به نام خدا

شبیه سازی شبکه های کامپیوتری

نرم افزار Ns

گردآوری و تالیف :

محمد رضائی

دانشگاه مهندسی فناوریهای نوین قوچان

فهرست

۱ اصول و مبانی

۱.۱ دانلود و نصب ns در nam

شما می توانید ns را هم از بسته های نرم افزاری متعددی (Tcl/Tk, otcl, etc) بگیرید و یا می توانید تمامی اینها را به صورت یک بسته نرم افزاری دانلود کنید، که البته راه دوم را برای شروع پیشنهاد می کنم، خصوصا اگر نمیدانید که چه نرم افزارهایی را در سیستم خود دارید و اینکه این نرم افزارها دقیقا کجا نصب شده اند.

عیب این بسته ی نرم افزاری حجم و سایز آن است چرا که مولفه هایی را شامل می شود که پس از کامپایل و تنظیم کردن ns و nam به کارتان نمی آید. برای مراحل اولیه همین بسته خوب است و می توانید بعدا از همان بسته های تک نرم افزاری استفاده کنید.

توجه: بسته نرم افزاری چندکاره تنها در سیستم های Unix عمل میکنند.شما میتوانید این بسته نرم افزاری جامع را از ns download page در UCB دانلود کنید و در صورت مواجه شدن با هرگونه مشکلی در نصب نرم افزار installation problems page در سرورشان مراجعه کنید و چنانچه بدین طریق هم مشکلتان حل نشد می توانید به ns-users mailing list مشکلتان را مطرح کنید.

پس از اینکه نصب بسته ی نرم افزاری تمام شد چنانچه بسته ی نرم افزاری چند کاره ns را نصب کردید باید مسیر های ns و nam را به مسیرهای پیش فرض سیستم عامل اضافه کرد.

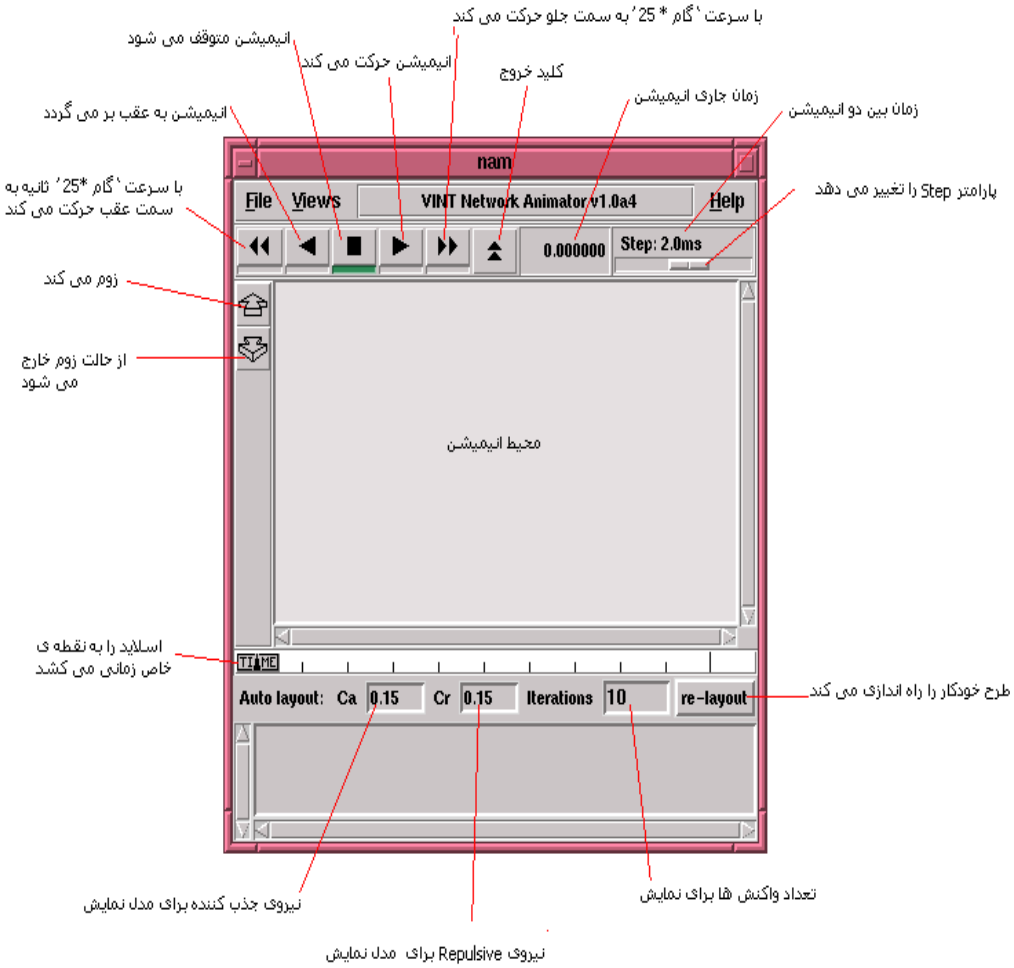
چناچه بسته ی نرم افزاری چند کاره ی ns را نصب کردید این نرم افزار باید در-'ns allinone/otcl/' باشد. در سیستم های سولاریس شما باید این مسیر را به متغییر محیطی 'LD_LIBRARY_PATH' اضافه کنید. در صورت نیاز به کمک برای دیگر سیستم ها installation problem page یا ns-users mailing list و یا Unix gurus محلی خودتان مراجعه کنید.

۱.۲ راه اندازی ns

با این فرض که شما در مسیر ns قرار دارید و یا اینکه مسیر ns را به مسیرهای پیش فرض سیستم عامل اضافه کرده اید، ns را با دستور 'ns <tclscript>' راه اندازی می کنید. <tclscript> اسم فایل اسکریپت Tcl میباشد که برنامه شبیه سازی را تعریف میکند(به عبارت دیگر وقایع و توپولوژی) شما می توانید بدون هیچ درگیری ns را راه اندازی کنید و وارد فرمانهای Tcl شوید، ولی این کار قطعا دشوارتر است.

۱.۳ راه اندازی ***nam***

برای راه اندازی nam هم می توان از دستور 'nam <nam-file>' استفاده کرد که در آن '<nam-file>' نام فایل ورودی به nam است که توسط ns بوجود آمده است. در ذیل شما می توانید screenshot پنجره ی nam را مشاهده کنید که در آن توضیح داده شده شده اند.



nam ۱-۱

۲ Tcl script اولیه

در این بخش، قرار است شما یک اسکریپت Tcl برای ns بوجود آورید که یک توپولوژی ساده را شبیه سازی می کند. در این بخش شما یاد می گیرید که چگونه لینک و گره (node) ایجاد کنید، چگونه اطلاعات را از یک گره به گره دیگر بفرستید.

چگونه یک صف را مونیتور کرده و بر آن نظارت کنید و اینکه برای مجسم سازی خودتان چگونه از اسکریپت شبیه سازی خود،nam را راه اندازی کنید.

۲.۱ چگونگی آغاز راه اندازی

اکنون شما باید یک الگو (Template) بنویسید که بتوانید از آن برای تمامی اسکریپت های Tcl اولیه استفاده کنید. شما می توانید اسکریپت های Tcl خود را در هر ویرایشگر متنی مثل Joe یا emacs بنویسید. پیشنهاد می کنم این مثال اول را بصورت 'example۱.tcl' نامگذاری کنید.

پیش از هر کاری باید یک شی شبیه ساز بوجود آورید. با دستور زیر این کار انجام می شود:

```
set ns [new Simulator]
```

اکنون برای نوشتن چیزی که قرار است آن را برای داده های پیمایش برنامه nam بکارگیریم فایلی را باز می کنیم:

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

خط اول این دستور فایل out.nam را برای نوشتن باز کرده و به آن نام nf می دهد. در خط دوم به شی شبیه ساز می گوییم که خط اول را برای نوشتن همه ی داده های شبیه ساز که متناسب با nam داخل این فایل می باشد بوجود آورده ایم.

مرحله ی بعدی این است که یک پروسه ی پایان (finish) را اضافه کنیم که فایل پیمایش برنامه را بسته و برنامه nam را شروع کند.

```
proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam &
```

```
    exit ۰
}
```

خط بعدی به Object شبیه ساز می گوید که پس از ۵ ثانیه بعد از زمان شبیه سازی پروسه ی 'finish' را اجرا کند.

```
$ns at ۵.۰ "finish"
```

ns این امکان را در اختیار شما می گذارد که به شیوه ای بسیار آسان تنها با استفاده از دستور 'at' وقایع را برنامه ریزی کنید.

در نهایت خط آخر شبیه سازی را آغاز می کند.

```
$ns run
```

اکنون شما می توانید فایل را ذخیره کرده و سعی کنید با استفاده از 'ns example۱.tcl' آن را راه اندازی کنید، گرچه پیام خطایی چون 'nam: empty trace file out.nam' دریافت می کند ، چراکه تاکنون ما هیچ واقعه یا شی (لینک ، گره ،...) را تعریف و مشخص نکرده ایم.

۲.۲ دو گره ، یک لینک

در این بخش یک توپولوژی بسیار ساده با دو گره که توسط یک لینک بهم مرتبط شده اند را تعریف می کنیم.دو خط ذیل دو گره را تعریف و مشخص می کنند. (شما باید آن کد را در این بخش قبل از خط '$ns run' و یا حتی بهتر است که قبل از خط '$ns at ۵.۰ "finish" وارد کنید)

```
set n۰ [$ns node]
```

set n۱ [$ns node]

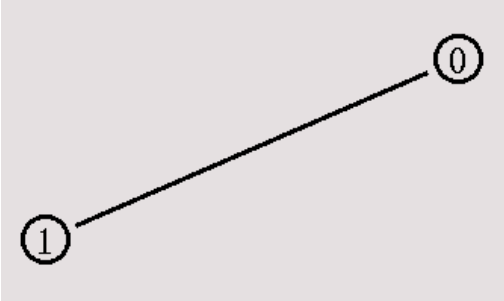
یک شی گره جدید با دستور '$ns node' بوجود می آید و متغییر های 'n۰' و'n۱' را بدانها تخصیص می دهد. خط بعدی دو گره را به هم متصل می کند.

$ns duplex-link $n۰ $n۱ ۱Mb ۱۰ms DropTail

این خط به شی شبیه ساز می گوید که گره های n۰ و n۱ را توسط یک لینک دوتایی با گستره ی باند ۱مگابایتی ، تاخیر ۱۰ میلی ثانیه و یک صف DropTail به یکدیگر متصل کند.

اکنون می توانید فایلتان را ذخیره کرده و اسکریپت را با 'ns example۱.tcl' راه اندازی کنید.

Nam بصورت خودکار اجرا خواهد شد و شما باید یک نتیجه را مشاهده کنیدکه شبیه تصویر زیر است.



شکل ۲-۱ شبکه ای با دو گره

۲.۳ ارسال داده ها و اطلاعات

گرچه از آنجاییکه شما تنها می توانید توپولوژی را مشاهده کنید ، این مثال چندان رضایت بخش نیست ولی هیچ اتفاق خاصی رخ نمی دهد و مرحله ی بعدی این است که مقداری اطلاعات از n۰ به n۱ ارسال کنیم .در ns داده ها و اطلاعات از یک عامل به عامل دیگر ارسال می شود بنابراین مرحله ی بعدی این است که شی عاملی بوجود آوریم که داده ها و اطلاعات را از گره n۰ ارسال دارد و همچنین باید شی عامل دیگری بوجود آوریم که داده ها و اطلاعات را در گره n۱ دریافت کند.

```
#Create a UDP agent and attach it to node n۰

set udp۰ [new Agent/UDP]

$ns attach-agent $n۰ $udp۰

# Create a CBR traffic source and attach it to udp۰

set cbr۰ [new Application/Traffic/CBR]

$cbr۰ set packetSize_ ۵۰۰

$cbr۰ set interval_ ۰.۰۰۵

$cbr۰ attach-agent $udp۰
```

این خط یک عامل UDP بوجود آورده و آن را به گره n۰ متصل می کنند سپس یک اجراکننده ترافیک CBR را به عامل UDP متصل می سازند. سایز بسته به اندازه ی ۵۰۰ بایت تنظیم شده و هر بسته هر ۰/۰۰۵ ثانیه ارسال میشود. (به عبارت دیگر ۲۰۰ بسته در هر ثانیه).

خطوط بعدی یک عامل تهی (null) را بوجود می آورند و آن را به گره n۱ وصل می کند.

```
set null۰ [new Agent/Null]

$ns attach-agent $n۱ $null۰
```

اکنون این دو عامل باید بهم متصل شوند.

```
$ns connect $udp۰ $null۰
```

اکنون باید زمان ارسال داده ها و اطلاعات و زمان توقف ارسال را به عامل CBR اطلاع دهیم.

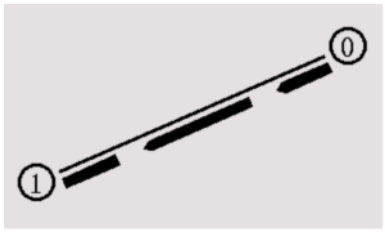
توجه: احتمالا بهترین کار این است که خطوط ذیل را درست قبل از خط "finish" ۵.۰ $ns at' قرار دهیم.

$ns at ۰.۵ "$cbr۰ start"

$ns at ۴.۵ "$cbr۰ stop"

این کد نیز خود باید توضیح دهنده خوب باشد.

اکنون می توانید فایل را ذخیره کرده و شبیه سازی را مجددا آغاز کنید. زمانی که روی دکمه ی شروع (Play) در پنجره ی nam کلیک کردید مشاهده خواهید کرد که پس از ۰.۵ ثانیه شبیه سازی گره ۰ شروع میکند به ارسال بسته های اطلاعاتی به گره ۱.میتوانید با 'Step' سرعت nam را کاهش دهید.



شکل ۲-۲ دو گره متصل

شما میتوانید برای نظارت بر بسته ای، روی هر یک از بسته های پنجره ی nam کلیک کنید. شما همچنین می توانید برای دریافت گراف های آماری مستقیما روی خود لینک کلیک کنید.

۳ جالب تر کردن پروسه

در این بخش به تعریف یک توپولوژی با ۴ گره می پردازیم که در آن یک گره به عنوان مسیریاب عمل کرده و اطلاعات را ارسال میکند و دو گره دیگر این اطلاعات را به گره چهارم ارسال می کنند ،پس از یافتن شیوه ای جهت تشخیص و تمیز دادن اطلاعاتی که از دو گره به یکدیگر ارسال می شوند را برایتان توضیح می دهم و نشان میدهم که چگونه می توان برای اینکه ببینیم چقدر یک صف پر است ان صف را کنترل و نظارت کنیم و اینکه از چه تعداد بسته چشم پوشی می شود.

۳.۱ توپولوژی (ساختار)

درست مثل همیشه اولین مرحله تعریف و مشخص کردن توپولوژی است. همانطور که قبلا گفتم این کد همواره یکسان خواهد بود. شما همیشه باید یک شی شبیه ساز بوجود آورید و باید شبیه سازی را با دستوری یکسان راه اندازی کنید و اگر می خواهید nam را بصورت خودکار راه اندازی کنید ، همیشه باید یک فایل پیمایش برنامه باز کرده و آن را مقداردهی کرده و پروسه ای را تعریف کنیم که آن را بسته و nam را راه اندازی کند.

اکنون خطوط زیر را بداخل کد وارد کنید تا چهار گره بوجود آورید.

```
set n۰ [$ns node]
set n۱ [$ns node]
set n۲ [$ns node]
set n۳ [$ns node]
```

یک قسمت کد Tcl کد در پایین آورده شده ، سه لینک دوتایی بین گره ها بوجود می آورد.

```
$ns duplex-link $n۰ $n۲ ۱Mb ۱۰ms DropTail
$ns duplex-link $n۱ $n۲ ۱Mb ۱۰ms DropTail
$ns duplex-link $n۳ $n۲ ۱Mb ۱۰ms DropTail
```

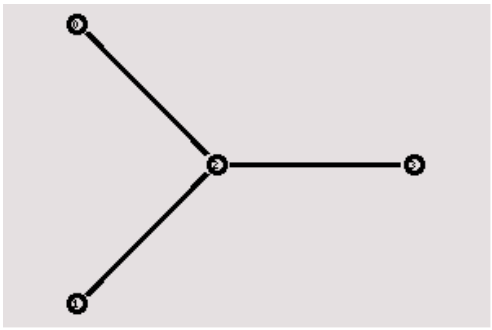
اکنون می توانید اسکریپت را ذخیره و راه اندازی کنید. ممکن است متوجه این نکته شوید که این توپولوژی در nam تا حدی نامناسب بنظر می آید. با کلیک روی کلید 're-layout' آن را بهتر جلوه دهید. ولی اگر شکل کلی را تنظیم کنید ظاهرش زیباتر می شود. سه خط بعدی را اسکریپت Tcl خود افزوده و دوباره آن را راه اندازی کنید.

$ns duplex-link-op $n۰ $n۲ orient right-down

$ns duplex-link-op $n۱ $n۲ orient right-up

$ns duplex-link-op $n۲ $n۳ orient right

وقتی به این توپولوژی در پنجره ی nam نگاه کنید، احتمالا می تولنید متوجه عملکرد و نقش این کد شوید.این توپولوژی در پنجره ی nam باید مثل تصویر زیر باشد.



شکل ۳-۱ چیدن گره ها

توجه داشته باشید که طرح کلی خود بخود قسمتهای مرتبط nam حذف شده اند، از اکنون طرح کلی در دست شماست.گزینه های چرخش یک لینک ، به راست،چپ،بالا،پایین و یا ترکیبی از این گزینه ها می باشد.

۳.۲ رویدادها

اکنون دوعامل UDP با منابع ترافیکی CBR بوجود آورده و آنها به گروه های n۰ و n۱ متصل می کنیم. سپس یک عامل تهی(null) بوجود آورده و آن را به گروه n۳ متصل میکنیم.

#Create a UDP agent and attach it to node n۰

set udp۰ [new Agent/UDP]

$ns attach-agent $n۰ $udp۰

```
# Create a CBR traffic source and attach it to udp۰

set cbr۰ [new Application/Traffic/CBR]

$cbr۰ set packetSize_ ۵۰۰

$cbr۰ set interval_ ۰.۰۰۵

$cbr۰ attach-agent $udp۰

#Create a UDP agent and attach it to node n۱

set udp۱ [new Agent/UDP]

$ns attach-agent $n۱ $udp۱

# Create a CBR traffic source and attach it to udp۱

set cbr۱ [new Application/Traffic/CBR]

$cbr۱ set packetSize_ ۵۰۰

$cbr۱ set interval_ ۰.۰۰۵

$cbr۱ attach-agent $udp۱

set null۰ [new Agent/Null]

$ns attach-agent $n۳ $null۰
```

دو عامل CBR باید به عامل تهی(null) متصل شود.

```
$ns connect $udp۰ $null۰
```

$ns connect $udp۱ $null۰

ما می خواهیم که عامل اول CBR در ۰/۵ ثانیه ارسال اطلاعات را شروع کرده و در ۴/۵ ثانیه آن را تمام کند . در حالی که عامل دوم CBR در ۱/۰ ثانیه شروع به ارسال اطلاعات کرده و در ۴/۰ ثانیه متوقف می شود.

$ns at ۰.۵ "$cbr۰ start"

$ns at ۱.۰ "$cbr۱ start"

$ns at ۴.۰ "$cbr۱ stop"

$ns at ۴.۵ "$cbr۰ stop"

زمانیکه با 'ns example۲.tcl' اسکریپت را راه اندازی کردید متوجه می شوید که نسبت به ترافیکی که n۲ به n۳ می تواند حمل کند، ترافیک لینک های n۰ به n۲ و n۱ به n۲ بیشتر است. یک ارزیابی ساده معید این موضوع است : ما در هر ثانیه ۲۰۰ بسته را به هر یک از دو لینک اولیه ارسال می کنیم و سایز هر بسته نیز ۵۰۰ بایت است .

گستره ی باندی ۸/۰ مگابایت در هر ثانیه را برای لینک های n۰ به n۲ و n۱ به n۲ موجب می شود که در کل یک گستره ی باندی ۱/۶ مگابایت در هر ثانیه را بوجود می آورد. ولی لینک بین n۲ و n۳ تنها ظرفیتی معادل یک مگابایت در هر ثانیه را دارا می باشد ، بنابراین کاملا واضح است که از برخی بسته ها چشم پوشی می شود (و دور ریخته می شود) ولی کدام بسته ها؟ هر دو جریان مشکی هستند. بنابراین تنها راه اینکه بفهمیم چه اتفاقی برای بسته ها رخ داده این است که با کلیک کردن بر روی آنها ، آنها را در nam کنترل کنید. در دو بخش بعدی به شما نشان می دهم که چگونه جریان های متفاوت را از یکدیگر تشخیص دهید و اینکه چگونه می توانید بفهمید که در واقع چه اتفاقی در صف لینک n۲ به n۳ رخ داده است.

۳.۳ علامتگذاری جریانها

دو خط ذیل را به تعاریف عامل CBR خود بیافزایید.

$udp۰ set class_ ۱

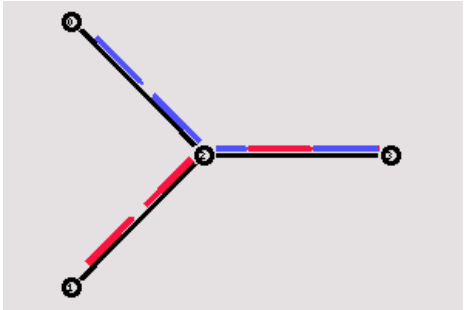
$udp۱ set class_ ۲

اکنون ترجیحا در ابتدای کار و پس از اینکه شبیه ساز بوجود آمد کد زیر را به اسکریپت Tcl خود بیافزایید زیرا این عمل بخشی از راه اندازی شبیه ساز می باشد.

$ns color ۱ Blue

$ns color ۲ Red

این کد به شما اجازه می دهد که رنگهای متفاوتی را برای شناسه هر جریان تنظیم کنید.



شکل ۳-۲ تمایز جریانها

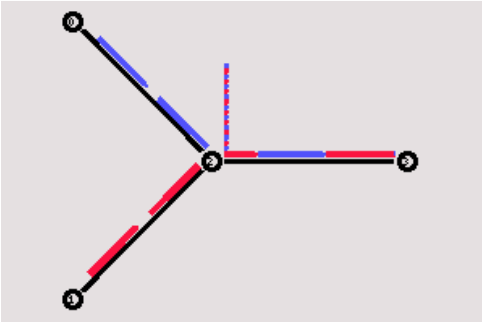
اکنون شما دوباره می توانید اسکریپت را راه اندازی کنید و یک جریان آبی باشد در حالی که جریان دیگر قرمز است .اگر یک لحظه به لینک n۲ به n۳ نگاه کنید متوجه می شوید که پس از مدتی توزیع جریان جریان بین بسته های قرمز و آبی دیگر خیلی عادلانه نیست (حداقل در سیستم من اینگونه است) در بخش بعدی بشما نشان خواهم داد که چگونه می توانید به درون صف این لینک نگاه کنید تا بفهمید در آنجا چه می گذرد.

۳.۴ مونیتورکردن یک صف

برای نظارت بر صف لینک n۲ به n۳ شما تنها باید خط زیر را به کد خود بیافزایید.

$ns duplex-link-op $n۲ $n۳ queuePos ۰.۵

دوباره ns را راه اندازی کنید و پس از چند لحظه تصویری مشابه تصویر ذیل مشاهده خواهید کرد.

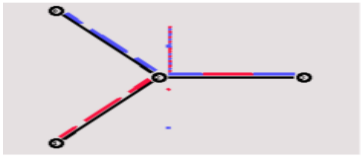


شکل ۳-۳ مونیتورکردن یک صف

اکنون می توانید بسته های موجود در صف را مشاهده کنید و پس از یک لحظه شما حتی می توانید چگونگی حذف بسته ها را مشاهده کنید. شما نمی توانید از یک صف ساده ی DropTail انتظار عدالت داشته باشید. بنابراین سعی میکنیم که با استفاده از یک SFQ (صف بندی عادلانه) برای لینک n۲ به n۳ صف بوجود آوریم. تعریف لینک را برای لینک بین n۲ و n۳ بصورت خط زیر تغییر دهید.

۰.۵ $ns duplex-link-op $n۲ $n۳ queuePos

اکنون تشکیل صف باید عادلانه باشد و میزان یکسانی از بسته های آبی و قرمز باید حذف شوند.



شکل ۳-۴ مونیتورکردن یک صف عادلانه

۴ شبکه های پویا

در این بخش قصد دارم مثالی برای یک شبکه ی پویا را به شما نشان دهم. شبکه پویا جایی است که در آن مسیر یابی با برقرار نشدن پیوند لینک منطبق می شود و به شما نشان می دهم که به جای تخصیص نامی خاص به هر گره چگونه گره های بیشتری را در آرایه ی Tcl نگهدارید.

۴.۱ بوجود آوردن یک توپولوژی بزرگتر

پیشنهاد می کنم که اسکریپت Tcl را برای این مثال 'example۳.tcl' بنامید.

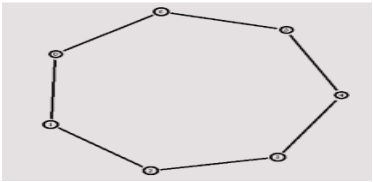
درست مانند همیشه در ابتدا توپولوژی باید ایجاد شود.گرچه این دفعه از شیوه ی متفاوتی استفاده می کنیم که برای ایجاد توپولوژی های بزرگتر راحت تر می باشد.کد ذیل ۷ گره را ایجاد کرده و آنها را در آرایه ی n۰ ذخیره می کند.

```
for {set i ۰} {$i < ۷} {incr i} {
  set n($i) [$ns node]
}
```

توجه داشته باشید که آرایه ها درست مانند دیگر متغییرها در Tcl در ابتدا نیازی به معرفی ندارند. اکنون می خواهیم جهت ایجاد یک توپولوژی مدور ، گره ها را بیکدیگر متصل کنید. ممکن است کد زیر در ابتدا کمی پیچیده بنظر برسد.

```
for {set i ۰} {$i < ۷} {incr i} {
  $ns duplex-link $n($i) $n([expr ($i+۱)%۷]) ۱Mb ۱۰ms DropTail
}
```

این حلقه ' for ' همه گره ها را به گره بعدی در آرایه وصل میکند بجز گره آخر که به گره اول متصل می شود. برای عملی کردن این کار اپراتور " ٪ " را بکار برده ام. اکنون وقتی اسکریپت را اجرا می کنید ممکن است توپولوژی در nam در ابتدا کمی عجیب بنظر برسد ولی پس از اینکه روی کلید 're-layout' کلیک کنید ساختار باید مانند شکل زیر باشد.



شکل ۴-۱ ایجاد یک توپولوژی مدور

۴.۲ برقرار نشدن پیوند لینک

مرحله ی بعدی اینست که مقداری اطلاعات از گره n(۰) به گره n(۱) ارسال کنیم.

```
#Create a UDP agent and attach it to node n(۰)
set udp۰ [new Agent/UDP]
$ns attach-agent $n(۰) $udp۰
# Create a CBR traffic source and attach it to udp۰
set cbr۰ [new Application/Traffic/CBR]
$cbr۰ set packetSize_ ۵۰۰
$cbr۰ set interval_ ۰.۰۰۵
$cbr۰ attach-agent $udp۰
set null۰ [new Agent/Null]
$ns attach-agent $n(۳) $null۰
$ns connect $udp۰ $null۰
$ns at ۰.۵ "$cbr۰ start"
$ns at ۴.۵ "$cbr۰ stop"
```

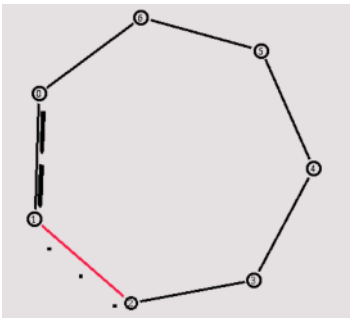
تنها تفاوت کد بالا با بخشهای پیش اینست که اکنون ما باید از مولفه های آرایه ی گره استفاده کنیم.

اگر اسکریپت را راه اندازی کنید خواهید دید همانطور که می توانست انتظار داشت ، ترافیک از کوتاه ترین راه از طریق گره ۱ و ۲ ، از گره ۰ به گره ۳ منتقل می شود . اکنون مشخصه ی جالب دیگری را اضافه می کنیم میگذاریم لینک برای یک ثانیه بین گره ۱ و ۲ (که ترافیک در حال استفاده از آن می باشد) به پایین برود.

$ns rtmodel-at ۱.۰ down $n(۱) $n(۲)

$ns rtmodel-at ۲.۰ up $n(۱) $n(۲)

اکنون می توانید مجددا اسکریپت را راه اندازی کنید سپس مشاهده خواهید کرد که لینک بین ثانیه های ۱.۰ و ۲.۰ پایین است و تمامی اطلاعاتی که از گره ۰ ارسال می شوند گم شده اند.



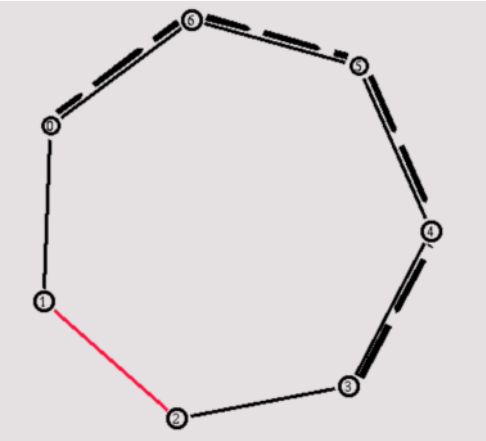
شکل ۴-۲ گم شدن بسته

اکنون نشان می دهم که چگونه از مسیریابی پویا برای حل این مشکل استفاده کنید . پس از اینکه شی شبیه ساز ایجاد شد خط زیر را به ابتدای اسکریپت Tcl خود بیافزایید.

$ns rtproto DV

شبیه سازی را دوباره آغاز کنید ، مشاهده خواهید کرد که چگونه در ابتدا تعداد زیادی بسته کوچک در شبکه در حرکتند . اگر سرعت nam را تا حدی پایین آوریدکه روی یکی از آنها کلیک کنید خواهید دید که آنها بسته های 'rtProtoDV' هستند که برای معاوضه ی اطلاعات مسیر یابی بین گره ها در حال استفاده می باشند.

وقتی که لینک مجددا در ثانیه ۱.۰ پایین می رود، مسیر یابی بروز رسانی شده و به ترافیک مسیر جدیدی از طریق گره های ۴ و ۵ و ۶ داده می شود.



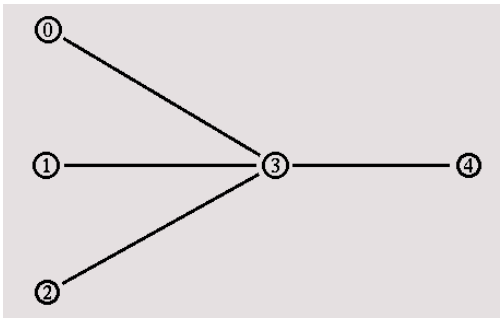
شکل ۴-۳ مسیریابی پویا

۵ ایجاد فایل های خروجی برای Xgraph

یک قسمت از بسته ی ns-allinone , 'xgraph' است. یک برنامه که می توان برای ایجاد یک نمایش تصویری و گرافیکی از نتایج شبیه سازی آن را بکار برد. در این بخش با چگونگی ایجاد فایل های خروجی در اسکریپت های Tcl آشنا می شویم. این فایل ها را می توانید بعنوان مجموعه های داده ها برای Xgraph بکار گیرید و همچنین چگونه استفاده کردن مولد های ترافیکی را به شما نشان خواهم داد.

۵.۱ توپولوژی و منابع ترافیکی

پیش از هر کاری ,توپولوژی زیر را ایجاد می کنیم:



۵-۱ توپولوژی و منابع ترافیکی

```
set n۰ [$ns node]
set n۱ [$ns node]
set n۲ [$ns node]
set n۳ [$ns node]
set n۴ [$ns node]

$ns duplex-link $n۰ $n۳ ۱Mb ۱۰۰ms DropTail
$ns duplex-link $n۱ $n۳ ۱Mb ۱۰۰ms DropTail
$ns duplex-link $n۲ $n۳ ۱Mb ۱۰۰ms DropTail
$ns duplex-link $n۳ $n۴ ۱Mb ۱۰۰ms DropTail
```

قصد داریم منابع ترافیکی را به گره های n۰ ,n۱ و n۲ متصل کنیم. ولی در ابتدا پروسه ای را می نویسیم که اضافه کردن منابع ترافیکی و مولد ها را به گره ها برایمان تسهیل می بخشد.

```
proc attach-expoo-traffic { node sink size burst idle rate } {
#Get an instance of the simulator
set ns [Simulator instance]
#Create a UDP agent and attach it to the node
set source [new Agent/UDP]
$ns attach-agent $node $source
#Create an Expoo traffic agent and set its configuration
parameters
set traffic [new Application/Traffic/Exponential]
$traffic set packetSize_ $size
```

```
$traffic set burst_time_ $burst
$traffic set idle_time_ $idle
$traffic set rate_ $rate
# Attach traffic source to the traffic generator
$traffic attach-agent $source
#Connect the source and the sink
$ns connect $source $sink
return $traffic
}
```

این تابع ۶ آرگومان را در بر میگیرد: یک گره یک سینک ترافیکی از پیش ایجاد شده, سایز بسته برای منبع ترافیکی زمان بیکاری وانفجار (برای توزیع نمایی) و نرخ نقطه ی اوج (Peak).

در ابتدا تابع یک منبع ترافیکی ایجاد می کند و آن را به گره متصل می سازد سپس یک شی Traffic/Expoo بوجود آورده, پارامتر های شکل بندی و ترکیب بندی آن را تنظیم کرده و قبل از اینکه منبع و سینک به یکدیگر متصل شوند آن شی را به منبع ترافیکی متصل می کند.این پروسه مثال خوبیست در مورد اینکه چگونه می توان اعمالی مثل یک اتصال یک منبع ترافیکی به تعداد زیادی گره که دوباره صورت میگیرند را کنترل کرد.اکنون برای اتصال منابع ترافیکی با نرخ های نقاط اوج به n۰, n۱ و n۲ وجهت ارتباط دادن آنها با سر سینک ترافیکی روی n۴ که باید در ابتدا بوجود آیند, این پروسه را بکار می گیریم.

```
set sink۰ [new Agent/LossMonitor]
set sink۱ [new Agent/LossMonitor]
set sink۲ [new Agent/LossMonitor]
$ns attach-agent $n۴ $sink۰
$ns attach-agent $n۴ $sink۱
```

```
$ns attach-agent $n۴ $sink۲
set source۰ [attach-expoo-traffic $n۰ $sink۰ ۲۰۰ ۲s ۱s ۱۰۰k]
set source۱ [attach-expoo-traffic $n۱ $sink۱ ۲۰۰ ۲s ۱s ۲۰۰k]
set source۲ [attach-expoo-traffic $n۲ $sink۲ ۲۰۰ ۲s ۱s ۳۰۰k]
```

در این مثال از شی های Agent/LossMonitor به عنوان سینک های ترافیکی استفاده می کنیم چرا که آنها میزان بایتهای دریافتی را ذخیره می کنند که می توان از آنها برای محاسبه ی گسترده ی باند استفاده کرد.

۵.۲ ثبت داده ها در فایل های خروجی

اکنون باید سه فایل خروجی را باز کنیم, خط های ذیل باید در اوایل اسکریپت Tcl آورده شوند.

```
set f۰ [open out۰.tr w]
set f۱ [open out۱.tr w]
set f۲ [open out۲.tr w]
```

این فایل ها به حدی که رسیدند, باید بسته شوند.بدین منظور ما از یک پروسه ی 'finish' تعدیل شده استفاده می کنیم تا این کار را انجام دهد.

```
proc finish {} {
    global f۰ f۱ f۲
    #Close the output files
    close $f۰
    close $f۱
    close $f۲
```

```
    #Call xgraph to display the results
    exec xgraph out۰.tr out۱.tr out۲.tr -geometry ۸۰۰x۴۰۰ &
    exit ۰
}
```

این پروسه نه تنها فایل های خروجی را می بندد بلکه xgraph را وادار به نمایش می کند. ممکن است بخواهید سایز پنجره (۴۰۰*۸۰۰) را با سایز صفحه ی خود تطبیق دهید.

اکنون می توانیم پروسه اس را بنویسیم که در واقع داده ها را در فایل های خروجی می نویسد.

```
proc record {} {
    global sink۰ sink۱ sink۲ f۰ f۱ f۲
    #Get an instance of the simulator
    set ns [Simulator instance]
    #Set the time after which the procedure should be called
again
    set time ۰.۵
    #How many bytes have been received by the traffic sinks?
    set bw۰ [$sink۰ set bytes_]
    set bw۱ [$sink۱ set bytes_]
    set bw۲ [$sink۲ set bytes_]
    #Get the current time
    set now [$ns now]
    #Calculate the bandwidth (in MBit/s) and write it to the files
    puts $f۰ "$now [expr $bw۰/$time*۸/۱۰۰۰۰۰]"
```

```
        puts $f۱ "$now [expr $bw۱/$time*۸/۱۰۰۰۰۰۰]"
        puts $f۲ "$now [expr $bw۲/$time*۸/۱۰۰۰۰۰۰]"
        #Reset the bytes_ values on the traffic sinks
        $sink۰ set bytes_ ۰
        $sink۱ set bytes_ ۰
        $sink۲ set bytes_ ۰
        #Re-schedule the procedure
        $ns at [expr $now+$time] "record"
}
```

این پروسه تعداد بایت های دریافت شده از سه سینک ترافیکی را بخواند.سپس گستره ی باند را در واحد مگابایت محاسبه کرده و همراه با زمان جاری پیش از تنظیم مجدد مقدار بایت ها بر سینک های ترافیکی آن را در سه فایل خروجی می نویسد, سپس خود را مجددا برنامه ریزی می کند.

۵.۳ انجام شبیه سازی

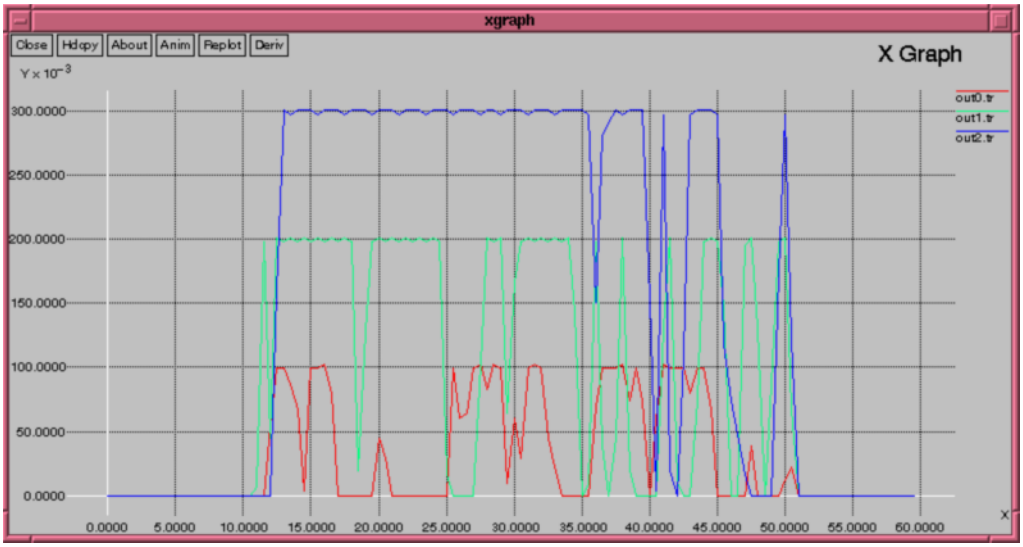
اکنون می توانیم وقایع زیر را برنامه ریزی کنیم:

```
$ns at ۰.۰ "record"
$ns at ۱۰.۰ "$source۰ start"
$ns at ۱۰.۰ "$source۱ start"
$ns at ۱۰.۰ "$source۲ start"
$ns at ۵۰.۰ "$source۰ stop"
$ns at ۵۰.۰ "$source۱ stop"
$ns at ۵۰.۰ "$source۲ stop"
```

$ns at ۶۰.۰ "finish"

$ns run

در ابتدا پروسه ی 'record' اجرا میشود و سپس این پروسه هر ۵ ثانیه یکبار مرتبا خود را مجددا برنامه ریزی می کند. بعد از آن سه منبع ترافیکی در عرض ۱۰ ثانیه شروع شده و در عرض ۵۰ ثانیه متوقف می شوند.در طی ۶۰ ثانیه پروسه ی 'finish' اجرا می شود.



xgraph خروجی ۵-۲

زمانی که شبیه سازی را شروع کردید,یک پنجره ی xgraph پس از مدتی باید باز شود که باید با پنجره شکل ۵-۲ ظاهرش یکسان باشد. توجه داشته باشید که فایل های خروجی ایجاد شده توسط پروسه ی 'record' می تواند توسط gnuplot نیز مورد استفاده قرار گیرد.

۶ اجرای شبیه سازی بی سیم در ns

در این بخش استفاده از مدل شبیه سازی بیسیم موبایل موجود در ns بررسی می شود. این بخش دارای دو قسمت است. در قسمت اول به چگونگی ایجاد و اجرای یک شبکه شبیه سازی بیسیم دو گره ای ساده می

پردازیم. در قسمت دوم به منظور ایجاد یک سناریوی نسبتا پیچیده تر ,مثال عنوان شده در قسمت اول را با تفصیل بیان می کنیم.

۶.۱ ایجاد یک سناریوی بی سیم ساده

در اینجا می خواهیم یک سناریوی بیسیم دو گره ای ساده را شبیه سازی کنیم. این توپولوژی از دو گره موبایل (متحرک) node_(۰) و node_(۱) تشکیل شده است. این گره های متحرک در محدوده ای که در این مثال ۵۰۰mX۵۰۰m تعریف شده است ,حرکت می کنند. در ابتدا گره ها از دو سمت مخالف این محدوده حرکت خود را آغاز میکنند. سپس در نیمه ی شبیه سازی این گره ها به سوی یکدیگر و در نیمه ی دوم شبیه سازی مجددا در جهت مخالف یکدیگر حرکت می کنند. یک ارتباط TCP بین این دو گره متحرک برقرار می شود. زمانی که گره ها در حوزه ی نزدیک به هم یکدیگر میرسند,بسته ها بین گره ها مبادله می شوند.همانطور که گره ها از هم دور می شوند,بسته ها نیز بیرون انداخته می شوند.درست مانند هر شبیه سازی ns دیگری برای شبیه سازی بیسیم, کار با ایجاد یک اسکریپت Tcl شروع می کنند.این فایل را simple-wireless.tcl میخوانیم.

یک گره متحرک از مولفه های شبکه ای چون Interface Queue (IfQ), like Link Layer (LL) ,MAC layer و گره های کانالی بی سیم که انتقال و دریافت سیگنالها را بر عهده دارند,تشکیل شده است. در ابتدای یک شبیه سازی بی سیم باید نوع هر یک از این مولفه های شبکه ای را مشخص و تعریف کنید. به علاوه باید دیگر پارامتر ها چون نوع آنتن مدل نشر رادیویی, نوع پروتکل مسیر یابی ad-hoc مورد استفاده ی گره های متحرک و ... را نیز تعریف کنید.

برای توصیف کوتاهی در مورد هر یک از متغییرها ی تعریف شده به توضیح موجود در کد زیر توجه کنید. آرایه ی مورد استفاده برای تعریف این متغییر ها, val() است ولی عمومیت ندارد چرا که در اسکریپت های بیسیم قدیم نیز موجود بوده است.

اسکریپت simple-wireless.tcl خود را با فهرستی از این پارامترهای متفاوتی که در بالا توضیح داه شدند به ترتیب زیر شروع می کنیم.

```
#
# ======================================
# ==============================
# Define options
#
# ======================================
# ==============================
set val(chan)         Channel/WirelessChannel  ;# channel type
set val(prop)         Propagation/TwoRayGround ;# radio-
propagation model
set val(ant)          Antenna/OmniAntenna      ;# Antenna type
set val(ll)           LL                       ;# Link layer type
set val(ifq)          Queue/DropTail/PriQueue  ;# Interface queue
type
set val(ifqlen)       ۵۰                       ;# max packet in ifq
set val(netif)        Phy/WirelessPhy          ;# network interface type
set val(mac)          Mac/۸۰۲_۱۱               ;# MAC type
set val(rp)           DSDV                     ;# ad-hoc routing protocol
set val(nn)           ۲                        ;# number of mobilenodes
```

سپس به قسمت اصلی برنامه رفته و کار خود را با ایجاد نمونه ای از شبیه ساز شروع می کنیم.

```
set ns_    [new Simulator]
```

سپس با بازکردن فایل از ردیابی simple.tr ساختار پشتیبانی می کنیم و تابع trace-all را بترتیب زیر فرا می خوانیم.

```
set tracefd     [open simple.tr w]
$ns_ trace-all $tracefd
```

سپس یک شی توپولوژی بوجود می آوریم که حرکات گره های متحرک را در محدوده ی توپولوژیک ردیابی می کند.

```
set topo          [new Topography]
```

قبلا ذکر کردیم که گره های متحرک در یک توپولوژی با ابعاد ۵۰۰mX۵۰۰m حرکت می کنند, ما شی توپوگرافی را با مختصات x,y محدوده مقرر میکنیم.(x=۵۰۰, y=۵۰۰)

```
$topo load_flatgrid ۵۰۰ ۵۰۰
```

توپولوژی به شبکه هایی تقسیم می شود و مقدار پیش فرض تجزیه ی شبکه ۱ می باشد.یک مقدار متفاوت می تواند برای load_flatgrid {} بالا ,به عنوان پارامتر سوم بکار گرفته شود.

در مرحله ی بعد, باید شی God را بترتیب زیر ایجاد کنید:

```
create-god $val(nn)
```

God شئی است که برای ذخیره سازی اطلاعات عمومی در موقعیت شبکه و یا گره بکار گرفته می شود. امروزه شی God تعداد کامل گره های متحرک و جدولی از کمترین تعداد hop های مورد نیاز برای رسیدن به یک گره از گره دیگر را ذخیره می سازد.

در ابتدا باید قبل از ایجاد گره ها آنها را شکل بندی کنیم.

```
# $ns_ node-config -addressingType flat or hierarchical or expanded
#                  -adhocRouting   DSDV or DSR or TORA
#                  -llType     LL
#                  -macType          Mac/۸۰۲_۱۱
#                  -propType         "Propagation/TwoRayGround"
```

```
#             -ifqType          "Queue/DropTail/PriQueue"
#             -ifqLen    ۵۰
#             -phyType          "Phy/WirelessPhy"
#             -antType          "Antenna/OmniAntenna"
#             -channelType    "Channel/WirelessChannel"
#             -topoInstance   $topo
#             -energyModel    "EnergyModel"
#             -initialEnergy  (in Joules)
#             -rxPower        (in W)
#             -txPower        (in W)
#             -agentTrace     ON or OFF
#             -routerTrace    ON or OFF
#             -macTrace       ON or OFF
#             -movementTrace  ON or OFF
```

تمام مقادیر پیش فرض برای این گزینه ها NULL میباشند غیر از:

نوع آدرس دهی : flat

قصد داریم از مقدار پیش فرض flat addressing استفاده کنیم و تنها به AgentTrace , RouterTrace اجازه می دهیم فعال شوند. شما می توانید با فعال کردن تمامی ردیاب ها, آنها را امتحان کنید. عامل ردیابی(AgentTrace) با AGT, ردیابی مسیریاب با (RouterTrace) با RTR و MacTrace با MAC مشخص شده و نشان داده می شوند .زمانی که ردیاب حرکت (MovementTrace) آغازبه کار می کند,حرکات گره های متحرک را نشان میدهد.

API شکل بندی برای ایجاد گره های متحرک بصورت زیر بنظر می رسد:

```
# Configure nodes
        $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) \
                         -topoInstance $topo \
                         -channelType $val(chan) \
                         -agentTrace ON \
                         -routerTrace ON \
                         -macTrace OFF \
                         -movementTrace OFF
```

سپس به ترتیب زیر دو گره متحرک ایجاد می کنیم:

```
for {set i ۰} {$i < $val(nn) } {incr i} {
                set node_($i) [$ns_ node ]
                $node_($i) random-motion ۰       ;# disable random motion
```

حرکت تصادفی گره ها در اینجا غیرفعال شده است چرا که قصد داریم جایگاه گره و دستورالعمل های حرکتی (سرعت و مسیر) را فراهم آوریم. اکنون که گره های متحرک را ایجاد کرده ایم,باید جایگاهی را برای شروع در اختیارشان بگذاریم:

```
# Provide initial (X,Y, for now Z=۰) co-ordinates for node_(۰) and
node_(۱)
$node_(۰) set X_ ۵.۰
$node_(۰) set Y_ ۲.۰
$node_(۰) set Z_ ۰.۰
$node_(۱) set X_ ۳۹۰.۰
$node_(۱) set Y_ ۳۸۵.۰
$node_(۱) set Z_ ۰.۰
```

گره ۰ موقعیت حرکتی(۵,۲) را داراست در حالیکه گره ۱ در جایگاه (۳۹۰,۳۸۵) شروع به فعالیت می کند.

سپس تعدادی حرکت گره بوجود می آوریم:

```
#
# Node_(۱) starts to move towards node_(۰)
#
$ns_ at ۵۰.۰ "$node_(۱) setdest ۲۵.۰ ۲۰.۰ ۱۵.۰"
$ns_ at ۱۰.۰ "$node_(۰) setdest ۲۰.۰ ۱۸.۰ ۱.۰"

# Node_(۱) then starts to move away from node_(۰)
$ns_ at ۱۰۰.۰ "$node_(۱) setdest ۴۹۰.۰ ۴۸۰.۰ ۱۵.۰"
```

"$node_(۱) setdest ۲۵.۰ ۲۰.۰ ۱۵.۰" $ns_ at ۵۰.۰ یعنی در زمان ۵۰.۰s ثانیه ,گره ی ۱ با سرعتی معادل ۱۵متر در ثانیه شروع به حرکت به سوی مقصد(x=۲۵,y=۲۰) می کند. این API برای تغییر مسیر و سرعت حرکات گره های متحرک استفاده می شود.

ترافیک بترتیب زیر, بین دو گره جریان می یابد:

```
# TCP connections between node_(۰) and node_(۱)
set tcp [new Agent/TCP]
$tcp set class_ ۲
set sink [new Agent/TCPSink]
$ns_ attach-agent $node_(۰) $tcp
$ns_ attach-agent $node_(۱) $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at ۱۰.۰ "$ftp start"
```

این برنامه بین دو گره با یک منبع Tcp بر گره ۰, یک ارتباط Tcp برقرار میکند.سپس باید زمان توقف شبیه سازی را تعریف و مشخص کرده و به گره های متحرک می گوییم که دوباره تنظیم و راه اندازی شوند که در واقع مولفه های شبکه ای داخلی خود را مجدد تنظیم و راه اندازی می کنند:

```
#
# Tell nodes when the simulation ends
#
for {set i ۰} {$i < $val(nn) } {incr i} {
    $ns_ at ۱۵۰.۰ "$node_($i) reset";
}
```

```
$ns_ at ۱۵۰.۰۰۰۱ "stop"
$ns_ at ۱۵۰.۰۰۰۲ "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
    global ns_ tracefd
    close $tracefd
}
```

در زمان ۱۵۰ثانیه شبیه سازی باید متوقف شود. گره ها در همان زمان مجدد راه اندازی و تنظیم شوند و در ۱۵۰.۰۰۰۲ ثانیه ,کمی پس از تنظیم و راه اندازی مجدد گره ها "$ns_ halt" فراخوانده می شود. برای دور ریختن ردیابی ها و بستن فایل ردیابی, روند {}stop انجام می شود.

و بالاخره فرمان شروع شبیه سازی:

```
puts "Starting Simulation..."
$ns_ run
```

فایل simple-wireless.tcl را ذخیره کنید. در مرحله ی بعد شبیه سازی را به شیوه ی معمولی اجرا کنید. در انتهای اجرای شبیه سازی فایل trace-output file simple.tr بوجود می آید. وقتی که AgentTrace و RouterTrace را اجرا میکنیم, مشاهده می کنیم که پیام های مسیریابی DSDV و بسته های Tcp در حال ارسال و دریافت توسط اشیا عامل و مسیریاب در گره -۰- و -۱- هستند. مشاهده می کنید که جریان Tcp در ۱۰.۰ ثانیه از گره ۰ شروع می شود. در ابتدا هر دو گره از یکدیگر فاصله دارند و بنابراین بسته های Tcp توسط گره ۰ بیرون انداخته می شوند چرا که نمی تواند از گره یک مطلع باشد. حدود ۸۱.۰ ثانیه اطلاعات مسیریابی بین هر دو گره مبادله می شود و حدود ۱۰۰.۰ ثانیه مشاهده می کنید که اولین بسته ی Tcp توسط عامل (Agent), در گره یک دریافت شده و سپس یک Ack را به گره ۰ باز پس می فرستد و ارتباط Tcp برقرار می شود.با این حال, همانطور که گره ۱ شروع به فاصله گرفتن از گره۰ می کند, حدود زمان ۱۱۶.۰ ثانیه ارتباط مجدد قطع می شود. همانطور که گره ها از یکدیگر فاصله می گیرند,بسته ها نیز دور انداخته می شوند.

۶.۲ بکارگیری فایل های حرکت گره و الگوی ترافیکی و دیگر مشخصه ها در شبیه سازی بی سیم

به منظور گسترده کردن مفاهیم ذکر شده در قسمت قبل, قصد داریم که یک سناریوی بی سیم Multihop را که در اینجا از سه گره متحرک تشکیل شده است را شبیه سازی کنیم. مانند قبل ,گره های متحرک در محدوده ی مرزی یک توپولوژی مشخص و تعریف شده حرکت می کنند. برای این مثال حرکات گره ها باید از یک فایل حرکتی گره به نام scen-۳-test خوانده شوند. scen-۳-test حرکات اتفاقی گره را برای سه گره متحرک یک توپولوژی با گستره ی ۶۷۰mX۶۷۰m تعریف و مشخص می کند.این فایل به عنوان بخشی از ns موجود بوده و می توان آن را به همراه دیگر فایل های حرکتی گره زیر دایرکتوری ns/tcl/mobility/scene~ پیدا کرد. فایل های حرکت های اتفاقی گره مثل scen-۳-test را می توان با استفاده از مولد حرکت گره "setdest",CMU تولید کرد.

علاوه بر حرکت های گره , جریانات ترافیکی که بین گره های موبایل برقرار می شوند, از یک فایل الگوی رفت و آمد موسوم به cbr-۳-test خوانده می شوند. جریانات تصادفی TCP , CBR بین گره های سیار ۳ برقرار می شوند

باید در اسکریپت simple-wireless.tcl که ایجادش کردیم, تغییراتی را اعمال کرده و فایل های حاصل را wireless۱.tcl بنامیم. علاوه بر متغییر هایی که در اول اسکریپت عنوان کردیم (LL, MAC, antenna etc), اکنون چند پارامتر دیگر همچون الگوی اتصالی و فایل حرکت گره, مقادیر x وy برای محدوده ی مرزی توپولوژی ,مقدار seed برای مولد تعدادتصادفی, زمان توقف شبیه سازی و زمان استراحت را تعریف می کنیم. این پارامتر ها بترتیب زیر لیست می شوند:

```
set val(chan)       Channel/WirelessChannel
set val(prop)       Propagation/TwoRayGround
set val(netif)      Phy/WirelessPhy
set val(mac)        Mac/۸۰۲_۱۱
set val(ifq)        Queue/DropTail/PriQueue
```

```
set val(ll)         LL
set val(ant)        Antenna/OmniAntenna
set val(x)          ۶۷۰   ;# X dimension of the topography
set val(y)          ۶۷۰   ;# Y dimension of the topography
set val(ifqlen)     ۵۰          ;# max packet in ifq
set val(seed)       ۰.۰
set val(adhocRouting)   DSR
set val(nn)         ۳           ;# how many nodes are simulated
set val(cp)         "../mobility/scene/cbr-۳-test"
set val(sc)         "../mobility/scene/scen-۳-test"
set val(stop)       ۲۰۰۰.۰          ;# simulation time
```

تعداد گره های متحرک تا ۳ عدد تغییر می کند. بعلاوه بجای استفاده از DSDV از DSR به عنوان پروتکل مسیر یاب استفاده میکنیم. پس از ایجاد _ns مثال برای شبیه ساز, فایل (wireless۱-out.tr) را برای ردیابهای بی سیم باز می کنیم. همچنین ردیابهای nam را راه اندازی می کنیم:

```
set tracefd  [open wireless۱-out.tr w]      ;# for wireless traces
$ns_ trace-all $tracefd

set namtrace [open wireless۱-out.nam w]         ;# for nam tracing
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)
```

سپس بعد از ایجاد گره های متحرک , فایل های اصلی منبع حرکت گره و الگو های شبیه ساز که قبلا بترتیب به عنوان val(sc) و val(cp) تعریف شده بودند را راه اندازی می کنیم.

```
# Define node movement model
puts "Loading connection pattern..."
source $val(cp)
# Define traffic model
puts "Loading scenario file..."
source $val(sc)
```

در فایل scen-۳-test حرکت گره , فرمان حرکت گره را میبینیم, مثل:

```
$ns_ at ۵۰.۰۰۰۰۰۰۰۰۰۰۰۰ "$node_(۲) setdest ۳۶۹.۴۶۳۲۴۴۹۱۵۷۴۳ \
۱۷۰.۵۱۹۲۰۳۱۱۱۱۵۲ ۳.۳۷۱۷۸۵۸۹۹۱۵۴"
```

همانطور که در قسمت قبل گفتیم ,این فرمان بدان معناست که در زمان ۵۰ ثانیه ,گره ۲ شروع می کند به حرکت به سمت مقصد (۳۶۸.۴,۱۷۰.۵) با سرعتی معادل ۳/۳۷ متر در ثانیه , همچنین خط های دیگری را می بینیم همچون:

```
$god_ set-dist ۱ ۲ ۲
```

این خطوط, خطوط دستوری می باشند که به منظور پرکردن شی god با اطلاعات کوتاهترین hop بکار می روند. این مطلب بدان معناست که کوتاهترین مسیر بین گره ۱ و ۲ , دو hop است. با گردآوری و فراهم ساختن این اطلاعات از محاسبه ی کوتاهترین فاصله ی بین گره ها توسط شی god در طی انجام عمل شبیه سازی که ممکن است خیلی وقتگیر باشد, ممانعت بعمل می آید.

برنامه ی setdest با استفاده از الگوریتم نقطه یابی تصادفی مسیر فایل های الگوی حرکت را تولید می کند. فایل های حرکت گره تولید شده با استفاده از setdest (مثل scen-۳-test) در حال حاضر خطوطی

همچون خطوط دستوری بالا را شامل می شوند تا در زمان مناسب, اطلاعات مناسب را در اختیار شی god قرار دهند.

در مرحله ی بعد برای آماده کردن جایگاه اولیه گره ها در nam, خط های دستوری زیر را اضافه می کنیم.در هر حال توجه داشته باشید که در حال حاضر می توان تنها حرکات گره را در nam مشاهده کرد.تخلیه ی اطلاعات ترافیکی و بنابراین تجسم حرکات بسته های اطلاعاتی در nam برای سناریوی بیسیم مورد تایید قرار نگرفته است.

```
# Define node initial position in nam
for {set i ۰} {$i < $val(nn)} {incr i} {

    # ۲۰ defines the node size in nam, must adjust it according to
your
    # scenario size.
    # The function must be called after mobility model is defined
    $ns_ initial_node_pos $node_($i) ۲۰
}
```

سپس هدرهای اطلاعاتی را برای فایل CMUTrace درست قبل از خط "ns_ run" به دستور اضافه می کنیم.

```
puts $tracefd "M ۰.۰ nn $val(nn) x $val(x) y $val(y) rp
$val(adhocRouting)"
puts $tracefd "M ۰.۰ sc $val(sc) cp $val(cp) seed $val(seed)"
puts $tracefd "M ۰.۰ prop $val(prop) ant $val(ant)"
```

بقیه ی اسکریپت بدون تغییر باقی می مانند. فایل wireless۱.tclرا ذخیره کنید. از وجود فایل های حرکت گره و الگوی اتصال در زیر دایرکتوری ها اطمینان حاصل کنید.

با تایپ کردن در اعلان, اسکریپت را اجرا کنید.

```
ns  wireless۱.tcl
```

در تکمیل سازی اجرا , فایل خروجی CMUTrace,"wireless۱-out.tr" و فایل خروجی nam, "wireless۱-out.nam" ایجاد می شوند. با اجرای wireless۱-out.nam مشاهده کردیم که سه گره متحرک در پنجره ی nam حرکت می کنند.همانطور که پیش از این گفتیم نمی توان هیچگونه جریان ترافیکی را مشاهده کرد.

برای گوناگونی خروجی های AgentTrace, RouteTrace, MacTrace و movementTrace را مانند آنچه که پیش از این در اسکریپت نشان داده شده و فعال و یا غیر فعال سازید. گره ۱ در گستره و محدوده ی گره ۰ قرار داشته و گره ۲ می تواند با هر دوی آنها ارتباط برقرار کند. بنابراین تمامی بسته هایی که مقصدشان گره ۰ و ۲ است, مسیرشان از میان گره ۱ است(از میان گره ۱ عبور میکنند).

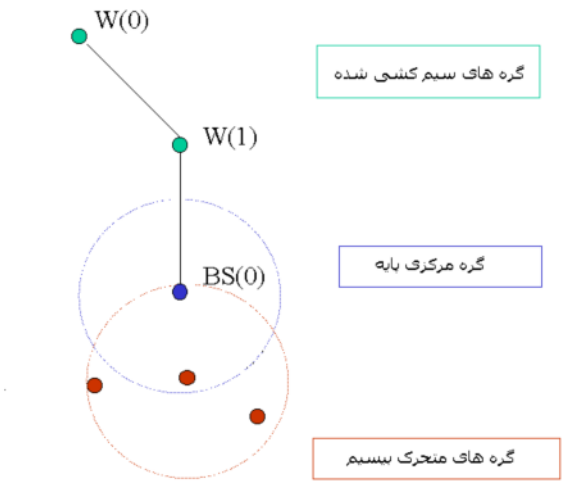
۷ شبیه سازی های شبکه های **CUM** بیسیم وسیم کشی شده و **MobileIP** در **ns**

۷.۱ ایجاد یک سناریوی ساده ی *wired-cum-wireless*

شبیه سازی بیسیم که در فصل قبل توضیح داده شد, برای شبکه های multi-hop ad-hoc یا LANهای بیسیم می باشد. ولی ممکن است مجبور باشید یک توپولوژی متشکل از چند LAN که از طریق گره های سیم دار به یکدیگر متصل می باشند را شبیه سازی کنید یا به عبارت دیگر باید یک توپولوژی **wired-cum-wireless** ایجاد کنید.

در این بخش قصد داریم توپوولژی بیسیم ساده ای را که در بخش قبل ایجاد شد را گسترس داده ویک سناریوی مختلط بوجود آوریم که از یک حوزه بیسیم و یک حوزه ی سیم کشی شده تشکیل شده و در آنجا اطلاعات بین گره های متحرک و ثابت مبادله می شوند.در اینجا قصد بر اسکریپت wireless۱.tcl که در قسمت قبل ایجاد شده ,اطلاعاتی را اعمال کرده و فایل سناریوی **wired-cum-wireless** بدست آمده را به نام wireless۲.tcl نامگذاری کنیم.

قصد داریم که برای سناریوی مختلط , دو گره سیم کشی شده W(۱),W(۰) داشته باشیم که به حوزه ی بیسیم ما که متشکل است از سه گره متحرک (۱, ۲ و ۳), ازطریق یک گره (مرکز پایه) BS متصل است. شکل ۱ توپولوژی مثالی که در بالا بیان شد را نشان می دهد.



۱-۷ شبکه مثال wired-cum-wireless

در ابتدا کار خود را با بررسی اینکه چه تغییراتی باید بر لیست متغییر هایی که در ابتدای wireless۱.tcl تعریف و مشخص شده اند آغاز می کنیم.

پروتکل مسیریاب Adhoc تغییر کرده و به DSDV تبدیل می شود.به علاوه ما در خود اسکریپت اتصالات CBR, TCP را بین گره های سیم کشی شده و بیسیم تعریف می کنیم. بنابراین دیگر به فایل الگوی اتصال که در شبیه سازی پیشین مورد استفاده قرار گرفت, نیاز نداریم. همچنین زمان توقف شبیه سازی را نیز تغییر می دهیم.

دراینجا توجه داشته باشید که بجای آرایه ی Val() ,آرایه ی Opt() را استفاده می کنیم تا نشان دهیم که Val() دیگر یک مقدار عمومی آرایه نبوده و حوزه ی آن تنها در اسکریپت آزمایشی تعریف می شود.

```
set opt(adhocRouting)   DSDV
set opt(cp)     ""      ;# cp file not used
set opt(stop)           ۳۰۰     ;# time to stop simulation
```

در اینجا زمانهای شروع برای جریانات Tcp را تعریف می کنیم.

set opt(ftp۱-start) ۱۶۰.۰

set opt(ftp۲-start) ۱۷۰.۰

و همچنین خط دستوری زیر را به منظور تعریف تعدادی از گره های سیمی و گره های دارای مرکز پایه اضافه کنید.

set num_wired_nodes ۲

set num_bs_nodes ۱

اکنون به سراغ قسمت اصلی برنامه می رویم. برای شبیه سازی های مختلط باید از مسیریابی سلسله مراتبی استفاده کنیم تا بسته ها را بین حوزه های بیسیم و سیم کشی شده مسیریابی کنیم. در ns اطلاعات مسیریابی برای گره های سیم کشی شده بر پایه ی قدرت اتصال توپولوژی یا به عبارتی به چگونگی اتصال گره ها به یکدیگر از طریق لینک ها قرار دارد. این اطلاعات اتصال برای مقداردهی جداول ارسال در هر گره سیم کشی شده بکار می روند. با این وجود گره های بیسیم هیچ مفهوم و تصویری از لینک ندارند. در یک توپولوژی بیسیم, بسته ها با استفاده از پروتکل های adhoc مسیریاب که با مبادله پرسجوهای مسیر یابی خود در بین همسایگانش جداول ارسال را می سازد, مسیر یابی می شوند. بنابراین به منظور مبادله بسته ها بین این گره های سیم کشی شده و بیسیم,از گره های دارای مرکز پایه استفاده می کنیم که نقش در ورودی و خروجی بین دو حوزه را ایفا می کنند. با قرار دادن گره های بیسیم و سیم کشی شده در حوزه های متفاوت ,آن دو را از هم تفکیک می کنیم. حوزه ها و حوزه های فرعی (یا کلاستر نامی که در اینجا به آنها اطلاق شده است) توسط ساختار توپولوژی طبقاتی تعریف می شوند, مانند آنچه که در زیر نشان داده شده است.بعد از خط "set ns [new Simulator]" , خطوط زیر را اضافه کنید.

```
$ns_ node-config -addressType hierarchical
AddrParams set domain_num_ ۲          ;# number of domains
lappend cluster_num ۲ ۱               ;# number of clusters in each
                                      ;#domain
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel ۱ ۱ ۴             ;# number of nodes in each
cluster
AddrParams set nodes_num_ $eilastlevel ;# for each domain
```

در خطوط دستوری بالا در ابتدا شی گره را شکل بندی کردیم تا گونه ای آدرس دهی سلسله مراتبی داشته باشیم. پس از این سلسله مراتب و درجات توپولوژی تعریف می شود. تعداد حوزه ها در این توپولوژی ۲ حوزه است (یک حوزه برای گره های سیم کشی شده و یک حوزه برای گره های بیسیم) تعداد کلاسترها در هر یک از این حوزه ها بصورت "۱ ۲" تعریف شده که نشان می دهدحوزه ی اول سیم کشی شده دارای ۲ کلاستر و حوزه ی دوم (بیسیم) دارای ۱کلاستر می باشند. خط بعدی تعداد گره ها در هر یک ازاین کلاستر ها را مشخص میکند که بدین ترتیب است "۴ ۱ ۱" به عبارت دیگر یک گره در هر یک از دو کلاستر اول (در حوزه ی سیم کشی شده) و ۴ گره در کلاستر حوزه ی بیسیم. بنابراین توپولوژی در یک سلسله مراتب ۳ طبقه ای تعریف می شود(شکل توپولوژی را در بالا مشاهده کنید.)

پس از این کار, ردیابی را برای شبیه سازی راه اندازی می کنیم. در اینجا توجه داشته باشید که برای شبیه سازی cum سیم کشی شده و بیسیم, ممکن است ردیابی ها برای هر دو حوزه بیسیم وسیعی تولید شود. هر دوی این ردیابی ها در یک فایل خروجی نوشته می شوند که در اینجا به صورت wireless۲-out.tr تعریف می شود.

به منظور تمایز قائل شدن بین ردیابی های سیم کشی شده و بیسیم, تمامی ردیابی های بی سیم با "WL" شروع می شوند. به علاوه ردیابی های nam را نیز راه اندازی می کنیم. همانطور که قبلا بیان شد ردیابی های nam برای گره های بیسیم در حال حاضر تنها حرکات گره را نشان می دهند. در مرحله ی بعد, باید گره های جایگاه پایه گره های سیم کشی شده و بیسیم را بوجود آوریم. در اینجا توجه داشته باشیدکه برای ایجاد هر گره ای باید آدرس دهی سلسله مراتبی گره مشخص شود. بنابراین پس از خط "create-god $opt(nn)" خطوط زیر را برای ایجاد گره های سیم کشی شده اضافه کنید.

```
# create wired nodes
set temp {۰.۰.۰ ۰.۱.۰}          ;# hierarchical addresses to be used
for {set i ۰} {$i < $num_wired_nodes} {incr i} {
    set W($i) [$ns_ node [lindex $temp $i]]
}
```

به منظور ایجاد گره های پایه باید مانند آنچه که در زیر نشان داده شده است, ساختار گره را شکل دهیم. این ساختار,قسمتی از API گره ی جدید می باشدکه ابتدا از شکل بندی و سپس از ایجاد گره ها تشکیل شده است. از آنجائیکه گره های جایگاه پایه در ورود و خروج بین حوزه های بیسیم و سیم کشی شده می باشند, باید مکانیسم مسیریابی سیم کشی شده خود را فعال سازندکه این امر با تنظیم گزینه ی مسیر یابی سیم کشی شده در شکل بندی گره روی ON صورت می پذیرد. پس از ایجاد گره پایه گره بیسیم را مجددا ترکیب بندی می کنیم و بنابراین مسیر یابی سیم کشی شده را غیر فعال می کنیم. تمام دیگر گزینه های ترکیب بندی گره مورد استفاده ی جایگاه پایه ,برای گره های متحرک بدون تغییر و یکسان باقی می مانند.به علاوه گره BS(۰) بعنوان گره پایه برای تمامی گره های متحرک در حوزه ی بیسیم تعیین شده است, بدین منظور که تمامی بسته هایی که از گره متحرک نشأت گرفته و ارسال شده اند و مقصدشان بیرون از حوزه ی بیسیم می باشد توسط گرههای متحرک به جایگاه پایه ارسال می شوند. توجه داشته باشید که قرار داشتن گره

جایگاه پایه درحوزه ی یکسانی با حوزه ی گره های بیسیم از اهمیت خاصی برخوردار است. این امر بدان منظور است که تمامی بسته هایی که از حوزه ی سیم کشی شده ارسال شده و مقصدشان یک گره بیسیم است, به گره پایه می رسند که سپس گره پایه از پروتکل مسیر یاب adhoc خود برای مسیریابی بسته به مقصد صحیح خود استفاده می کند. بنابراین در یک شبیه سازی مختلط که گره های سیم کشی شده و بیسیم را در بر می گیرد لازم است که:

۱. برای فعال ساختن مسیریابی سلسله مراتبی
۲. برای ایجاد کردن حوزه هایی متفاوت برای گره های سیم کشی شده و بی سیم, ممکن است چند حوزه ی سیم کشی شده و بیسیم برای شبیه سازی پندین شبکه وجود داشته باشد.
۳. برای داشتن یک گره جایگاه پایه در هرحوزه ی بیسیم,که از طریق آن ممکن است گره های بیسیم با گره های خارج از حوزه ی خود ارتباط برقرار کنند.

برای اینکه ببینیم چگونه سلسه مراتب بوجود می آید ,مرحله به مرحله با مثال پیش می رویم. در اینجا دو حوزه داریم: حوزه ی ۰ برای سیم کشی شده و حوزه ی ۱ برای بیسیم. دو گره سیم کشی شده در دو کلاستر جدا قرار داده می شوند. کلاستر ۰ و ۱.بنابراین آدرس دهی آنها بدین صورت بنظر می رسد: ۰(حوزه ی ۰).۰(کلاستر ۰).۰(فقط گره)و ۰(همان حوزه ی ۰).۱(کلاستر ۱).۰(دوباره تنها,گره).

برای گره های بیسیم که در حوزه ی ۱ هستند,یک کلاستر(۰) را تعریف کرده ایم, بدین ترتیب همه گره ها در این کلاستر می باشند.از این جهت آدرس دهی ها به ترتیب زیر می باشند:

جایگاه پایه :۱ (حوزه ی دوم ,۱).۰(کلاستر ۰).۰(گره اول در کلاستر)

گره WL□۱ (۱ # WL node): ۱.۰.۱ (گره دوم در کلاستر)

گره WL□۲ :۱.۰.۲ (گره سوم)

گره WL□۳ : ۱.۰.۳(گره چهارم)

ما می توانستیم دو گره سیم کشی شده را در کلاستر یکسانی در حوزه ی سیم کشی شده ی ۰ قرار دهیم. همچنین می توانستیم دیگر گره های بیسیم در کلاستر های متفاوت را در حوزه ی بیسیم قرار دهیم. به علاوه بسته به توپولوژی مان ,ممکن است از تمامی کلاستر ها خلاص شویم و تنها دو سطح سلسله مراتب داشته باشید: گره ها و حوزه ها

```
# configure for base-station node
$ns_ node-config -adhocRouting $opt(adhocRouting) \
                 -llType $opt(ll) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propType $opt(prop) \
                 -phyType $opt(netif) \
                 -channelType $opt(chan) \
                     -topoInstance $topo \
                 -wiredRouting ON \
                     -agentTrace ON \
                 -routerTrace OFF \
                 -macTrace OFF

#create base-station node
set temp {۱.۰.۰ ۱.۰.۱ ۱.۰.۲ ۱.۰.۳}   ;# hier address to be used for
                                        ;# wireless domain
```

```
set BS(٠) [ $ns_ node [lindex $temp ٠]]
$BS(٠) random-motion ٠              ;# disable random motion

#provide some co-ordinates (fixed) to base station node
$BS(٠) set X_ ١٠٠
$BS(٠) set Y_ ٢٠٠
$BS(٠) set Z_ ٠٠٠

# create mobilenodes in the same domain as BS(٠)
# note the position and movement of mobilenodes is as defined
# in $opt(sc)
# Note there has been a change of the earlier AddrParams
# function 'set-hieraddr' to 'addr٢id'.

#configure for mobilenodes
$ns_ node-config -wiredRouting OFF

# now create mobilenodes
for {set j ٠} {$j < $opt(nn)} {incr j} {
    set node_($j) [ $ns_ node [lindex $temp \
            [expr $j+١]] ]
    $node_($j) base-station [AddrParams addr٢id \
            [$BS(٠) node-addr]]   ;# provide each mobilenode with
                                  ;# hier address of its base-station
```

}

سپس BS و گره های سیم کشی شده ا متصل کرده و ترافیک TCP را بین گره بیسیم (گره۰-(۰)-node-), و گره سیم کشی شده ((۰)w) وبین (۱)w و (۲)-node برقرار می کنیم مانند آنچه که در زیر نشان داده شده است.

```
#create links between wired and BS nodes
$ns_ duplex-link $W(۰) $W(۱) ۵Mb ۲ms DropTail
$ns_ duplex-link $W(۱) $BS(۰) ۵Mb ۲ms DropTail

$ns_ duplex-link-op $W(۰) $W(۱) orient down
$ns_ duplex-link-op $W(۱) $BS(۰) orient left-down

# setup TCP connections
set tcp۱ [new Agent/TCP]
$tcp۱ set class_ ۲
set sink۱ [new Agent/TCPSink]
$ns_ attach-agent $node_(۰) $tcp۱
$ns_ attach-agent $W(۰) $sink۱
$ns_ connect $tcp۱ $sink۱
set ftp۱ [new Application/FTP]
$ftp۱ attach-agent $tcp۱
```

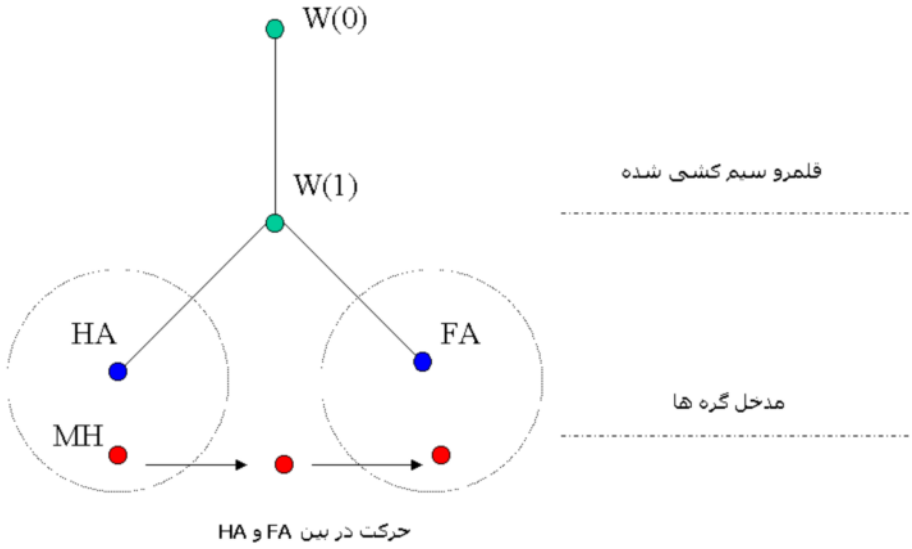
```
$ns_ at $opt(ftp۱-start) "$ftp۱ start"
set tcp۲ [new Agent/TCP]
$tcp۲ set class_ ۲
set sink۲ [new Agent/TCPSink]
$ns_ attach-agent $W(۱) $tcp۲
$ns_ attach-agent $node_(۲) $sink۲
$ns_ connect $tcp۲ $sink۲
set ftp۲ [new Application/FTP]
$ftp۲ attach-agent $tcp۲
$ns_ at $opt(ftp۲-start) "$ftp۲ start"
```

فایل های ردیابی ns, nam در انتهای اجرای شبیه سازی تولید می شوند. اجرای wireless۲- out.nam, حرکات گره های متحرک و ترافیک را در حوزه ی سیم کشی شده نشان می دهد. همانطور که قبلا گفتیم جریان ترافیک برای گره های متحرک هنوز در nam مورد تایید قرار نگرفته است.در فایل ردیابی wireless۲-out.tr ردیابی ها به هر دو حوزه یسیم کشی شده و بیسیم را مشاهده می کنیم (که بیسیم با WL شروع شده است). در ۱۶۰.۰ ثانیه یک اتصال TCL بین -۳- ,(که گره ۰ می باشد) و ۰(که W(۰) می باشد) برقرارمی گردد. توجه داشته باشید که شناسه های گره در داخل توسط شبیه ساز بوجود آمده و در ترتیب ایجاد گره قرار داده می شوند. در ۱۷۰ ثانیه یک اتصال TCP دیگر در جهت مخالف از حوزه ی سیم کشی شده به بیسیم برقرار می شود.

۷.۲ اجرای ***IP*** متحرک در یک توپولوژی *wired-cum-wireless* ساده

تاکنون یک توپولوژی wired-cum-wireless ایجاد کرده و بسته ها را از طریق یک گره پایه بین حوزه ی سیم کشی شده و بیسیم مبادله کرده ایم. ولی یک گره متحرک ممکن است خارج از محدوده ی گره پایه ی خود قرار گرفته باشد و هنوز باید به دریافت بسته هایی که مقصدشان آن گره می باشد, ادامه دهد.

برای این مثال با همان حوزه ی سیم کشی شده که متشکل است از دو گره سیم کشی شده W۱,W۰ را داریم. ما دو گره جایگاه پایه داریم که آنها را بترتیب عامل خانگی یا HomeAgent(HA) و عامل خارجی ForeignAgent(FA) می نامیم. همانطور که در شکل زیر نشان داده شده است گره سیم کشی شده ی W۱ به HA و FA متصل می شود. یک گره متحرک بنام MobileHost(MH) وجود دارد که بین عامل خانگی و عامل خارجی خود حرکت می کند. ما یک جریان TCP بین W۰ وMH برقرار می کنیم. همانطور که MH از حوزه ی HA خود به حوزه ی FA حرکت می کند, مشاهده می کنید که چگونه طبق تعریف های پروتکل IP متحرک بسته هایی که مقصدشان MH بوده توسط HA آن بسته به FA مجددا هدایت می شوند. برای توپولوژی شرح داده شده در بالا, شکل زیررا ببینید.



MobileIP مثال ۲-۷

باید به منظور ایجاد اسکریپت wireless-mip که wireless۳.tcl نام دارد، wireless۲.tcl ایجاد شده در بخش قبلی را ویرایش کنید. ممکن است که کل wireless۳.tcl در اینجا مورد بررسی قرار نگیرد.

تعداد گره های متحرک و زمان شبیه سازی را تغییر دهید.

```
set opt(nn)          ۱          ;# just one MH
set opt(stop)        ۲۵۰
```

در این مثال اتصال TCP را برقرار کرده و حرکت MH در خود اسکریپت را اجرا می کنیم. از این جهت قصد استفاده از فایل های CP و SC را نداریم.

```
set opt(cp)          ""
```

```
set opt(sc)             ""
```

زمان شروع جریان TCP راتعریف می کنیم:

```
set opt(ftp۱-start)     ۱۰۰.۰
```

تعداد گره های سیم کشی شده بیسیم و جایگاه پایه را تغییر می دهیم. در هر حال توجه داشته باشید که متغییر num_bs_nodes در واقع در این اسکریپت بکار برده نمی شود. گره های جایگاه پایه FA و HA بطور جداگانه و تنهایی ایجاد و مدیریت می شوند.

```
set num_wired_nodes     ۲
```

پس از دو خطی که مثال ns را ایجاد کرده و فرمت آدرس دهی را بطور سلسه مراتبی تنظیم می کنند, به منظور تعریف سلسله مراتب توپولوژیک خطوط پایین را اضافه کنید. این کار کاملا شبیه wireless۲.tcl است با این تفاوت که اکنون ما یک حوزه ی سوم برای FA داریم.کلاستر راتغییر داده و طبق آن پارامتر های گره را تغییر دهید.

```
AddrParams set domain_num_ ۳          ;# number of domains
lappend cluster_num ۲ ۱ ۱             ;# number of clusters in each
domain
AddrParams set cluster_num_ $cluster_num
```

```
lappend eilastlevel ۱ ۱ ۲ ۱                ;# number of nodes in each
cluster
AddrParams set nodes_num_ $eilastlevel ;# of each domain
```

سپس ردیابی ns و nam را برای wireless-mip ایجاد می کنیم.

```
set tracefd  [open wireless۳-out.tr w]
set namtrace [open wireless۳-out.nam w]
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)
```

بنابراین در این توپولوژی یک حوزه ی سیم کشی شده داریم (که با ۰ نشان داده می شود)و دو حوزه ی بیسیم داریم (که به ترتیب با ۱ و ۲ نشان داده می شوند) از این جهت همانطور که در بخش قبلی توضیح دادیم, آدرس دهی های گره سیم کشی شده یکسان و بدون تغییر باقی می مانند (۰.۰.۰ , ۰.۱.۰). در اولین حوزه ی بیسیم در یک کلاستر جایگاه پایه HA و گره متحرک MH داریم. آدرس دهی های آنها به ترتیب ۱.۰.۰ , ۱.۰.۱ می باشد. برای حوزه یبیسیم دوم یک جایگاه پایه , FA با آدرس ۲.۰.۰ داریم.با این وجود در جریان شبیه سازی MH به درون حوزه ی FAحرکت می کند و ما باید ببینیم که چگونه بسته هایی که از یک حوزه ی سیم کشی شده ارسال شده و مقصدشان MH است در نتیجه ی پروتکل IP متحرک به MH می رسند.

گره های سیم کشی شده زودتر ایجاد می شوند.در مکان یک گره جایگاه پایه ی تنها, یک HA و FA ایجاد می شوند. در اینجا توجه دلشته باشید که Flag, IP متحرک را فعال کنید, بر طبق آن ما با استفاده از گزینه ی ON در IP متحرک , ساختار گره را ترکیب بندی می کنیم.

```
# Configure for ForeignAgent and HomeAgent nodes
```

```
$ns_ node-config -mobileIP ON \
                -adhocRouting $opt(adhocRouting) \
                -llType $opt(ll) \
                -macType $opt(mac) \
                -ifqType $opt(ifq) \
                -ifqLen $opt(ifqlen) \
                -antType $opt(ant) \
                -propType $opt(prop) \
                -phyType $opt(netif) \
                -channelType $opt(chan) \
                      -topoInstance $topo \
                -wiredRouting ON \
                      -agentTrace ON \
                -routerTrace OFF \
                -macTrace OFF


# Create HA and FA
set HA [$ns_ node ١.٠.٠]
set FA [$ns_ node ٢.٠.٠]
$HA random-motion ٠
```

$FA random-motion ۰

#provide some co-ord (fixed) to these base-station nodes.

$HA set X_ ۱.۰۰۰۰۰۰۰۰۰۰۰۰

$HA set Y_ ۲.۰۰۰۰۰۰۰۰۰۰۰۰

$HA set Z_ ۰.۰۰۰۰۰۰۰۰۰۰۰۰

$FA set X_ ۶۵۰.۰۰۰۰۰۰۰۰۰۰۰۰

$FA set Y_ ۶۰۰.۰۰۰۰۰۰۰۰۰۰۰۰

$FA set Z_ ۰.۰۰۰۰۰۰۰۰۰۰۰۰

سپس همانطور که پیش می رویم میزبان متحرک (mobilehost) را ایجاد می کنیم. توجه کنید که درست مانند قبل باید گزینه ی مسیر یابی سیم کشی شده که برای ایجاد گره های جایگاه پایه بکار می روند را قبل از ایجاد گره های متحرک ,غیر فعال کنید. همچنین HA بعنوان عامل خانگی (home-agent) برای میزبان متحرک برقرار می شود.

configure for mobilehost
$ns_ node-config -wiredRouting OFF

```
# create mobilehost that would be moving between HA and FA.
# note address of MH indicates its in the same domain as HA.
set MH [$ns_ node ۱.۰.۱]
set node_(۰) $MH
set HAaddress [AddrParams set-hieraddr [$HA node-addr]]
[$MH set regagent_] set home_agent_ $HAaddress

# movement of the MH
$MH set Z_ ۰.۰۰۰۰۰۰۰۰۰۰۰۰
$MH set Y_ ۲.۰۰۰۰۰۰۰۰۰۰۰۰
$MH set X_ ۲.۰۰۰۰۰۰۰۰۰۰۰۰

# MH starts to move towards FA (۶۴۰, ۶۱۰) at a speed of ۲۰m/s
$ns_ at ۱۰۰.۰۰۰۰۰۰۰۰۰۰۰۰ "$MH setdest ۶۴۰.۰۰۰۰۰۰۰۰۰۰۰۰
۶۱۰.۰۰۰۰۰۰۰۰۰۰۰۰
۲۰.۰۰۰۰۰۰۰۰۰۰۰۰"

# and goes back to HA (۲, ۲) at a speed of ۲۰ m/s
$ns_ at ۲۰۰.۰۰۰۰۰۰۰۰۰۰۰۰ "$MH setdest ۲.۰۰۰۰۰۰۰۰۰۰۰۰ ۲.۰۰۰۰۰۰۰۰۰۰۰۰
۲۰.۰۰۰۰۰۰۰۰۰۰۰۰"
```

بین گره های سیم کشی شده و HA/FA لینک ایجاد کرده و اتصال TCP ایجاد کنید:

```
# create links between wired and BaseStation nodes
$ns_ duplex-link $W(٠) $W(١) ٥Mb ٢ms DropTail
$ns_ duplex-link $W(١) $HA ٥Mb ٢ms DropTail
$ns_ duplex-link $W(١) $FA ٥Mb ٢ms DropTail

$ns_ duplex-link-op $W(٠) $W(١) orient down
$ns_ duplex-link-op $W(١) $HA orient left-down
$ns_ duplex-link-op $W(١) $FA orient right-down

# setup TCP connections between a wired node and the MobileHost

set tcp١ [new Agent/TCP]
$tcp١ set class_ ٢
set sink١ [new Agent/TCPSink]
$ns_ attach-agent $W(٠) $tcp١
$ns_ attach-agent $MH $sink١
$ns_ connect $tcp١ $sink١
set ftp١ [new Application/FTP]
$ftp١ attach-agent $tcp١
$ns_ at $opt(ftp١-start) "$ftp١ start"
```

بقیه ی اسکریپت بدون تغییر باقی می ماند.(به عبارت دیگر به گره های متحرک می گوید که چه زمانی شبیه سازی متوقف می شود).اسکریپت را ذخیره و اجرا کنید.

در حالی که اسکریپت را اجرا میکنید ممکن است با هشدار هایی مواجه شوید چون:

"warning: Route to base_stn not known: dropping pkt". این هشدار به این معناست که همینطور که MH از حوزه ی یک جایگاه پایه به حوزه ی یک جایگاه پایه ی دیگر حرکت می کند ممکن است مرحله ای موقتی وجود داشته باشد که در هیچ جایگاه پایه ای ثبت نشده است و بنابراین نمی داند که بسته هایی را که مقصدشان خارج از حوزه اش می باشد را به کسی ارسال کند.در هنگام اتمام اجرا فایل های خروجی ردیابی nam و"wireless۳-out.tr" و "wireless۳-out.nam" ایجاد می شوند. خروجی nam حرکت میزبان متحرک و جریان ترافیکی در حوزه ی سیم کشی شده را نشان میدهد. خروجی ردیابی ns ردیابی های گره های سیم کشی شده و حوزه ی بی سیم را نشان می دهد. در ابتدا بسته های TCP مستقیما توسط HAی MH , به MH تحویل داده می شوند.همینطور که MH از حوزه یha فاصله گرفته و به حوزه ی FA می رود , بسته هایی به مقصد MH را می یابیم که خلاصه شده اند. به FA ارسال می شوند که سپس FA قطعه قطعه شده و یا بسته را به حالت عادی برگردانده و آن را به MH میدهد.

۸ ایجاد فایل های حرکت گره و اتصال ترافیک برای سناریوی بیسیم بزرگ

ما از فایل های الگوی ترافیک و حرکت گره موجود با توزیع ns برای مثال های شبیه سازی در بخشهای قبلی استفاده کرده ایم. در این بخش به چگونگی استفاده از ترافیک و سناریوی CMU که اسکریپت های ایجاد این فایل ها را تولید می کند, می پردازیم. این بخش به دو قسمت تقسیم شده است. قسمت اول در مورد الگوی ترافیک تولید اسکریپت, چگونگی کارکردن آن و جریان های تصادفی TCP و یا CBR ای تولید می کند, صحبت می کنیم. قسمت دوم در مورد تولیدکننده حرکت گره CMU "setdest " صحبت می کند که می تواند برای ایجاد و حرکات اتفاقی گره نقطه مسیر برای گره های متحرک مورد استفاده قرار گیرد.

۸.۱ ایجاد الگوی تصادفی ترافیک برای سناریوی بی سیم

اتصالات ترافیکی تصادفی TCPو CBR می توانند با استفاده از یک اسکریپت تولید کننده سناریوی ترافیک بین گره های متحرک ایجاد شوند. این اسکریپت تولید کننده ترافیک تحت ~ns/indep-utils/cmu-scen-gen موجود است و cbrgen.tcl خوانده می شود و می توان از آن برای ایجاد اتصالات ترافیک TCP و cbr بین گره های متحرک بیسیم استفاده کرد. به منظور ایجاد یک فایل ترافیک باید نوع اتصال ترافیک (TCPیا CBR) تعداد گره ها و حداکثر تعداد اتصالی که قرار است بین آنها ایجاد شوند, را تعریف کرد بنابراین خط دستور اینگونه بنظر می رسد:

```
ns cbrgen.tcl [-type cbr|tcp] [-nn nodes] [-seed seed] [-mc connections]
[-rate rate]
```

زمان های شروع برای اتصالات CBR/TCP بصورت تصادفی با حداکثر مقدار تنظیم شده بر روی ۱۸۰.۰ ثانیه تولید می شوند. بطور مثال سعی می کنیم که با حداکثر ۸ اتصال , با مقدار seed معادل ۰.۱ و آهنگی معادل ۴.۰ فایل اتصال CBR بین ۱۰ گره ایجاد کنیم. بنابراین در Prompt اینگونه تایپ می کنیم :

```
ns cbrgen.tcl -type cbr -nn ۱۰ -seed ۱.۰ -mc ۸ -rate ۴.۰ > cbr-۱۰-test
```

از cbr-۱۰-test (که خروجی بر روی آن هدایت می شود) فایل ایجاد می شود. یکی از اتصالات CBR اینگونه به نظر می رسد:

```
#
# ۲ connecting to ۳ at time ۸۲.۵۵۷۰۲۳۷۴۶۲۲۰۸۶۴
#
set udp_(۰) [new Agent/UDP]
```

```
$ns_ attach-agent $node_(۲) $udp_(۰)
set null_(۰) [new Agent/Null]
$ns_ attach-agent $node_(۳) $null_(۰)
set cbr_(۰) [new Application/Traffic/CBR]
$cbr_(۰) set packetSize_ ۵۱۲
$cbr_(۰) set interval_ ۰.۲۵
$cbr_(۰) set random_ ۱
$cbr_(۰) set maxpkts_ ۱۰۰۰۰
$cbr_(۰) attach-agent $udp_(۰)
$ns_ connect $udp_(۰) $null_(۰)
$ns_ at ۸۲.۵۵۷۰۲۳۷۴۶۲۲۰۸۶۴ "$cbr_(۰) start"
```

بنابراین یک اتصال UDP بین گره ۲ و ۳ ایجاد می شود.منابع کل UDP که از گره ۰-۱۰ انتخاب شده اند و تعداد کل اجرای اتصالات در آخر فایل cbr-۱۰-test به ترتیب ۵ و ۸ بودند.

به همین ترتیب فاتیل های اتصالی TCP می توانند با استفاده از "Type" به عنوان TCP ایجاد شوند.مثالی در این زمینه:

```
ns cbrgen.tcl -type tcp -nn ۲۵ -seed ۰.۰ -mc ۸ > tcp-۲۵-test
```

یک اتصال از tcp-۲۵-test بدین صورت به نظر می رسد:

```
#
```

```
# ۵ connecting to ۷ at time ۱۶۳.۰۳۹۹۶۴۲۴۳۳۲۲۶
#
set tcp_(۱) [$ns_ create-connection TCP $node_(۵) TCPSink $node_(۷) ۰]
$tcp_(۱) set window_ ۳۲
$tcp_(۱) set packetSize_ ۵۱۲
set ftp_(۱) [$tcp_(۱) attach-source FTP]
$ns_ at ۱۶۳.۰۳۹۹۶۴۲۴۳۳۲۲۶ "$ftp_(۱) start"
```

۸.۲ ایجاد حرکت گره برای سناریوی بیسیم

مولد حرکت گره تحت دایرکتوری ~ns/indep-utils/cmu-scen-gen/setdest موجود بوده و از Makefile و setdest{.cc,.h} تشکیل شده است. به منظور کامپایل کردن setdest.cc به ترتیب زیر عمل کنید :

۱. به دایرکتوری ns رفته و "configure" را اجرا کنید (ممکن است زمانی که در حال ساختن ns بوده اید, اینکار را انجام داده باشید.) این کار یک makefile برای setdest ایجاد می کند.
۲. به indep-utils/cmu-scen-gen/setdest رفته و "make" را اجرا کنید, که در ابتدا یک فایل stand-alone object برای ~ns/rng.cc بوجود می آورد.(نسخه ی stand-alone از Tclcl libs استفاده نمی کند) و سپس setdest قابل اجرا را ایجاد می کند.
۳. setdest را با مولفه اجرا کنید, مانند آنچه که در زیر آورده شده است:

```
./setdest [-n num_of_nodes] [-p pausetime] [-s maxspeed] [-t simtime] \
```

[-x maxx] [-y maxy] > [outdir/movement-file]

می خواهیم یک سناریوی حرکت گره ایجاد کنیم که از ۲۰ گره تشکیل شده که با حداکثر سرعت ۱۰.۰m/s با میانگین وقفه بین حرکات معادل با ۲s حرکت می کند. می خواهیم که شبیه سازی بعد از ۲۰۰s متوقف شود.محدوده ی مرزی توپولوژی به صورت ۵۰۰*۵۰۰ تعریف شده است. بنابراین خط دستوریمان اینطور به نظر می رسد:

./setdest -n ۲۰ -p ۲.۰ -s ۱۰.۰ -t ۲۰۰ -x ۵۰۰ -y ۵۰۰ > scen-۲۰-test

خروجی توسط پیش فرض در stdout نوشته می شود.ما خروجی را مجددا به سمت فایل scen-۲۰-test هدایت می کنیم.این فایل با جایگاه اولیه ی گره ها شروع شده و تا تعریف حرکات گره پیش می رود.

$ns_ at ۲.۰۰۰۰۰۰۰۰۰۰ "$node_(۰) setdest ۹۰.۴۴۱۱۷۹۰۳۳۴۵۷ ۴۴.۸۹۶۰۹۵۵۴۴۰۱۰ ۱.۳۷۳۵۵۶۹۶۰۰۱۰"

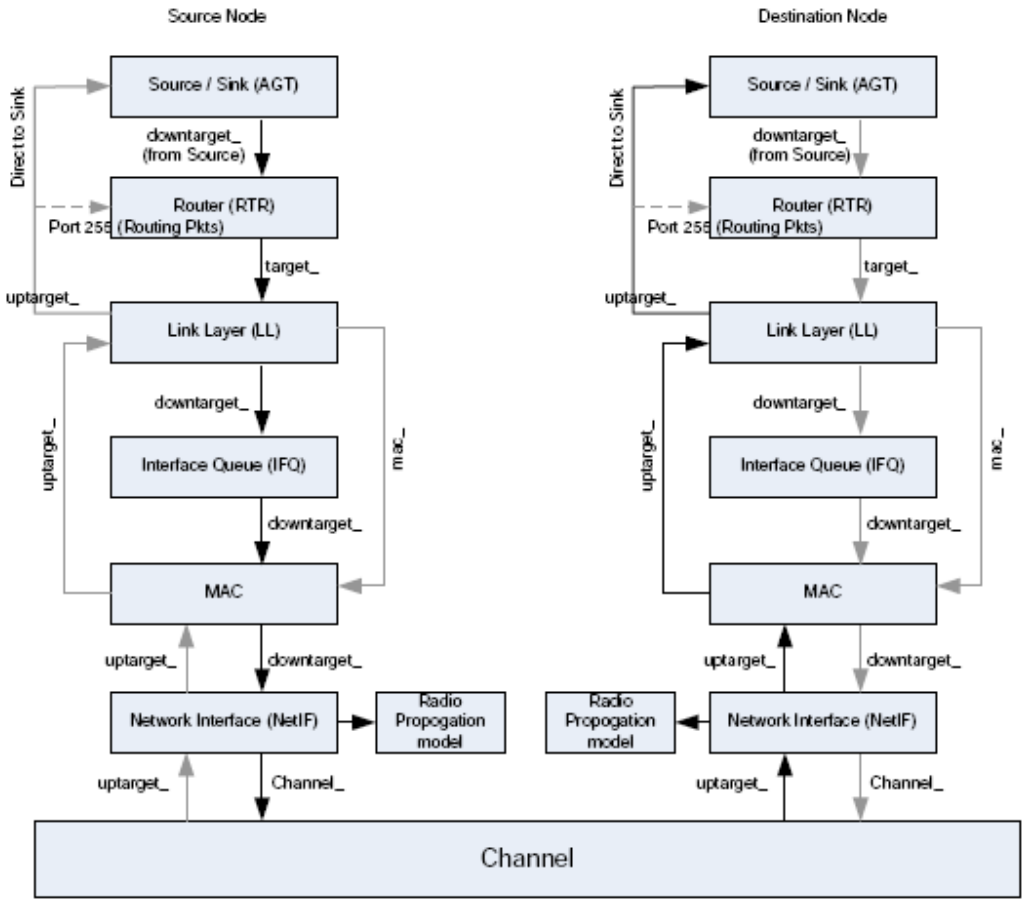
این خط از scen-۲۰-test مشخص می کند که node_(۰) در زمان ۲.۰s حرکت خود را به سمت مقصد (۹۰.۴۴, ۴۴.۸۹) با سرعتی معادل ۱.۳۷m/s شروع می کند. این خطوط فرمانی می توانند جهت تغییر جهت و سرعت حرکت گره های متحرک بکار روند.

۹ نوشتن پروتکل جدید در ns

۹.۱ شمای گره های متحرک در *ns*

در ns اجزائ شبکه بصورت کلاسهایی به نام agent طراحی شده اند. مانند agent های مسیریابی. این اجزا بر اساس کد شبیه سازی به یکدیگر مرتبط شده و شبکه مورد نظر را ایجاد می کنند.

در شکل ۹-۱ یک پیاده سازی شبکه ای ساده که از دو گره تشکیل شده است, نشان داده شده است. فلشهای کم رنگ, ارتباطات مرسوم بین agent ها می باشند و فلشهای مشکی ارتباطات فعال بین agent ها را مشخص می کند. Agent مبدا که در این مثال یک منبع ترافیکی CBR است, بسته های داده ای را بر اساس پشته پروتکل از طریق اشیائ شبکه به پایین ارسال می کند. در شیئ مسیریابی پرش بعدی مشخص می شود. صف رابط IFQ, لایه پیوند و لایه MAC بر اساس طول صف, برخورد و زمان پخش رادیویی, زمان تاخیری را به بسته ها اضافه می کنند.



۹-۱ شمای درونی یک گره متحرک در ns

۹.۲ شروع کد نویسی

ماقصدداریم یک پروتکل مسیریابی جدید راکه myroute نامیده می شود گام به گام اجراکنیم. برای اختصاص دادن کدهایمان، ابتدا یک directory جدید که Myroute نامیده می شود در کنار directory اصلی ns۲ بوجود می آوریم. ۵ فایل در اینجا بوجود می آوریم :

Myroute.h: این فایل header می باشد که دراین فایل تمام تایمرهای ضروری و عامل مسیریابی که پروتکل را اجرا می کند ،تعریف شده است.

Myroute.cc: در این فایل همه تایمرها و عامل مسیریابی به طورواقعی اجرا می شوند (فایل اصلی) .

Myroute-pk.h: در این فایل قالب بسته های پروتکل جدید مشخص می شوند.

Myroute-rtable.h: فایل header که کلاسهای جدول مسیریابی را مشخص می کند.

Myroute-rtable.cc: دراین فایل عملیات مسیر یابی پیاده سازی می شود.

Myroute-ntable.h : فایل header که کلاسهای جدول همسایگان را مشخص می کند.

Myroute-ntable.cc : دراین فایل عملیات مربوط به گره های همسایه پیاده سازی می شود.

اکنون ساختار فیزیکی(فایلها) راداریم، حال باید ساختار منطقی (کلاسها) ادامه دهیم را ایجاد کنیم. برای اجرای یک پروتکل مسیریابی درns۲ بایستی یک عامل با ارث بردن از کلاس Agent (عامل)تولید کنیم. این یک کلاس اصلی می باشد،که مجبوریم به منظور اجرای پروتکل مسیریابی مان ،کدگذاری کنیم. بعلاوه این کلاس، یک رابط TCLرا پیشنهاد می کند، که قادر به کنترل پروتکل مسیریابی مان ازطریق متن شبیه سازی نوشته شده درTCL، می باشیم.

عامل (agent) مسیریابی یک وضعیت داخلی ,یک جدول مسیریابی و یک جدول همسایگان را نگهداری می کند. وضعیت داخلی می تواند مانندیک کلاس جدید یا مانند مجموعه ای از صفات

درکنارعامل مسیریابی،نمایش داده شود. ما جدول مسیریابی و جدول همسایگان را مانند یک کلاس جدید موردعمل قرار می دهیم.

بنابراین پروتکل جدید ،بایستی حداقل یک نوع بسته جدید را که قالب بسته های کنترل خودش را نشان می دهد ،تعریف کند. همانگونه که گفتیم این نوع بسته هادر-myroute myroute/pkt.h تعریف شده اند. هنگامیکه پروتکل نیاز به ارسال بسته ها به صورت دوره ای دارد یا بعد از مدت زمانی یک رخداد اتفاق می افتد ، باید از یک کلاس Timer(تایمر) استفاده شود.

کلاس مهم دیگری که قبل ازورود به جزئیات باید بدانیم، کلاسTrace (ردگیری)می باشد که اساس نوشتن فایلهای log می باشد.

۹.۳ انواع بسته

بسته جدید را در فایل myroute/myroute-pkt.h تعریف می کنیم. قسمتی از تعریف آن در زیر آمده است :.

myroute-pkt.h/myroute

```
# ifndef _ _myroute_pkt_h_ _
#define _ _myroute_pkt_h_ _
#include <packet.h>
define HDR_MYROUTE_PKT(p) hdr_myroute_pkt::access(p)
struct hdr_myroute_pkt{
nsaddr_t   pkt_src_;    // Node which originated this packet
u_int۱۶_t  pkt_len_;     // Packet length (in bytes)
u_int۸_t   pkt_seq_num_; // Packet sequence number
inline    nsaddr_t&   pkt_src()    { return pkt_src_;}
inline    u_int۱۶_t& pkt_len()     { return pkt_len_;}
```

```
inline    u_int۸_t&   pkt_seq_num() { return pkt_seq_num_;}
static int offset _;
inline static int& offset() { return offset_;}
inline static hdr_myroute_pkt* access(const Packet* p) {
return (hdr_myroute_pkt*)p->access(offset_ );
        }
....
.....
.....
.....
};
#endif
```

در فایل myroute-pkt.h ساختار بسته جدید را بیان می کنیم. در تعریف اعلانهای زیر دیده می شود :

nsaddr-t : هرزمان که بخواهیم آدرس شبکه رادرns۲ اعلام کنیم بایداین نوع رااستفاده کنیم.

u-int۱۶-t:۱۶ بیت عددصحیح بدون علامت.

u-int۸-t:۸ بیت عددصحیح بدون علامت.

همه این انواع وبیشترشان درheader فایل config.h تعریف شده است.

کد شامل اضافه کردن common/packet.h می باشد که کلاس packet (بسته) است. بسته ها برای مبادله اطلاعات بین اشیاء درشبیه سازی استفاده می شود وهدفمان اضافه کردن ساختارجدید hdr-myroute-pkt به آنها می باشد. با انجام کنترل بسته ها باید قادر به ارسال و دریافت توسط گره ها درشبیه سازی باشیم. در ns header تمام بسته ها در یک آرایه کراکتری

بدون علامت (مجموعه ای از بیتها) می باشد. برای دسترسی به header یک بسته نیاز به دانستن مکان بسته در این آرایه می باشیم. برای این امر از offset استفاده می کنیم. تابع access نیز دسترسی به بسته را با استفاده از offset فراهم می آورد.

باید بین بسته و agent مسیریابی ارتباط برقرار کرد. این کار از طریق کدهای زیر انجام م شود :

```
int myroute_pkt::offset_;
static class MyrouteHeaderClass : public PacketHeaderClass{
public:
MyrouteHeaderClass() : PacketHeaderClass("PacketHeader/Myroute,"
sizeof(hdr_myroute_pkt )){
bind_offset(&hdr_myroute_pkt::offset;_)
}
} class_rtProtoMyroute_hdr;
```

۹.۴ عامل مسیریابی

اکنون ما برنامه نویسی خود عامل راشروع می کنیم. درون myroute/myroute.h یک کلاس جدیدکهmyroute نامیده می شود تعریف می کنیم که شامل صفتها و توابع موردنیاز برای کمک به پروتکل در اجرای وظیفه اش می باشدهست. به دلیل ارسال متناوب بسته های LIVE از timer استفاده می شود :

myroute/myroute.h

```
# ifndef __myroute_h__
#define __myroute_h__
```

```
#include  " myroute_pkt.h"
#include  <agent.h>
#include  <packet.h>
#include  <trace.h>
#include  <timer-handler.h>
#include  <random.h>
#include  <classifier-port.h>

#define CURRENT_TIME         Scheduler::instance().clock()
#define JITTER               (Random::uniform()*5)

 class Myroute;             // forward declaration

*/Timers /*

class Myroute_PktTimer : public TimerHandler{
public:
 Myroute_PktTimer(Myroute* agent) : TimerHandler () {
      agent_ = agent;
 }
 protected:
```

```
Myroute*    agent_;
 virtual void expire(Event* e);
} ;

*/Agent*/

class Myroute : public Agent{

/ * Friends /*
friend class Myroute_PktTimer;

 * Private members*/
nsaddr_t          ra_addr_;
myroute_state    state_;
myroute_rtable        rtable_;  myroute_ntable        ntable_;
int               accessible_var_;
u_int8_t          seq_num_;

protected:

 PortClassifier*    dmux_;      // For passing packets up to agents.
```

```
Trace*          logtarget_; // For logging.
   Myroute_PktTimer pkt_timer_; // Timer for sending packets.

inline nsaddr_t&       ra_addr()       { return ra_addr_;}
  inline myroute_state& state()         { return state_;}
inline int&            accessible_var() { return accessible_var_;}

void  forward_data(Packet*);
void  recv_myroute_pkt(Packet*);
void  send_myroute_pkt( );

void  reset_myroute_pkt_timer( );

public:

Myroute(nsaddr_t);
 int   command(int, const char*const*);
  void recv(Packet*, Handler*);

 };
```

۶۷: #endif

خطوط ۱۰-۴ شامل فایلهای headerمورد نیاز عامل می باشد.درذیل شرح می دهیم که چراآنها مفیدند.

myroute/myroute-pkt.h:سربرگ بسته راتعریف می کند.

Common/agent.h:کلاس اصلی عامل راتعریف می کند.

Common/packet.h:کلاس بسته راتعریف می کند.

common/timer-handler.h:کلاس اصلی تایمر راتعریف می کند.

Trace/trace.h: کلاس ردگیری را تعریف می کندکه برای نتایج خروجی نوشتن شبیه سازدریک فایل رهگیری استفاده شده است.

Classifier/classifier-port-h :کلاسportClassifier (رده بندی پورت) را تعریف می کندکه برای عبوربسته ها به لایه های بالاتر استفاده می شود.

خط۱۲یک ماکروی مفیدرابرای بدست آوردن زمان جاری تعریف می کند. این ماکرو با دسترسی به نمونه شیئی ازکلاس زمانبندی عملیاتش را انجام می دهد. این شیئ تمام رخدادهایی را که درطول شبیه سازی تولید شده اند و همچنین ساعت داخلی شبیه سازرا مدیریت می کند.

ماکروی دیگری درخط۱۳ می باشد. این تابع یک عددتصادفی درفاصله زمانی[۵-۰] تولید می کند. از این تابع برای تصادفی کردن ارسال بسته های کنترلی استفاده می شودکه ازهمزمانی یگ گره با همسایگانش که منجر به برخوردمیشود، جلوگیری می کند و بنابراین درزمان ارسال این بسته ها تاخیر رخ می دهد.

خطوط۲۷-۱۹ تایمر را برای ارسال دوره ای بسته های کنترلی بیان می کند.کلاس myroute_pktTimer از تایمر ارث برده می شود و ارجاعی به عاملی که آنرابوجودآورده است دارد و برای زمانبندی ارسال مجدد بسته استفاده می شود.

کلاسmyroute درخط های ۶۵-۳۱ تعریف شده است. در ابتدا متغییر های مورد نیاز مانند آدرس, جدول مسیریابی, جدول همسایگان و غیره تعریف شده است.

متغییر مهم دیگر، شی ردگیری(trace) می باشد (خط۴۶رابببینید)که برای تولیدlogها استفاده شده است ودرفایل trace ذخیره شده است.

خط۴۷ تایمرمعمولی رابیان می کند.وخطوط۵۱-۴۹ بخشهای تابع برای دسترسی به برخی خصوصیات داخلی می باشد.

تابع درخط ۵۳ برای فرستادن بسته های اطلاعاتی به مقصدصحیح آنها استفاده خواهدشد.تابع دیگردرخط۵۴ هروقت که یک بسته کنترلی دریافت شده باشد صدا زده می شود وتابع خط۵۵ برای ارسال یک بسته کنترلی استفاده می شود.

خط۵۷ یک تابع راکه به منظورزمانبندی خاتمه تایمر معمولی استفاده شده است.

خطوط۶۳-۶۱ شامل توابع عمومی کلاس myroute می باشد. ازکلاس پایه عامل(agent) ارث می بردکه نیازدارد ( )command,( )recاجراشده باشد.( )rec هروقت که عامل یک بسته دریافت می کندصدازده می شود. مثلا هنگامیکه یک گره خودش یک بسته راتولیدمی کند (درحقیقت لایه بالایی عامل مانندTCP یا UDP می باشد.) یا هنگامیکه از گره دیگری دریافت می شود رخ دهد.تابع ( )command دستورات ارجاع شده از کدهای شبیه سازی را مدیریت می کند.

۹.۵ ارتباطات ***TCL***

دربخش قبل دیدیم که چگونه بسته هایمان را به TCL مرتبط کنیم. اکنون مشابه آنرابرای کلاس عاملهایمان انجام می دهیم. هدف این است که که کاربران بتوانند با دستورات TCL به پروتکل جدید دستیابی داشته باشند. بنابراین برای انجام آن بایستی کلاسTCLclass همانطورکه در کدبعدی شرح داده می شود به ارث برده شود.

myroute/myroute.cc

```
١:static class MyrouteClass : public TclClass{
٢:public:
٣:    MyrouteClass() : TclClass("Agent/Myroute " ) {}
٤:TclObject* create(int argc, const char*const* argv} (
 ٥:       assert(argc ==) ٥;
٦:return
(new Myroute((nsaddr_t)Address::instance().str٢addr(argv[٤])));
٧: }
٨:} class_rtProtoMyroute;
```

سازنده کلاس درخط۳ می باشد و صرفا کلاس پایه توسط "agent/myroute" همانگونه که استدلال شد،صدازده می شود.

درخطوط۷-۴ یک شیئ Myroute ایجاد می کنیم.

۹.۶ تایمرها

ما بایستی درموردتایمرها که یک روش ( )expire می باشد ،در myroute/myroute.cc کدنویسی کنیم . اجرای آن تاحدی آسان می باشد زیرا فقط می خواهیم یک بسته کنترلی جدید را بفرستیم وتایمر خودش را دوباره زمانبندی کند.مطابق با تصمیمات طراحی ،این دوکار بایستی توسط عامل مسیریابی انجام شود ، بنابراین ،این فراخوانی ها همانند مثال بعدی درخواست می شود.

myroute/myroute.cc

```
١:void
٢:Myroute_PktTimer::expire(Event* e ){
٣:agent_->send_myroute_pkt( )
```

```
۴:agent_->reset_myroute_pkt_timer();
۵: }
```

۹.۷ عامل

۹.۷.۱ سازنده (Constructor)

در این قسمت مقادیر اولیه ای که باید مقدار دهی شوند را معرفی می کنیم. این کدها زمانی اجرا می شوند که کاربر در شبیه سازی گره ای از نوع Myroute ایجاد کند. عمل bind متغییر accessible_var_ توسط تابع bind_bool انجام می شود .

myroute/myroute.cc

```
۱:Myroute::Myroute(nsaddr_t id) : Agent(PT_MYROUTE),
pkt_timer_(this)}
۲: bind_bool("accessible_var_", &accessible_var_);
۳:  ra_addr_ = id;
۴: }
```

۹.۷.۲ command( )

تابع command مسئول اجرای دستوراتی است که در شبیه سازی بر روی شیئ مسیریابی انجام می شود و در حقیقت دستورات کاربر را انجام می دهد. مثلا چا÷ جدول مسیر یابی.

```
۱:int
۲:Myroute::command(int argc, const char*const* argv){
۳:    if (argc == ۲){
۴:     if (strcasecmp(argv[۱], "start") == ۰){
۵ :          pkt_timer_.resched(۰.۰);
```

```
        return TCL_OK;
        }
else if (strcasecmp(argv[1], "print_rtable") == 0){
 if (logtarget_ != 0){
sprintf(logtarget_->pt_->buffer(), "P %f _%d_ Routing Table,"
CURRENT_TIME,
   ra_addr());
logtarget_->pt_->dump();
  rtable_.print(logtarget_);
    }
else {
 fprintf(stdout, "%f _%d_ If you want to print this routing table"
  " you must create a trace file in your tcl script,"
                CURRENT_TIME,
               ra_addr());
    }
   return TCL_OK;
}
}
else if (argc == 3) {
// Obtains corresponding dmux to carry packets to upper layers
```

```
if (strcmp(argv[1], "port-dmux") == 0){
 dmux_ = (PortClassifier*)TclObject::lookup(argv[2]);
if (dmux_ == 0){
fprintf(stderr, "%s: %s lookup of %s failed\n",
        __FILE__ ,
          argv[1],
      argv[2],
   return TCL_ERROR;
}
return TCL_OK;
}
//    Obtains corresponding tracer
 else if (strcmp(argv[1], "log-target") == 0||
 strcmp(argv[1], "tracetarget") == 0){
logtarget_ = (Trace*)TclObject::lookup(argv[2]);
if (logtarget_ == 0)
return TCL_ERROR
 ;return TCL_OK;
}
}
//   Pass the command to the base class
```

```
۴۸:  return Agent::command(argc, argv);
۴۹:}
```

۹.۷.۳ تابع **rec( )**

تابع بعدی( )rec می باشد که هروقت عامل مسیریابی یک بسته را دریافت می کند، درخواست می شود.هر بسته یک header معمولی دارد که hde_cmn نامیده می شود که درcommon/packet.hتعریف شده است. برای دسترسی به این header یک ماکرو مانند آنچه که قبلا برای نوع بسته مان تعریف کردیم درخط۳ استفاده کرده ایم ،وجوددارد . خط۴ عمل مشابهی اما برای بدست آوردن IP header انجام می دهد. hdr_ip درip.h شرح داده شده است.

```
۱:void
۲: Myroute::recv(Packet* p, Handler* h){
۳:    struct hdr_cmn* ch = HDR_CMN(p);
۴: struct hdr_ip* ih   = HDR_IP(p);
۵:
۶:  if (ih->saddr() == ra_addr())}
۷: //       If there exists a loop, must drop the packet
۸:       if (ch->num_forwards() > ۰){
۹:        drop(p, DROP_RTR_ROUTE_LOOP);
۱۰:       return;
۱۱:        }
۱۲: //          else if this is a packet I am originating, must add IP header
۱۳:     else if (ch->num_forwards() == ۰)
```

```
۱۴:    ch->size() += IP_HDR_LEN;
۱۵:}
۱۶: //      If it is a myroute packet, must process it
۱۷:
۱۸:  if (ch->ptype() == PT_MYROUTE)
۱۹:    recv_myroute_pkt(p);
۲۰: //      Otherwise, must forward the packet (unless TTL has reached zero)
۲۱:    else{
۲۲:    ih->ttl_--;_
۲۳:    if (ih->ttl_ == ۰){
۲۴:    drop(p, DROP_RTR_TTL);
۲۵:    return;
۲۶:    }
۲۷:   forward_data(p);
۲۸:         }
۲۹:}
```

اولین چیزی که بایدچک شوداین مطلب است که بسته ای راکه خودمان ارسال کرده ایم دریافت نکرده ایم.اگر این حالت باشد،بایستی بسته رارها کنیم وبرگشت دهیم همانگونه که در خطوط۱۱-۸ انجام داده ایم.بعلاوه اگر بسته درداخل گره تولیدشده باشد(توسط لایه های بالاترازگره)بایستی طول بسته رابه سربار اضافه کنیم که پروتکل مسیریابی اضافه می

شود(دربایتها).مافرض می کنیمmyroute برروی IPکار می کندهمانگونه که درخطوط۱۴-۱۳ نشان داده شده است.

هنگامی که ما بسته ای ازنوع PT_MYROUTE دریافت می کنیم ،ما تابع-Recv-myroute pkt( ) را برای پردازش آن صدامی زنیم.(خطوط۱۹-۱۸)

اگربسته ازنوع داده باشد بایستی آنرا بفرستیم(اگرآن درگره دیگری تعریف شده باشد.)یا آنرابه لایه بالاترتحویل دهیم(اگرآن یک بسته پخشی باشدیاقبلا توسط خودمان انتخاب شده باشد.)مگراینکه TTL=۰ باشد.خطوط۲۸-۲۱انجام می دهدآنچه را درمورداستفاده از تابع-forward data( ) توصیف کردیم.

مابایددرک کنیم تابعdrop( )برای رهاکردن وارسال بسته هااستفاده شده است. این استدلالها اشاره ای به خودبسته وثابت ارائه کننده دلیل دورانداختن آن می باشد. چندین نوع از این ثابتهاوجود دارد.شما می توانید نگاهی به فایل trace/cmu-trace.h بیندازید.

۹.۷.۴ تابع **rec-myroute-pkt( )**

این تابع زمانی صدا زده می شود که گره , پیام LIVE و یا پیام کنترلی دیگری را از همسایگانش دریافت کند. در این تابع کدهای کنترلی و بروز رسانی جداول قرار داده می شود.

```
۱:void
۲:Myroute::recv_myroute_pkt(Packet* p){
۳:struct hdr_ip* ih          = HDR_IP(p);
۴:
۵:struct hdr_myroute_pkt* ph = HDR_MYROUTE_PKT(p);
۶: //    All routing messages are sent from and to port RT_PORT,
۷: //    so we check it.
```

```
۸:  assert(ih->sport() == RT_PORT);
۹:assert(ih->dport() == RT_PORT) ;
۱۰:
۱۱:/*processing of myroute packet……*/
۱۲:
۱۳: //release resources
۱۴: Packet:
۱۵:}
```

۹.۷.۵ تابع **send-myroute-pkt( )**

از این تابع برای ارسال پیامهای کنترلی به سایر گره ها استفاده می شود..

Myroute/myroute.cc

```
۱:void
۲:Myroute::send_myroute_pkt( ){
۳:  Packet* p                  = allocpkt();
۴: struct hdr_cmn* ch          = HDR_CMN(p);
۵:    struct hdr_ip* ih          = HDR_IP(p);
۶:  struct hdr_myroute_pkt* ph = HDR_MYROUTE_PKT(p);
۷:
۸: ph->pkt_src()        = ra_addr();
۹:    ph->pkt_len()        = ۷;
۱۰:  ph->pkt_seq_num()     = seq_num_++;
```

```
۱۱:
۱۲:ch->ptype()          = PT_MYROUTE;
۱۳:  ch->direction()      = hdr_cmn::DOWN;
۱۴: ch->size()           = IP_HDR_LEN + ph->pkt_len();
۱۵:  ch->error()          = ۰;
۱۶:  ch->next_hop()       = IP_BROADCAST;
۱۷:   ch->addr_type()      = NS_AF_INET;
۱۸:
۱۹:  ih->saddr()          = ra_addr;()
۲۰:  ih->daddr()          =IP_BROADCAST;
۲۱:  ih->sport()          = RT_PORT;
۲۲:  ih->dport()          = RT_PORT;
۲۳:  ih->ttl()            = IP_DEF_TTL;
۲۴:
۲۵:        Scheduler::instance().schedule(target_, p,JETTER);
۲۶:}
```

برای ارسال یک بسته ، می دانیم که ابتدا بایدآنرا تعیین کنیم.ما تابع allocate( ) را برای آن استفاده می کنیم.این تابع برای همه عاملها تعریف شده است. سپس به طورمعمول header های بسته myroute وip را بدست می آوریم(خطوط۶-۳).هدف ما پرکردن تمام این سربرگها با مقادیری که می خواهیم است.

header بسته myroute در خطوط ۱۰-۸ پرشده است.

header معمولی درns چندین فیلددارد.

۹.۷.۶ تابع **reset-myroute-pkt-timer( )**

تایمر ارسال بسته ،فراخوانی دیگری برای زمانبندی خودش انجام می دهد. آن درتابع-reset myroute-pkt-timer( )انجام شده است.

Myroute/myroute.cc

```
۱:void
۲:Myroute::reset_myroute_pkt_timer (){
۳:Pkt_timer_.resched((double)۵.۰);
۴:}
```

۹.۷.۷ تابع **forward-data()**

به طوراصولی برروی بسته هایmyroute تمرکز می کنیم،اماآن زمان مبادله با بسته های داده می باشد.تابع ( )forward-data تصمیم می گیردکه آیا یک بسته مجبوراست به عاملهای لایه بالاتر تحویل داده شود یا به گره دیگری ارسال می شود.

Myroute/myroute.cc

```
۱:void
۲:Myroute::forward_data(Packet* p){
۳:   struct hdr_cmn* ch = HDR_CMN(p);
۴:struct hdr_ip* ih = HDR_IP(p);
۵:
۶:    if (ch->direction() == hdr_cmn::UP&&
۷:    ((u_int۳۲_t)ih->daddr()=IP_BROADCAST|| ih->daddr()= ra_addr ())){
```

```
٨:    dmux_->recv(p, ٠.٠);
٩: return;
١٠:}
١١: else {
١٢:ch->direction() = hdr_cmn::DOWN;
١٣: ch->addr_type() = NS_AF_INET;
١٤:         if ((u_int٣٢_t)ih->daddr() == IP_BROADCAST)
١٥:        ch->next_hop() = IP_BROADCAST;
١٦:         else {
١٧:        nsaddr_t next_hop = rtable_.lookup(ih->daddr());
١٨:   if (next_hop == IP_BRADCAST;
١٩:  debug("%f:Agent %d can not forward a packet destined to %d
loss=%d\n,"
٢٠:                    CURRENT_TIME,
٢١:             ra_addr(),
٢٢:         ih->daddr());
٢٣:drop(p, DROP_RTR_NO_ROUTE);
٢٤:                  return;
٢٥:}
٢٦:       else
٢٧:       ch->next_hop() = next_hop;
```

```
۲۸:}
۲۹:Scheduler::instance().schedule(target_, p, ۰.۰);
۳۰:}
۳۱:}
```

۹.۸ تغییرات موردنیاز

بعد از نوشتن کدها باید آنها را در NS اعلام کنیم تا بت.انیم در شبیه سازی از پروتکل مورد نظر استفاده کنیم.

۹.۸.۱ اعلام نوع بسته

باید نوع بسته را در فایل packet.h در ساختار packet_t اضافه کنیم.

Common/packet.h

```
۱: enum packet_t {
۲: PT_TCP,
۳:PT_UDP,
۴:PT_CBR,
۵:/*…much more packet types …*/
۶:PT_MYROUTE,
۷:PT_NTYPE //This MUST be the LAST one
۸: };
```

در این تابع باید نامی به پروتکل جدید بدهیم :

Common/packet.h

```
۱: p_info( ) {
```

```
۲:name_[PT_TCP]="tcp";
۳: name_[PT_UDP]="udp";
۴:name_[PT_CBR]="cbr";
۵:/*…much more m=names…*/
۶: name_[PT_MYROUTE]="myroute";
۷:}
```

۹.۸.۲ پشتیبانی ردگیری (رد یابی)

همانگونه که می دانیم هدف شبیه ساز بدست آوردن یک فایل رهگیری است که توصیف می کندچه اتفاقاتی درحین اجرا رخ داده است. یک شی رهگیری(trace)برای نوشتن اطلاعات خواسته شده ازیک بسته ،درهرزمان که دریافت ،ارسال یا رها (انداخته) شده است،استفاده می شود.برای گزارش اطلاعات نوع بسته ها تابع ( ) format_myroute رادرون کلاس CMUTrace اضافه می کنیم.

برای این منظور باید کدهای زیر را به فایل trace/cmu-trace.h اضافه کنیم.

trace/cmu-trace.h

```
۱: class CMUTrace :public TRACE {
۲:   /*…definitions…*/
۳:private:
۴:      /*…*/
۵:void format_aodv(packet *p, int offset);
۶:void format_myroute(packet *p, int offset);
۷:};
```

بخش بعدی کدانواع متفاوتی ازردگیری ها را نشان می دهد.(ازفایل trace/cmu- trace.cc استخراج شده است)

trace/cmu- trace.cc

```
۱: #include <myroute/myroute_pkt.h>
۲:
۳: /*...*/
۴:
۵:  void
۶:CMUTrace::format_ip(Packet *p, int offset(
۷:{
۸:struct hdr_myroute_pkt*ph= HDR_MYROUTE_PKT(P);
۹:
۱۰:       if (pt_->tagged()){
۱۱:              sprintf(pt_->buffer() + offset,
۱۲:              -"Myroute:o %d -myroute:sp %d –myroute:l%d" ,
۱۳:           ph->pkt-src(),
۱۴:           ph->pkt-seq_num(),
۱۵:          ph->pkt-len());
۱۶:
۱۷:       {else if (newtrace_){
۱۸:          sprintf(pt_->buffer() + offset,
۱۹:            "-p myroute –po %d –ps %d –pl %d",
```

```
٢٠::            ph->pkt-src(),
٢١:             ph->pkt-seq_num(),
٢٢:            ph->pkt-len());
٢٣: }
٢۴: else {
٢۵:                 sprintf(pt_->buffer() + offset,
٢۶: "[myroute  %d %d %d] ",
٢٧:                  ph->pkt-src(),
٢٨:             ph->pkt-seq_num(),
٢٩:            ph->pkt-len());
٣٠: }
٣١: }
```

مامی توانیم برای کدبالا نتیجه گیری کنیم که سه قالب متفاوت رهگیری وجود دارد که عبارتنداز : ره گیری برچسب ره گیری ،شکل ره گیری ورده بندی ره گیری. ترکیب دنبال شده درهریک، با این وجود، متفاوت، خیلی آسان وذاتا همانگونه ای که می گوییم، می باشد. برچسب های موجود برای شناسایی هرفایل اطلاعات است که چاپ شده است. تصمیم داریم "O"راهمانند آدرس منبع(origin(مبدا))و"S"را همانند ترتیب اعداد و"L"را همانند طول بسته مرتبط ،استفاده کنیم.

به منظور اینکه تمام توابع ایجادشده اخیر را صدا بزنیم بایدتابع(format را در Trace/cmu-trace.ccتغییردهیم.

Trace/cmu-trace.cc

```
١:Void
٢;CMUTrace::format(Packet* p, const char *why)
```

```
۳:{
۴:        /*…*/
۵:                case PT_PING:
۶:                break;
۷:
۸:                case PT_MYROUTE:
۹:            format_myroute(p, offset);
۱۰:break;
۱۱:
۱۲:  default:
۱۳:    /*…*/
۱۴:}
```

۹.۸.۳ کتابخانه **TCL**

اکنون نیاز به انجام بعضی تغییرات در TCLداریم. درواقع قصد داریم نوع بسته مان را اضافه کنیم،مقادیر پیش فرض را برای خصوصیات بهم وابسته مان بدست آوریم و زیر ساختارمورد نیاز برای ایجادگره های بی سیم را درحین اجرای پروتکل مسیریابی Myroute فراهم می آوریم .

بایدکد بعدی را درtcl/lib/ns-packet.tcl قراردهیم و myrouteرا به لیست اضافه کنیم.(همانگونه که در خط ۲ انجام داده ایم)

tcl/lib/ns-packet.tcl

```
۱: foreach port {
۲:  myroute
```

۳: AODV

۴: ARP

۵: #...

۶: NV

۷: }{

۸: add-packet-header $port

۹: }

برای مقداردهی اولیه به خصوصیات باید آن ها را در فایل ns-default.tcl مقداردهی کنیم.

tcl/lib/ns-default.tcl

۱:#...

۲: #default defined for myroute

۳:Agent/myroute set accessible_var_true

در پایان باید مجبوریم tcl/lib/ns-lib.tcl اصلاح شود. باید یک دستور OTCL برای راه اندازی پروتکل مسیریابی خود ایجاد کنیم. این دستورات را باید در فایل ns-lib.tcl بنویسیم.

tcl/lib/ns-lib.tcl

۱: simulator instproc create-wireless-node args {

۲: #...

۳: switch-exact $routingAgent- {

۴: myroute

۵: set ragent [$self create-myroute-agent $node]

۶: }

```
۷:  #...
۸:  }
۹: #...
۱۰:}
```

tcl/lib/ns-lib.tcl

```
۱: simulator instproc create-myroute-agent {node} {
۲:  #create myroute routing agent
۳:  set ragent [new Agent/myroute [$node node-addr]]
۴: $self at ۰.۰ "$ragent start"
۵: $node set ragent_$ragent
۶: return $ragent
۷: }
```

خط۳ یک عاملmyrouteتوسط آدرس گره بوجود می آورد.

۹.۸.۴ اولویت صف

برای اولویت بندی بسته پروتکل جدید باید کد زیر را در فایل priqueue.cc اضافه کنیم.

Queue/priqueue.cc

```
۱:void
۲:priQueue::recv(packet *p,Handler *h)
۳:{
۴:  struct hdr_cmn *ch=HDR_CMN(p);
۵:
```

```
٦:  if (prefer_Routing_Protocols){
٧:
٨:     switch(ch->ptype()) {
٩:   case PT_DSR:
١٠:  case PT_MESSAGE:
١١:  case PT_TORA:
١٢:  case PT_AODV:
١٣:  case PT_MYROUTE:
١٤:  recHighPriority(p,h);
١٥:    break;
١٦:
١٧:  default:
١٨:      Queue::recv(p,h);
١٩:}
٢٠:}
٢١:else {
٢٢:   Queue::recv(p,h);
٢٣: }
٢٤:  }
```

۹.۸.۵ ایجادفایل(**makefile**)

اکنون باید فایلها را کامپایل کنیم. برای انجام، فایل makefile رابا اضافه کردن شی فایلهادرون متغیرOBJ-CC همانگونه که درذیل مشاهده می شود(خط۴)،ویرایش می کنیم.

```
۱:OBJ-CC =\
۲: tools/random.o tools/rng.o tools/ranvar.o common/misc.o common/timer-handler.o
۳: #...
۴: myroute/myroute.o myroute/myroute-rtable.o myroute/myroute-ntable.o \
۵: #...
۶:$(OBJ_STL)
```

سپس با دستور make فایلهای o لازم ایجاد می شود و پروتکل مسیریابی به ns اضافه می شود.

```
[ns-۲-۲۷]$ make
```