروتکل از پیغام‌های جدا برای کشف و نگهداری لینک‌ها استفاده می‌کند که شامل Route Request RREQ، Route Reply RREP و Route Error RERR است. این پیغام‌ها از نوع پیغام‌های UDP هستند و سرآیند آن‌ها همان سرآیند IP معمولی هستند.

Aodv از یک شماره ترتیب مقصد برای هر ورودی مسیرها استفاده می‌کند. این شماره ترتیب توسط مقصد در جواب درخواست اطلاعات مسیریابی به درخواست‌کننده ارسال می‌شود. شماره ترتیب در یک چرخه آزاد مشخص می‌کند که کدام مسیرها به مقاصد مختلف تازه‌تر هستند. زمانی که گره قصد دانستن یک مسیر به سایرین را دارد یک بسته RREQ را به تازه‌ترین آدرس‌های تمامی آدرس‌های مقصد موجود ارسال می‌کند (در واقع بین آدرس‌های موجود به یک مقصد آنی انتخاب می‌شود که شماره ترتیب بزرگتری دارد). سپس مقصد RREP به فرستنده باز گردانده خواهد شد که مسیر را گزارش می‌دهد.

گره‌هایی که قسمتی از یک مسیر فعال هستند ممکن است که به صورت دوره‌ای اطلاعات اتصال را با پیغام‌های محلی Hello به صورت broadcast پخش کنند (در واقع می‌توانند به جای مقصد نیز RREP بدهند). اگر پیغام‌های Hello از یک همسایه در یک زمان آستانه‌ای دیگر دریافت نشوند گره متوجه می‌شود که آن اتصال از بین رفته است.

زمانی که یک گره متوجه می‌شود مسیر به یک همسایه معتبر نیست ابتدا ورودی جدول مسیریابی را حذف می‌کند و به همسایه‌هایی که آن مسیر را فعال می‌بینند خود یک پیغام RERR را ارسال می‌کند. این روال تکرار خواهد شد تا زمانی که یک پیغام RERR دریافت کند. یک منبع که پیغام RERR را دریافت می‌کند می‌تواند یک پیغام RREQ را دوباره ارسال کند. روش AODV اجازه‌ی اداره کردن لینک‌های غیرمستقیم را نمی‌دهد.

این پروتکل در واقع ترکیبی از دو پروتکل DSR و DSDV است که از قابلیت‌های route discovery و route maintenance پروتکل DSR و از قابلیت‌های hop by hop touting و sequence number و beacon پروتکل DSDV استفاده می‌کند. تمام گره‌های میانی در اینجا شرکت خواهند کرد و جدول خود را به روز خواهند رساند (اطلاعات بیشتر در ns/aodv and ns/tcl/lib/ns-lib.tc قرار دارند).

۴-۳ اضافه‌کردن پروتکل جدید در NS

در ادامه یک پروتکل جدید و ساده به نام ping protocol را اضافه خواهیم کرد که در آن یک گره قادر است به سایر گره‌ها یک پاکت ارسال کند و آن گیرنده هم آن را سریع برگرداند. با این کار زمان رفت و برگشت یا round-trip را محاسبه کرد.

۴-۳-۱ ایجاد یک فایل سرآیند[1]

در این فایل سرآیند جدید ابتدا ما ساختار داده سرآیند پاکت ping جدید را مشخص می‌کنیم که داده‌های مرتبط را انتقال خواهد داد.

```
struct hdr_ping {
char ret;
double send_time;
};
```

کارکتر ret در صورتی که بسته از فرستنده به گیرنده ping شده باشد '0' می‌شود و اگر پاسخ ping باشد '1' خواهد شد. متغیر send_time برحسب زمانی است که در هنگام ارسال روی بسته زده می‌شود و بعداً در محاسبه زمان رفت و برگشت محاسبه خواهد شد.

تکه کد زیر کلاس PingAgent را به عنوان زیر کلاسی از کلاس Agent تعریف می‌کند.

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

تابع PingAgent() سازنده را تعریف می‌کند و متدهای command() و recv() در این وراثت از نوع int تعریف شده‌اند. از 'off_ping_' برای دسترسی به سرآیند یک پاکت ping استفاده خواهد شد. دقت داشته باشید متغیرها با محدوده اشیاء محلی معمولاً در آخرشان '_' خواهد داشت.

در زیر کد کامل ping.h را ملاحظه می‌فرمائید:

```
/*
* File: Header File for a new 'Ping' Agent Class for the ns
* network simulator
*
*/
#ifndef ns_ping_h
#define ns_ping_h
#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"
struct hdr_ping {
char ret;
double send_time;
};
class PingAgent : public Agent {
public:
PingAgent();
int command(int argc, const char*const* argv);
void recv(Packet*, Handler*);
protected:
int off_ping_;
};
#endif
```

---

1. Header

۴-۴ کدهای C++

ارتباط بین کدهای tcl و C++ تعریف شده می‌باشد و لازم نیست که به‌طور کامل یاد بگیرید. در زیر کدهای ارتباط C++ برای کلاس سرآیند ping را مشاهده می‌کنید.

```
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;
```

قست بعدی کد سازنده کلاس PingAgent است که متغیرها طوری مقید (متصل bind) می‌کندکه هم در Tcl و هم در C++ قابل دسترسی باشند.

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

تابع command() زمانی فراخوانی می‌شود که یک دستور Tcl برای کلاس 'PingAgent' اجرا می‌شوند. در این حالت این تابع '$pa send' است (فرض کنیم که 'pa' یک نمونه از کلاس Agent/Ping است) و دلیل آن این است که ما قصد داریم پاکت‌هایی را از یک agent به ping agent دیگری ارسال کنیم. به طور اساسی شما دستور 'command' را تجزیه[1] خواهید کرد و اگر تابع مطابق پیدا نشود شما دستور را با آرگمان‌ها به تابع 'command' در کلاس پایه ارسال خواهید کرد (در این حالت (Agent::command()). کد تابع 'command()' طولانی است و توضیح آن خط به خط در کد نوشته شده است.

```
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
```

1. Parse

```
// command has been processed
46
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));
}
```

تابع 'recv()' عملی را مشخص می‌کند که وقتی یک پاکت می‌رسد باید آن را انجام دهیم. اگر فیلد 'ret' برابر 0 باشد پاکت با همان برچسب زمانی ارسال خواهد شد و فیلد 'ret' برابر با 1 خواهد شد. اگر 'ret' برابر با 1 باشد (که توسط کاربر در tcl تعیین می‌شود) یک تابع tcl فراخوانی و پردازش خواهد شد. در این پردازش زمان رفت و برگشت برای کاربر نشان داده خواهد شد. آدرس NodeShift شماره فرستنده را به ما می‌دهد.

در زیر فایل کامل ping.cc که در داخل آن متد recv() نیز پیاده‌سازی شده را ملاحظه می‌فرمائید.

```
/*
* File: Code for a new 'Ping' Agent Class for the ns
* network simulator
* Author: Marc Greis (greis@cs.uni-bonn.de), May 1998
*
*/
#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_);
}
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));
}
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);
}
}
```

زمانی که تابع 'recv' فراخوانی می‌شود مهمترین قسمت فراخوانی tcl.eval() است که با شماره گره ping شده و زمان رفت و برگشت به عنوان پارامتر فراخوانی می‌شود. در بخش‌های بعد می‌بینیم که

چگونه کد برای این تابع نوشته می‌شود. قبل از این‌ها باید ببینیم که چگونه سایر فایل‌ها پیش از اینکه ns کامپایل دوباره شود ویرایش خواهند شد.

تغییرات لازم:

شما برخی تغییرات برروی فایل‌های منبع ns برای اینکه agent جدید اضافه کنید لازم دارید. به ویژه اگر از فرمت پاکت جدید هم استفاده کنید. پیشنهاد می‌کنیم که تغییرات خود را با توضیحات لازم در برنامه اعمال کنید (گذاشتن command و استفاده از #ifdef و غیره). بنابراین شما به راحتی می‌توانید تغییرات خود را برداشته و آن‌ها را در یک ورژن جدید برای ns بگذارید.

حال که ما به یک نوع از پاکت برای ping agent نیاز داریم، گام اول ویرایش فایل 'packet.h' است. در آنجا شما می‌توانید تعاریف برای ID پروتکل‌های مختلف را پیدا کنید (مثل PT_TCP و PT_TELNET و غیره). در آنجا تعاریف جدید برای PT_PING را اضافه کنید. در ورژن ویرایش شده packet.h ما تعدادی از خطوط enum packet_t{} شبیه به کدهای زیر بود (شاید بعدها تغییراتی در آن اعمال شود).

```
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
};
```

همچنین شما باید 'p_info()' را در همان فایلی که شامل 'Ping' است را ویرایش کنید.

```
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";
}
.....
}
```

به یاد داشته باشید که قبل از انجام دستور 'make' باید دستور 'make depend' را انجام دهید که در غیر این صورت ممکن است که این دو فایل کامپایل دوباره نشوند.

همچنین باید فایل 'tcl/lib/ns-default.tcl' را نیز ویرایش کرد. در این فایل تمام مقدارهای پیش‌فرض برای اشیاء Tcl تعریف شده است. خط زیر برای تنظیم اندازه پاکت پیش‌فرض برای Asent/Ping در این فایل درج می‌شود.

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

همچنین شما باید یک ورودی برای پاکت‌های ping جدید در فایل 'tcl/lib/ns-packet.tcl' در لیست ابتدای فایل درج کنید. تکه کدها شبیه به زیر خواهد بود.

```
{ SRMEXT off_srm_ext_}
{ Ping off_ping_ }} {
set cl PacketHeader/[lindex $pair 0]
```

آخرین تغییر، تغییری است که در فایل Makefile انجام خواهید داد. شما فایل 'ping.o' را به لیست فایل‌های شئ برای ns را اضافه می‌کنید.
لیست شما باید شبیه به زیر باشد.

```
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 \
```

شما باید قادر به کامپایل با تایپ 'make' در دایرکتوری ns باشید.

۵-۴ کد Tcl

حال کد کامل برای یک مثال Tcl برای Ping agent را برای شما نشان خواهیم داد. به شما نشان خواهیم داد که چگونه روال 'recv' وقتی که یک پاکت 'echo' ping می‌رسد از تابع 'recv()' در کد C++ فراخوانی خواهد شد.

```
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] received ping answer from \
$from with round-trip-time $rtt ms."
}
```

کد گفته شده در بالا به سادگی قابل فهم است. چیز جدیدی که دارد دست‌یابی‌ها به متغیر عضو 'node_' از کلاس پایه 'Agent' برای گرفتن شماره گره node id (آن گره‌ای که به آن ping کرده‌ایم) می‌باشد. در ادامه کد کامل ping.tcl برای شما نوشته شده:

```
 #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
```

حال که شما تا حدودی در این کار تجربه پیدا کردید اگر در فیلد 'ret' مقدار را بعد از دریافت '1' نکنیم چه اتفاقی خواهد افتاد؟

همیشه شما می‌توانید برخی کدها را برای اجازه دادن به ارسال پاکت با $pa send $node (به‌طوری که 'pa' یک ping agent بوده و 'node' یک گره است) بدون اتصال به 'pa' با یک ping agent روی 'node' در ابتدا انتخاب کنید، البته در ابتدا این کار پیچیده به نظر می‌رسد.