فصل دوم

مفاهیم پایه

۲-۱ مقدمه

در ns برای شبیه‌سازی شبکه‌ها از زبان Tcl Script استفاده می‌شود Tcl Script یک زبان اسکریپتی است که از آن برای معرفی گره‌های شبکه، لینک‌های بین گره‌ها، پهنای باند لینک‌ها و سایر مشخصات شبکه استفاده می‌شود.

برای استفاده از شبیه‌ساز، در ابتدا باید توپولوژی شبکه دلخواه خود را با استفاده از Tcl Script در یک فایل معرفی کرده و در انتها فایل را با پسوند tcl. ذخیره کنید. می‌توانید این کار را با ویرایشگرهای موجود در لینوکس مثل emacs انجام دهید. در انتها با اجرای دستور ns filename.tcl می‌توانید شبیه‌سازی را انجام دهید.

در این فصل به بررسی مفاهیم پر کاربرد و اساسی در بسته نرم‌افزاری NS می‌پردازیم. در ابتدا بحث خود را با زبان دست نویس otcl به عنوان رابط کاربر ns و همچنین اصل Tcl Script آغاز و سپس با ارایه مثال‌هایی، قابلیت‌های آن عنوان خواهد شد.

۲-۲ OTcl : زبان کاربر

همانگونه که در بخش قبل ذکر شد، ns در واقع یک مفسر OTcl به همراه کتابخانه‌ای از اشیاء شبیه‌سازی شبکه است. به همین سبب آشنایی با زبان دست‌نویس OTcl برای استفاده از ns ضروری و اجتناب‌ناپذیر می‌باشد. در این بخش مثال‌هایی از دست‌نویس‌های Otcl و Tcl به منظور انتقال مفاهیم بنیادی و پایه‌ای این محیط دست‌نویس ارائه می‌شود. با استفاده از این مثال‌ها ایده‌های اصلی برنامه‌سازی در OTcl به منظور برپایی همبندی شبکه و شبیه‌سازی آن منتقل می‌شود. این مثال‌ها از پنجمین کارگاه معرفی ns استخراج شده و با زبان‌های برنامه‌سازی C و C++ و مفاهیم اولیه برنامه‌سازی شیءگرا آشنایی دارد.

شکل ۲-۱ یک دست‌نویس کلی به زبان Tcl است. در این مثال نحوه ایجاد رویه[1] و فراخوانی آن، انتصاب مقادیر به متغیرها و نحوه ایجاد حلقه‌های تکرار نشان داده شده است. با عنایت بر این که OTcl گونهٔ گسترش یافته و شیءگرای زبان دست‌نویس Tcl است، این نکته روشن می‌شود که تمامی دستورات Tcl در OTcl نیز قابل استفاده می‌باشند. در واقع ارتباط بین Tcl و OTcl مشابه ارتباط زبان‌های C و C++ است. برای اجرای این دست‌نویس باید نام دست‌نویس (به عنوان مثال ex-tcl.tcl) را به عنوان پارامتر برنامه ns معرفی نمایید. در صورتی که tcl8.0 بر روی کامپیوتر شما نصب شده باشد، فرمان tcl ex-tcl.tcl نیز همین عمل را انجام خواهد داد.

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

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

شکل ۲-۱: یک دست‌نویس ساده Tcl

در زبان دست‌نویس Tcl کلمه کلیدی proc برای تعریف یک رویه به کار برده می‌شود. به دنبال این کلمه کلیدی، نام رویه و سپس پارامترهای رویه (در داخل {}) ذکر می‌شوند. کلمه کلیدی set برای انتصاب یک مقدار به یک متغیر به کار برده می‌شود. عبارت [expr...] سبب می‌شود که شبیه‌ساز مقدار عبارت داخل قلاب ([]) و مقابل کلمه کلیدی expr را محاسبه نماید. نکته قابل توجه دیگر آن است که برای دسترسی به مقدار یک متغیر می‌بایست از نماد $ قبل از نام متغیر استفاده نمود. (به خط if($k<5) در شکل ۲-۱ توجه کنید.) کلمه کلیدی puts رشته کاراکتری ذکر شده و محصور در نمادهای گیومه را برروی صفحه نمایش نشان می‌دهد. خروجی این مثال به شرح زیر است:

1. Procedure

```
k < 5, pow = 1.0
k < 5, pow = 1120.0
k < 5, pow = 1254400.0
k < 5, pow = 1404928000.0
k < 5, pow = 1573519360000.0
k >= 5, mod = 0
k >= 5, mod = 4
k >= 5, mod = 0
k >= 5, mod = 0
k >= 5, mod = 4
```

شکل ۲-۲: خروجی مثال ۲-۱

مثال بعدی نمونه‌ای از برنامه‌سازی شئ‌گرا را در دست‌نویس‌های OTcl نشان می‌دهد. این مثال بسیار ساده است و چگونگی ایجاد و استفاده از یک شئ را نشان می‌دهد. در صورتی که شما یک کاربر معمولی و عادی ns باشید، احتمال نوشتن یک شئ جدید توسط شما خیلی کم خواهد بود ولی از آنجا که تمام اشیاء در ns چه توسط زبان برنامه‌نویسی C++ نوشته شده و به دست‌نویس OTcl پیوند خورده باشند و چه اساساً در OTcl ایجاد شده باشند؛ نهایتاً به صورت یک شئ OTcl رفتار می‌کنند، لذا درک اشیاء OTcl سودمند خواهد بود.

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

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

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

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

شکل ۲-۳: یک دست‌نویس ساده OTcl

مثال ۲-۳ یک دست‌نویس OTcl است که دو کلاس (شئ)، mom و kid را تعریف کرده و شئ kid فرزند شئ mom می‌باشد. در هر دو شئ (کلاس) تابع greet نیز وجود دارد. پس از تعریف اشیاء نمونه‌ای[1] از هر یک توصیف می‌شود و متغیر age در هر یک مقداری را می‌پذیرد. (در کلاس mom مقدار

1. Instance

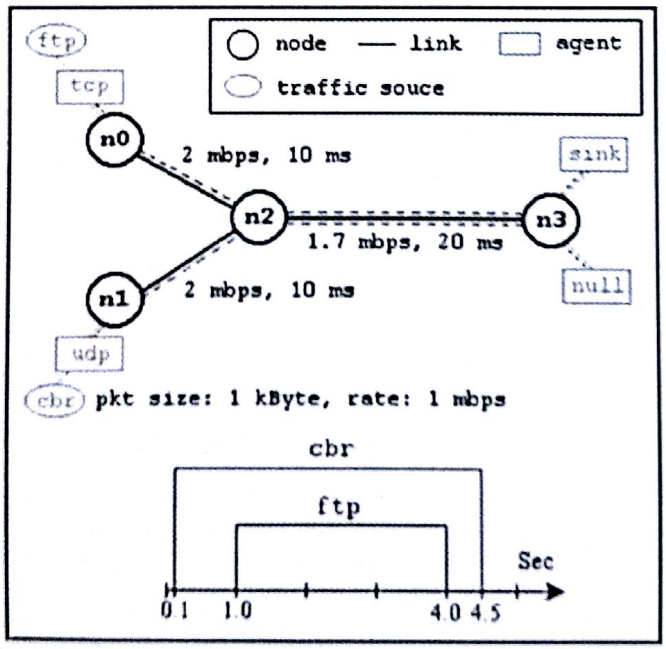
۴۵ و در کلاس kid مقدار ۱۵ در متغیر age جایگزین می‌شود) سپس تابع greet فراخوانی می‌شود. در این مثال کلمه کلیدی class یک شئ ایجاد کرده و کلمه کلید instproc یک تابع (Member Function) را تعریف می‌نماید. برای برقراری رابطه توارث[1] از کلمه کلیدی superclass استفاده شده است. در هنگام تعریف توابع (متدها) کلمه کلیدی $self مشابه اشاره‌گر this در زبان برنامه‌سازی C++ عمل می‌کند. عبارت instvar کنترل می‌کند که آیا متغیر ذکر شده در این کلاس (شئ) یا در کلاس والد (شئ جد) تعریف شده است یا خیر. اگر متغیر تعریف شده باشد، همان متغیر ارجاع داده می‌شود و اگر تعریف نشده باشد، یک متغیر جدید تعریف می‌شود. در نهایت برای ایجاد یک شئ نمونه[2] از کلمه کلیدی new استفاده می‌شود. خروجی این مثال به شکل زیر است:

```
45 year old mom say:
    How are you doing?
15 year old kid say:
    What's up, dude?
```

شکل ۲-۳: خروجی مثال ۲-۲

۲-۲-۱ مثال ساده شبیه‌سازی

در این بخش به بررسی یک مثال ساده شبیه‌سازی می‌پردازیم. شکل ۲-۵ یک سناریوی شبیه‌سازی ساده را نشان می‌دهد. در این مثال یک دست‌نویس ساده OTcl دیده می‌شود که یک پیکربندی ساده شبکه را ایجاد می‌کند. این پیکربندی سناریوی شبیه‌سازی شکل ۲-۵ را اجرا می‌کند. برای اجرای این مثال فایلی به نام ns-simple.tcl را ایجاد کرده و دستورات مربوطه (پیوست ۱) را در آن وارد کنید. سپس با استفاده از فرمان ns ns-simple.tcl سناریوی شبیه‌سازی را اجرا کنید.



شکل ۲-۵: یک همبندی ساده شبکه و سناریوی شبیه‌سازی آن

1. Inheritance
2. Instance

۲-۳ نحوه شبیه‌سازی با ns

همانطور که گفته شد جهت انجام شبیه‌سازی در ns ابتدا باید یک فایل TCL ایجاد کرد و در فایل فوق از دستورات موجود در ns و همچنین از دستورات TCL برای بیان توپولوژی شبکه و سایر ویژگی‌های شبکه موردنظر استفاده نمود.

مهم‌ترین این ویژگی‌ها عبارتند از:

- تعداد نودها
- مشخصه هر نود
- مشخصه لینک‌ها اتصال‌دهنده نودها
- نوع پروتکل‌های متصل به نودهای مبدأ و مقصد
- انواع ترافیک ارسالی
- زمان شبیه‌سازی و زمان‌های فعال شدن هر منبع ترافیکی
- مشخصات فایل‌های مربوط به مونیتور کردن ترافیک‌های شبکه و استخراج مشخصه‌های کارآیی شبکه

۲-۳-۱ شروع کار با ns

برای شروع ما یک template برای تمامی اسکریپت‌های اولیه خود ایجاد می‌کنیم. شما می‌توانید با این فرمت در هر ویراستاری سناریوهای خود را بنویسید. توجه داشته باشید که در انتها باید فایل خود را با پسوند Tcl ذخیره کنیم.

در ابتدا ما یک شئ از نوع شبیه‌ساز را ایجاد می‌کنیم و سپس یک فایل خروجی برای نوشتن خروجی برای nam باز می‌کنیم. داده‌های trace به این فایل ریخته خواهند شد. توجه کنید که قبل از هر شئ $ را باید قرار دهید.

در اولین قدم برای انجام شبیه‌سازی، باید یک شئ از کلاس شبیه‌ساز ایجاد گردد. این امر با کمک دستور زیر امکان‌پذیر است:

```
set NS[ new  Simulator]
```

بعد از دستور فوق، با کمک دستورات NS و OTcl، توپولوژی شبکه که شامل نودها، لینک‌ها و عاملان شبکه می‌باشند، ایجاد می‌شوند. بعد از ایجاد نودها و عاملان شبکه، با کمک لینک‌ها، نودهای شبکه به یکدیگر متصل می‌شوند و همچنین عاملان شبکه نیز به نودهای مربوطه اتصال می‌یابند، به عنوان مثال، پروتکل‌های مسیریابی دینامیکی، منابع ترافیکی و بسیاری از پروتکل‌های لایه ارسال نمونه‌ای از عاملان شبکه می‌باشند.

بعد از ایجاد عاملان شبکه و اتصال آنها به نودها، می‌توان مشخصه‌های آنها را نیز تنظیم نمود. به عنوان مثال دو دستور زیر یک عامل ارسال از نوع TCP تعریف می‌نمایند و سپس مشخصه طول پنجره ارسال که یک متغیر در پروتکل TCP است را به مقدار ۲۵ مقدار دهی می‌کنند.

```
set tcp[new Agent/TCP]
$tcp set window_ ۲۰
```

برای زمان‌بندی رخدادها در NS، با کمک دستور at می‌توان در هر لحظه دلخواه در طول زمان شبیه‌سازی روال‌های otcl را فعال نمود. بدین‌ترتیب امکان تعیین زمان شروع و پایان ارسال منابع ترافیکی، ایجاد خرابی‌های موقتی در لینک‌های شبکه در زمان‌های خاص، پیکره‌بندی دوباره توپولوژی شبکه و مانند اینها فراهم می‌آید.

به عنوان مثال دستوراتی که در زیر آمده است، باعث می‌شود که منبع ترافیکی از قبل تعیین شده $cbr در زمان ۰،۵ شروع به ارسال ترافیک نماید و در زمان ۴،۵ متوقف گردد.

```
$NS at ۰.۵ "$cbr۰ start"
$NS at ٤.۵ "$cbr۰ stop"
```

با کمک دستور run شبیه‌سازی آغاز می‌شود و همچنین دستورات stop یا exit باعث خاتمه عملیات شبیه‌سازی می‌گردند.

۲-۳-۲ فرمان‌های اولیه ns-2

برای شبیه‌سازی، در ابتدا باید یک نمونه از شئ Simulator ایجاد گردد. این کار با دستور زیر انجام می‌گیرد:

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

برای ذخیره اطلاعات مورد استفاده nam trace باید فایلی باز شود. برای این کار از دستور زیر استفاده می‌کنیم:

```
set nf [open out.nam w]
```

این دستور، فایل out.nam را برای نوشتن باز می‌کند و nf, handler آن می‌باشد. یعنی برای اشاره به فایل خروجی از nf استفاده خواهد شد.

برای این که تمام خروجی‌های مرتبط با nam به فایل nf منتقل شوند از دستور زیر استفاده می‌شود:

```
$ns namtrace-all $nf
```

در خاتمه شبیه‌سازی از تابع finish استفاده می‌شود. که فایل‌های شبیه‌سازی را بسته، بافرها را آزاد کرده و سایر کارهای لازم را انجام می‌دهد.

۲-۳-۳ دستورات NS

در این بخش به بررسی دستورات ns برای ایجاد نودهای شبکه، تعریف لینک‌ها اتصال دهنده نودها، تعیین نوع مکانیسم صف‌بندی در بافر نودها، نحوه نمایشی کردن لینک‌ها و استخراج اطلاعات آماری، تعریف عاملان ارسال و نحوه اتصال آنها به نودها، تعیین نوع مسیریابی در شبکه و تعریف منابع ترافیکی مختلف، می‌پردازیم.

همچنین مشخصه‌های مربوط به هر دستور نیز مورد بررسی قرار می‌گیرند.

در نرم‌افزار شبیه‌ساز ns، شبیه‌سازی هر نود با کمک دستور زیر انجام می‌شود:

```
set node_name [$NS node]
```

که در دستور فوق node_name نام نود شبکه می‌باشد. با ایجاد هر نود، ns یک عدد یکتای مشخص کننده نود به آن نسبت می‌دهد. بعد از ایجاد نودهای شبکه، عاملان ارسال تعریف گردیده و به نودهای شبکه متصل می‌شوند. آدرس هر عامل ارسال در ns دارای طول ۱۶ بیت می‌باشد. که ۸ بیت بالایی آن نشان‌دهنده نود و ۸ بیت بعدی نشان‌دهنده عامل ارسال در نود فوق می‌باشد. بنابر این با توجه به محدودیت ۸ بیتی در تعیین آدرس نودها، نمی‌توان در ns بیشتر از ۲۵۶ نود ایجاد نمود و اگر چنانچه تعداد نودها از ۲۵۶ بیشتر باشد، باید فیلد آدرس نود توسعه یابد. بدین منظور از دستور زیر استفاده می‌شود:

```
Node expander
```

دستور فوق باعث می‌شود که فضای آدرس‌دهی به ۳۰ بیت توسعه یافته و از ۲۲ بیت بالای آن برای تخصیص آدرس نود استفاده می‌گردد.

حالت پیش‌فرض عملکرد هر نود در ns، به صورت یک پراکنی می‌باشد ولی چنانچه بخواهیم نودهایی با قابلیت چند پراکنی ایجاد کنیم، باید متغیر کلاسی EnableMcast را برابر با یک قرار دهیم.

با ورود هر بسته به یک نود (از طریق ورودی نود Node entry)، ابتدا آدرس مقصد بسته توسط واحد دیگر باشد، بسته دریافتی به یکی از لینک‌ها خروجی ارسال می‌شود. ولی اگر عامل مقصد بسته ورودی به نود فعلی متصل باشد، در این صورت از طریق واحد Port classifier شماره درگاه بسته ورودی تجزیه و تحلیل شده و سپس بسته ورودی عامل ارسال مربوطه انتقال می‌یابد.

در هنگام استفاده از مسیریابی دینامیکی، بالاترین بیت در ناحیه آدرس، نشان‌دهنده چند پراکنی یا یک پراکنی بودن آدرس می‌باشد. چنانچه بیت فوق صفر باشد، آدرس دهی از نوع یک پراکنی است و در غیر این صورت آدرس از نوع چند پراکنی می‌باشد بنابراین در حالت چند پراکنی حداکثر تعداد نودها برابر با ۱۲۸ است.

برای پیکره‌بندی و کنترل نودها در ns از روال‌های زیر استفاده می‌شود:

۲-۳-۳-۱ عملیات کنترلی

این عملیات در قالب دستورات ns زیر انجام می‌گردند.

- $node entry

این دستور مقدار مشخص‌کننده نقطه ورودی نود را که اولین المان نود است و به وسیله آن بسته‌های ورودی پردازش می‌شوند را بازگشت می‌دهد.

- $ node reset

تمام عامل‌های متصل به نود را مجدداً آغاز می‌نمایند.

- $ node reset port

عاملی که به درگاه شماره port در نود node متصل است را مجدداً آغاز می‌نماید.

- $node enable_ mcast

یک روال داخلی ns می‌باشد که به وسیله آن یک نود از نوع یک پراکنی به یک نود چند پراکنی تبدیل می‌گردد.

۲-۳-۳-۲ عملیات مدیریت درگاه‌ها و آدرس نودها

این عملیات توسط دستورهای زیر صورت می‌گیرد:

- $node id

یک عدد یکتا که مشخص کننده نود می‌باشد را برمی‌گرداند.

- $node agent port

این دستور یک مشخص کننده برای عامل agent با درگاه شماره port برمی‌گرداند.

- $node join_group agent group

این دستور، عامل agent را به کلیه میزبان‌های چند پراکنی که با شماره گروه group مشخص شده‌اند، اضافه می‌کند.

۲-۳-۳-۳ عملیات مدیریت عامل

برای اتصال یک عامل جدید به نود و یا حذف عامل قبلی از نود موردنظر، از دستورات زیر استفاده می‌شود:

- $NS attach_agent node agent
- $NS detach_agent node agent

دو دستور فوق به ترتیب باعث اتصال یا قطع عامل در agent در نود node می‌شوند.

۲-۳-۳-۴ ردیابی همسایگان

در ns هر نود یک لیستی از همسایگان مجاور خود را نگهداری می‌کند. بدین‌منظور از دستور زیر استفاده می‌شود:

- $node neighbors

این دستور لیستی از مشخص‌کننده‌های نودهای همسایه نود node را برمی‌گرداند.

۲-۳-۴ لینک‌ها در NS

در این بخش، یکی دیگر از المان‌ها تشکیل‌دهنده توپولوژی شبکه را که لینک می‌باشد، بررسی می‌کنیم. فقط لینک‌های نقطه به نقطه مورد بررسی قرار می‌گیرند. البته شبیه‌ساز ns علاوه بر لینک‌ها نقطه به نقطه، سایر لینک‌ها متداول مانند: لینک‌ها با دسترسی چندگانه شبکه‌های محلی و لینک‌ها شبکه‌های بدون سیم را نیز پشتیبانی می‌کند.

هر لینک در شبیه‌ساز ns توسط پنج متغیر زیر تعریف می‌شود:

- head : نقطه ورودی به لینک که اولین شئ لینک می‌باشد.
- queue : نشان‌دهنده آدرس صف موجود در لینک است. در لینک‌های کاملاً یک طرفه یک صف وجود دارد، در حالی که در سایر لینک‌ها امکان وجود بیشتر از یک صف می‌باشد.
- link : اشاره‌کننده به المانی است که لینک را با مشخصه‌های داده شده مدل‌سازی می‌کند.
- Ttl : اشاره‌گر به المانی است که مقدار زمان زندگی (TTL) هر بسته را نگهداری می‌کند.
- Drophead : اشاره‌گر به شئ است که اتلاف در لینک را پردازش می‌کند.

چنانچه متغیر $trace All File تعریف شده باشد، در این صورت امکان ردیابی بسته‌ها از لحظه ورود به صف تا لحظه خروج از آن وجود دارد.

۲-۳-۴-۱ دستورات تعریف لینک‌ها در ns

دستورات زیر برای تعریف لینک‌ها در ns به کار می‌روند.

دستور زیر برای ایجاد یک لینک یک طرفه بین دو نود n۱ و n۲ به کار می‌رود. متغیرهای دستور فوق به صورت زیر تعریف می‌شوند:

```
$NS simplex-link n۱ n۲ bw delay q_type
```

- bw : میزان پهنای باند لینک اتصال‌دهنده نودهای n۱ و n۲
- delay : مقدار تأخیر لینک اتصال‌دهنده نودهای n۱ و n۲
- q_type – : نوع مکانیسم صف‌بندی در بافر لینک که بعداً بیشتر در مورد آن توضیح خواهیم داد.

```
$NS duplex-link n۱ n۲ bw delay q_type
```

مشابه دستور قبل می‌باشد با این تفاوت که لینک اتصال دهنده نودها، دو طرفه می‌باشد.

```
$NS simplex-link-op n۱ n۲ op args
```

این دستور برای تعیین برخی مشخصه‌های خاص مانند: جهت لینک در هنگام نمایش در nam (نرم‌افزاری که به صورت انیمیشن نتایج حاصل از شبیه‌سازی شبکه را نشان می‌دهد)، رنگ بسته‌های ارسالی به لینک و یا سایر مشخصه‌های مربوط به ترافیک‌های ارسالی بین دو نود n۱ و n۲ به کار می‌رود.

```
$NS duplex-link-op n۱ n۲ op args
```

مشابه دستور قبلی می‌باشد، با این تفاوت که برای لینک‌های دو طرفه به کار می‌رود.

```
$link cost cost-value
```

مقدار ارزش لینک را برابر با cost-value قرار می‌دهد.

در حالت پیش‌فرض تمام لینک‌ها دارای ارزش ۱ هستند.

$link cost?

مقدار ارزش عددی لینک را برمی‌گرداند.

$link up

لینک را به وضعیت فعال می‌برد.

$link down

لینک را به وضعیت غیرفعال می‌برد.

$link up?

وضعیت فعلی لینک را نمایش می‌دهد.

۲-۳-۵ مدیریت صف و انواع زمان‌بندی آن

شبیه‌ساز ns توانایی پشتیبانی از شش نوع مکانیسم صف‌بندی را دارد که عبارتند از: RED, FIFO, SFQ, FQ, CBQ و DRR.

مشخصه‌های پیکره‌بندی صف‌ها در شبیه‌سازی ns عبارتند از:

- limit : میزان گنجایش صف بر حسب بسته،
- blocked : در حالت پیش‌فرض، مقدار این متغیر بولی غلط می‌باشد؛ ولی چنانچه صف بلوکه گردد (یعنی که صف قادر به ارسال هیچ بسته‌ای به همسایه‌های مجاور خود نباشد)، مقدار متغیر صحیح می‌گردد.
- unblock-on-resume : به‌طور پیش‌فرض مقدار آن صحیح است و نشان‌دهنده آن است که در انتهای انتقال آخرین بسته، صف باید خود را غیربلوکه نماید.

مهم‌ترین دستورات مدیریت صف و زمان‌بندی آن در ns عبارتند از:

$NS queue-limit n۱ n۲ limit

حداکثر اندازه بافر صف لینک بین نودهای n۱ و n۲ را برابر با limit قرار می‌دهد.

$NS trace-queue n۱ n۲ file_name

کلیه رخدادهای صف بین نودهای n۱ و n۲ را ردیابی کرده و در فایل با نام file_name قرار می‌دهد.

$NS namtrace-queue n۱ n۲ file_name

کلیه رخدادهای صف بین نودهای n۱ و n۲ را برای استفاده برنامه انیمیشن nam ردیابی کرده و در فایلی با نام file_name قرار می‌دهد.

$NS namtrace-queue n۱ n۲ file_name

۲-۳-۶ مونیتور کردن صف

برای جمع‌آوری اطلاعات آماری مانند: تعداد بسته‌ها و بایت‌های ورودی به صف و طول متوسط صف، از دستورات زیر استفاده می‌شود:

```
$NS monitor-queue n۱ n۲ qtrace sampleinterval
```

مقدار پیش‌فرض متغیر Sampleinterval برابر با ۰،۱ می‌باشد. در مورد نحوه مونیتور کردن صف و جمع‌آوری اطلاعات آماری، در قسمت‌های بعد بیشتر توضیح داده خواهد شد.

۲-۳-۷ عاملان در NS

عاملان شبکه در حقیقت نقاط انتهایی یک اتصال را در سطح لایه شبکه بیان می‌کنند و بسته‌های ارسالی لایه شبکه از این عامل تولید می‌شود و در طرف گیرنده نیز بسته‌های لایه شبکه تحویل عامل متناظر می‌شوند. در شبیه‌ساز NS، هر عامل به صورت یک کلاس C++ طراحی می‌شود که دارای متغیرهای زیر می‌باشد.

- addr : آدرس نود مبدأ
- dst : آدرس نود مقصد
- size : طول بسته‌های ارسالی برحسب بایت
- type : نوع بسته‌های ارسالی
- fid : مشخص‌کننده اولویت بسته‌های IP
- prio : فیلد مشخص‌کننده اولویت بسته‌های IP
- flags : فیلد پرچم بسته‌ها
- defttl : مقدار پیش‌فرض فیلد TTL بسته‌های IP

شبیه‌ساز NS عاملان متعددی را پشتیبانی می‌کند که عبارتند از:

- TCP : عامل فرستنده مطابق با پروتکل Tahoe,TCP
- TCP/Reno : عامل فرستنده مطابق با پروتکل Reno TCP
- TCP/New Reno : عامل فرستنده و بهبود یافته پروتکل Reno, TCP
- TCP/Sack۱ : عامل فرستنده SACK TCP
- TCP/Full TCP : عامل فرستنده مطابق با استاندارد TCP کامل از نوع دو طرفه
- TCP/Vegas : عامل فرستنده TCP از نوع Vagas
- TCP/Begas/RBP : عامل فرستنده از نوع Vegas TCP به همراه Rate Based Pacing
- TCP/Asym : یک عامل فرستنده از نوع Tahoe TCP برای لینک‌های نامتقارن
- TCP/Newreno/Asym : یک عامل فرستنده از نوع TCP NewReno برای لینک‌های نامتقارن
- TCP/Sink : یک عامل گیرنده از نوع Reno TCP یا Tahoe TCP
- TCPSink/Delack : یک گیرنده TCP به همراه پیام گواهی

- TCPSink/Asym : یک عامل گیرنده TCP برای لینک‌های نامتقارن
- TCPSink/Sack۱ :یک گیرنده از نوع SACK TCP
- TCPSink/Sack۱/DelAck: یک عامل گیرنده از نوع SACKTCP به همراه پیام گواهی
- UDP : یک عامل پایه UDP
- RTP : یک فرستنده و گیرنده RTP
- RTCP : یک فرستنده گیرنده RTCP
- LossMonitor : یک عامل دریافت کننده به همراه قابلیت مونیتور نمودن میزان اتلاف
- IVS/Source : یک فرستنده از نوع IVS
- IVS/Receiver : یک گیرنده از نوع IVS
- SRM : یک عامل از نوع SRM به همراه زمان‌سنج‌های غیر وفقی
- SRM/Adaptive : یک عامل از نوع SRM به همراه زمان‌سنج‌های وفقی
- Tap : واسط شبیه‌ساز به دنیای شبکه واقعی
- Null : یک عامل دریافت و حذف کننده بسته‌های ورودی
- RtProto/DV : یک عامل مسیریابی از نوع بردار فاصله

به عنوان مثال برای ایجاد یک عامل TCP با طول پنجره ارسال برابر با ۲۰، از دستورات زیر استفاده می‌شود:

در مثال فوق ابتدا یک عامل TCP به نام newtep ایجاد می‌شود و سپس مقدار طول پنجره عامل فوق برابر با ۲۰ مقداردهی می‌شود.

```
set newtcp[new Agent/TCP]
$newtcp set window_ ۲۰
```

مثال ۲-۱: برنامه OTcl زیر یک عامل TCP ایجاد کرده و آن را فعال می‌سازد.

```
set tcp [new Agent/TCP]
$tcp set fid ۲
set sink [new Agent/TCPSink]
$NS attach-agent $n۰ $tcp
$NS attach-agent $n۳ $sink
$NS connect $tcp $sink
set ftp[new Application/FTP]
$ftp attach-agent $tcp
$NS at ۱ ,۲ "$ftp start"
```

در برنامه OTcl فوق، ابتدا یک عامل از نوع TCP با نام Tep ایجاد می‌شود. سپس مشخصه جریان (fid) برابر با ۲ برای جداسازی ترافیک‌های آن از سایر ترافیک‌ها (به خصوص هنگامی که از برنامه انیمیشن nam استفاده می‌شود) به آن نسبت داده می‌شود. در خط سوم برنامه یک عامل گیرنده از نوع TCPSink و با نام sink ایجاد می‌شود. در خطهای چهارم و پنجم برنامه فوق عامل‌های tcp و sink به

ترتیب به نودهای n۰ و n۳ متصل می‌شوند. البته باید توجه نمود که قبلاً لازم است که نودهای فوق با کمک دستور node ایجاد شده باشند.

بعد از اتصال عامل‌های TCP به نود مربوطه، با کمک دستور Connect، دو عامل tcp و sink به یکدیگر متصل می‌شوند. در خطوط ۷ و ۸ برنامه فوق، یک برنامه کاربردی از نوع FTP و با نام ftp ایجاد شده و به عامل tcp متصل می‌گردد. در انتها برنامه کاربردی ftp، در زمان ۲،۱ شروع به کار نموده و ترافیک ایجاد می‌نماید.

دستورات NS مربوط به ایجاد عامل‌ها و اتصال آنها به نودها و منابع ترافیکی به شرح زیر می‌باشد:

```
set agent[new Agent/AgentType]
```

این دستور یک عامل از نوع AgentType با نام agent ایجاد می‌نماید.

```
$NS attach-agent node agent
```

عامل agent را به نود node متصل می‌نماید. البته لازم است که قبل از این دستور عامل agent و نود node با کمک دستورات مربوطه ایجاد شده باشند.

```
$agent port
```

شماره درگاهی را که عامل agent به آن درگاه متصل است را برمی‌گرداند.

```
$agent dst-port
```

شماره درگاه عامل متناظر در مقصد، که عامل جاری به آن متصل است را برمی‌گرداند. هنگام اتصال عامل‌های متصل در دو نود به یکدیگر، هر عاملی شماره درگاه مقصد خود را در متغیر dstport ذخیره می‌سازد.

```
$agent attach-app s_type
```

این دستور برنامه کاربردی s_type را به عامل agent متصل می‌نماید.

```
$agent attach-source s_type
```

برای اتصال منبع ترافیکی s_type به عامل agent به کار می‌رود.

```
$NS connect src dst
```

عاملان src و dst (به ترتیب عاملان مبدأ و مقصد) به یکدیگر متصل می‌شوند.

```
$NS creat-connection srctype src dsttype dst pktclass
```

برای برقراری یک اتصال کامل بین دو عامل src و dst استفاده می‌شود. با کمک دستور فوق عامل‌های src و dst به ترتیب از نوع srctype و dsttype ایجاد می‌شوند و سپس به یکدیگر متصل می‌گردند. اشاره‌گری به عامل مبدأ برگشت داده می‌شود.

```
$NS creat-connection-list  srctype src dsttype dst pktclass
```

این دستور مشابه دستور creat-connection می‌باشد، با این تفاوت که به جای برگشت دادن یک اشاره‌گر به عامل مبدأ، لیستی از اشاره‌گرها به عاملان مبدأ و مقصد برگشت داده می‌شود.

۲-۳-۷-۱ عاملان UDP

عامل UDP بسته‌های اطلاعاتی با طول متغیر را از لایه بالاتر (لایه کاربردی) دریافت می‌دارد و در صورت لزوم آنها را به بسته‌های کوچکتر تقسیم می‌نماید. حداکثر طول بسته‌های UDP، به طور پیش‌فرض برابر با ۱۰۰۰ بایت می‌باشد، ولی در صورت لزوم می‌توان طول بسته‌ها را تغییر داد.

مثال۲-۲: در زیر برنامه OTcl برای ایجاد عاملان UDP و نحوه استفاده از آنها آورده شده است.

```
set NS [new Simulator]
set n۰ [$NS node]
set n۱ [$NS node]
$NS duplex-link $n۰ $n۱ ۵Mb ۲ms DropTail
set udp۰ [new Agent/UDP]
$NS attach-agent $n۰ $udp۰
set cbr۰ [new Application/Traffic/CBR]
$cbr۰ attach-agent $udp۰
$udp۰ set packetSize ۵۳۶
set null۰ [new Agent/Null]
$NS attach-agent $n۱ $null۰
$NS connect $udp۰ $null۰
$NS at ۱.۰ "$cbr۰ start"
```

در مثال فوق، ابتدا دو نود n۰ و n۱ ایجادمی‌شوند و سپس یک لینک دو طرفه با سرعت ۵ مگابیت بر ثانیه، تأخیر ۲ میلی ثانیه و از نوع DropTail(FIFO) ، بین دو نود n۰ و n۱ به وجود می‌آید. یک عامل UDP با نام udp۰ در نود n۰ و یک منبع ترافیک از نوع CBR (نرخ ارسال ثابت) و با نام cbr۰ ایجاد می‌شوند و سپس به یکدیگر متصل می‌گردند.

مقدار مشخصه packetSize بسته‌های UDP برابر با ۵۳۶ قرار داده شده است. یک عامل دریافت از نوع Null و با نام null۰ تعریف شده و به نود n۱ متصل شده است و به دنبال آن عامل udp۰ به عامل null۰ متصل شده است. بدین ترتیب عامل ارسال udp۰ در نود n۰ فرستنده می‌باشد و عامل null۰ در نود n۱ به عنوان گیرنده بسته‌های UDP عمل می‌کند. در زمان ۱ منبع ترافیکی cbr۰ که به عامل ارسال udp۰ در نود n۰ متصل است. شروع به ارسال بسته‌های ترافیکی به مقصد عامل null۰ متصل به نود n۱ می‌نماید.

دستورات مربوط به ایجاد و اتصال عاملان UDP به شرح زیر می‌باشد:

```
set udp۰ [new Agent/UDP]
```

این دستور، یک عامل UDP با نام udp۰ ایجاد می‌کند.

```
$NS attach-agent node agent
```

عامل agent را به نود node متصل می‌کند.

```
$udp set packetSize pktsize
```

مشخصه طول بسته‌های UDP را (مشخصه packetsize) را به مقدار pktsize قرار می‌دهد.

```
$udp set dst-addr address
```

مشخصه آدرس مقصد بسته‌های UDP (مشخصه dst_addr) را برابر با address قرار می‌دهد.

```
$udp set dst_port portnum
```

مشخصه شماره درگاه مقصد بسته‌های UDP (مشخصه dst_port) را برابر با portnum قرار می‌دهد.

```
$udp set ttl time_to_live
```

۲-۳-۷-۲ عامل‌های TCP

به طور کلی دو نوع عامل TCP در شبیه‌ساز NS موجود می‌باشد. این دو عامل عبارتند از:

۱. عامل‌های یک طرفه

۲. عامل‌های دو طرفه

در عامل‌های دو طرفه، فرستنده و گیرنده هر دو عامل‌های یکسان می‌باشند. این نوع عامل‌ها در NS هنوز تحت بررسی و توسعه می‌باشند. عامل‌های یک طرفه به طور مجزا برای فرستنده و گیرنده تعریف می‌شوند.

عامل‌های یک طرفه فرستنده TCP عبارتند از:

Agent / TCP, Agent / TCP / Remo, Agent / TCP / NewReno, Agent / TCP / Sack۱, Agent / TCP / Vegas, Agent / TCP / Fack.

عامل‌های یک طرفه گیرنده TCP، که در حال حاضر در شبیه‌ساز NS پشتیبانی می‌شوند، عبارتند از:

Agent / TCPSink, Agent / TCPSink/DelAck, Agent / TCPSink/Sack۱, Agent / TCPSink / Sack۱ / DelAck

در شبیه‌ساز NS در حال حاضر تنها عامل دو طرفه Agent/TCP/FullTCP پشتیبانی می‌شود.

۲-۳-۷-۲-۱ عامل‌های یک طرفه فرستنده TCP

در نرم‌افزار شبیه‌ساز NS عامل‌های یک طرفه فرستنده TCP زیر موجود است:

Tahoe TCP,Renoe TCP, NewRenoe TCP, Vegas TCP, Sack TCP, Fack TCP

مثال ۲-۳: برنامه شبیه‌ساز زیر را در نظر بگیرید:

```
set NS [new Simulator]
set tcp۱ [$NS creat-connection TCP $node۱ TCPSink $node۲ ٤۲]
$tcp۱ set window ٥٠
set ftp۱ [new Application/FTP]
$ftp۱ attach-agent $tcp۱
$NS at ٠,٠ "ftp start"
```

در برنامه فوق، با کمک دستور creat-connection یک عامل ارسال TCP با نام tcp۱ در نود node۱ و یک عامل دریافت TCPSink در نود node۲ به وجود می‌آیند و به یکدیگر متصل می‌گردند. همچنین از مشخصه جریان ۴۲ برای مشخص کردن ترافیک‌های اتصال فوق استفاده شده است. مشخصه window (طول پنجره) عامل tcp۱ برابر با ۵۰ قرار داده شده است. یک برنامه کاربردی از نوع FTP و با نام ftp۱ ایجاد شده است و به عامل tcp۱ متصل گردیده است. برنامه کاربردی ftp۱ در زمان صفر شروع به کار می‌نماید.

با توجه به مثال فوق دیده می‌شود که عامل TCP خود به تنهایی تولید ترافیک نمی‌نماید، بلکه حتماً باید یک برنامه کاربردی مانند FTP و یا TELNET و یا یک منبع ترافیکی به آن متصل شود.

۲-۳-۷-۲-۲ عامل‌های یک طرفه گیرنده TCP

عامل یک طرفه فرستنده TCP که در بالا توصیف شد، اقدام به ارسال بسته‌های TCP می‌کند. در طرف گیرنده باید از عامل متناظر TCP استفاده نمود. یکی از متداولترین عامل‌های یک طرفه گیرنده TCP، عامل TCPSink می‌باشد. این عامل با دریافت هر بسته TCP یک بسته گواهی ارسال می‌دارد که به صورت زیر می‌توان طول بسته گواهی را تنظیم کرد.

```
Agent/TCPSink set packetSize ٤٠
```

در دستور فوق طول پیام گواهی همه عامل‌های گیرنده TCPSink برابر با ۴۰ بایت قرار داده شده است.

در عامل دریافت Delayed-Ack TCPSink، که یک نوع از عامل‌های دریافت TCP می‌باشد، برای هر بسته TCP یک پیام گواهی صادر نمی‌شود بلکه در فواصل قابل تنظیمی پیام‌های گواهی فرستاده می‌شوند. البته چنانچه یک بسته TCP خارج از ترتیب دریافت گردد، در این صورت سریعاً پیام گواهی به سرعت فرستاده می‌شود. فاصله زمانی ارسال پیام‌های گواهی به صورت زیر قابل تنظیم می‌باشد:

```
Agent/TCPSink/DelAck set interval ١٠٠ms
```

دستور فوق باعث می‌شود که فاصله زمانی بین ارسال پیام‌های گواهی برای تمام عامل‌های از نوع TCPSink/DelAck، برابر با ۱۰۰ میلی ثانیه گردد.

در عامل دریافت Sack TCPSink که یک نوع دیگر عامل دریافت TCP است، ارسال پیام‌های گواهی به صورت انتخابی صورت می‌گیرد.

۲-۳-۶-۲-۳- عامل دو طرفه FullTCP

این عامل جدیداً به شبیه‌ساز NS اضافه شده است و هنوز تحت توسعه می‌باشد. تفاوت عامل فوق با سایر عامل‌های TCP به شرح زیر می‌باشد:

الف) برخلاف عامل‌های ارسال TCP دیگر، در این عامل از طریق ارسال بسته‌های SYN/FIN امکان پایه‌گذاری و قطع اتصال وجود دارد.

ب) دراین عامل ارسال داده‌ها به صورت دو طرفه پشتیبانی می‌شود.

ج) شماره بسته‌ها بر حسب بایت می‌باشد نه بر حسب تعداد بسته‌های ارسالی.

بنابراین با توصیفات فوق دیده می‌شود که عامل دو طرفه FullTCP نسبت به سایر عامل‌های TCP موجود در شبیه‌ساز NS به مراتب نزدیکتر به واقعیت می‌باشد.

مثال ۲-۴: در زیر یک مثال در مورد نحوهٔ ایجاد اتصال‌های FullTCP آورده شده است.

```
set src[new Agent/TCP/FullTCP]
set sink [new Agent/TCP/FullTCP]
$NS attach-agent $node۱ $src
$NS attach-agent $node۲ $sink
$src set fid ۰
$sink set fid ۰
$NS connect $src $sink
$sink listen
$src set window ۱۰۰
```

در برنامه فوق دو عامل TCP از نوع FullTCP و با نام‌های src و sink به وجود می‌آیند. به دنبال آن عامل‌های فوق به نودهای node۱ و node۲ اتصال می‌یابند. مقدار مشخصه جریان (fid) عامل‌های فوق برابر با صفر مقداردهی شده است.

با کمک دستور connect عامل‌های src و sink به یکدیگر اتصال یافته‌اند، سپس عامل sink به وضعیت listen رفته و منتظر دریافت بسته از عامل src می‌گردد. همچنین طول پنجره ارسال عامل src برابر با ۱۰۰ مقداردهی شده است.

۲-۳-۷-۲ عامل‌های SRM

جهت پیاده‌سازی پروتکل‌های چند پراکنی از عامل‌های SRM استفاده می‌شود. عملیات لازم برای این کار در سه مرحله زیر انجام می‌شود:

- مرحله ۱: ایجاد و پیکره‌بندی عامل SRM،
- مرحله ۲: اتصال منابع ترافیکی،
- مرحله ۳: شروع به کار عامل و منابع ترافیکی.

جهت ایجاد عامل SRM از دستور زیر استفاده می‌شود:

```
set srm [new Agent/SRM]
```

با کمک دستور فوق، یک عامل SRM با نام srm ایجاد می‌شود.

مثال ۲-۵: برنامه شبیه‌ساز زیر برای ایجاد عامل‌های SRM و استفاده از آن به کار می‌رود.

```
set NS [new Simulator]
$NS enableMcast
set node [$NS node]
set group [$NS allocaddr]
set srm [new Agent/SRM]
$srm set dst $grup
$NS attach-agent $node $srm
$srm set fid ۱
$srm log [open srmStats.tr w]
$srm trace [open srmEvents.tr w]
```

در مثال فوق، ابتدا یک نود به نام node و با قابلیت چند پراکنی ایجاد می‌شود (با فعال ساختن متغیر enable Mcast) و سپس یک گروه چند پراکنی به نام group تعریف می‌گردد. به دنبال آن یک عامل SRM با نام srm تعریف شده است و به نود node اتصال یافته است. گروه group که در بالا تولید شد، به عامل srm اختصاص یافته است. از مشخصه جریان (fid) برابر با ۱ برای ترافیک‌های SRM استفاده شده است. همچنین دو فایل به نام‌های srmStats.tr و srmEvents.tr برای ثبت اطلاعات آماری و ردیابی داده‌ها باز شده‌اند.

البته باید توجه نمود که عامل SRM به تنهایی هیچ‌گونه داده کاربردی ایجاد نمی‌نماید بلکه باید منابع ترافیکی موردنظر ایجاد شده و به عامل SRM متصل گردند.

مثال ۲-۶: برنامه شبیه‌سازی زیر را در نظر بگیرید:

```
set packetSize ۲۱۰
set exp۰ [new Application/Traffic/Exponential]
$exp۰ set packetSize $packetSize
$exp۰ set burst_time ۵۰۰ms
$exp۰ set idle_time ۵۰۰ms
$exp۰ set rate ۱۰۰k
$exp۰ attach-agent $srm
$srm set packetSize $packetSize
$srm set tag $exp۰
$srm set app fid ۰
```

در مثال فوق که در ادامه مثال قبل می‌باشد: ابتدا یک منبع ترافیکی از نوع نمایی با مشخصه‌های: طول بسته‌های ارسالی مساوی با ۲۱۰، میانگین ناحیه انفجار برابر با ۵۰۰ میلی‌ثانیه، میانگین طول ناحیه سکوت برابر با ۵۰۰ میلی‌ثانیه و حداکثر نرخ ارسال برابر با ۱۰۰ کیلوبیت بر ثانیه ایجاد شده است و سپس به عامل srm متصل گردیده است. مشخصه packetSize عامل srm برابر با ۲۱۰ مقداردهی شده است. همچنین مولد ترافیکی عامل srm منبع ترافیکی exp۰ قرار داده شده است. همچنین از مشخصه جریان برابر با صفر برای مشخص کردن بسته‌های تولیدی منبع ترافیکی exp۰ متصل به عامل srm استفاده شده است.

البته امکان اتصال هر نوع منبع ترافیکی به عامل SRM موجود می‌باشد. هنگامی که عامل SRM بسته‌های ترافیکی منبع ترافیکی را دریافت کرد، سرآیند مربوط به خود را به بسته‌های دریافتی اضافه کند و آدرس مقصد بسته‌ها را برابر با آدرس Multicast group که قبلاً تعریف شده است، قرار می‌دهد و آنها را به سمت مقصد می‌فرستد. در سرآیند بسته‌های SRM اطلاعاتی مانند: نوع پیام، مشخص کننده فرستنده پیام، شماره بسته ارسالی، موجود می‌باشد.

برای شروع به کار عامل SRM و همچنین منبع ترافیکی متصل به آن، از دستور start به صورت زیر استفاده می‌شود:

```
$srm start
$exp۰ start
```

البته می‌توان به جای دو دستور فوق که عامل srm و منبع ترافیکی exp۰ را فعال می‌سازد، یک دستور واحد زیر را استفاده نمود:

```
$srm start-source
```

۲-۴ لایه کاربرد و عامل ارسال API

در شبیه‌ساز NS، لایه کاربرد در بالای عامل‌های ارسال قرار می‌گیرد. در NS دو نوع لایه کاربرد وجود دارند که عبارتند از:

الف) منابع ترافیکی

ب) پروتکل‌های شبیه سازی شده لایه کاربرد

برنامه‌های کاربردی، برای دستیابی به سرویس‌های شبکه و لایه‌های پایین‌تر، از واسط برنامه‌نویسی کاربردی API[1] استفاده می‌نمایند. به عنوان مثال، سوکت‌ها نوع متداولی از API می‌باشد.

مراحل اتصال لایه کاربرد به عامل‌های ارسال و ارتباط آنها از طریق API به صورت زیر می‌باشد:

مرحله ۱: ایجاد عامل‌های ارسال و اتصال آنها به نودهای شبکه

در اولین مرحله، عامل‌های ارسال، ایجاد شده و به نودهای شبکه متصل می‌گردند. مثال زیر این مرحله را نشان می‌دهد.

```
set src [new Agent/TCP/FullTCP]
set sink [new Agent/TCP/FullTCP]
$NS attach-agent $node۱ $src
$NS attach-agent $node۲ $sink
$NS connect $src $sink
```

در این مثال ابتدا دو عامل فرستنده و گیرنده از نوع FullTCP، به نام‌های src و sink ایجاد شده‌اند و سپس به نودهای node۱ و node۲ که قبلاً ایجاد شده اند، متصل شده‌اند. سپس عامل‌های ارسال src و دریافت sink به کمک دستور connect به یکدیگر اتصال یافته‌اند.

---

1. Application Programming Interface

مرحله ۲: اتصال لایه کاربرد به عامل‌های ارسال

بعد از ایجاد لایه کاربرد، با کمک دستور attach-agent، اتصال عامل ارسال با لایه کاربرد برقرار می‌گردد. مثلاً برای ایجاد یک لایه کاربرد FTP با نام ftp۱ و اتصال آن به عامل ارسال src، دستورات زیر استفاده می‌شوند:

```
set ftp۱ [new Application/FTP]
$ftp۱ attach-agent $src
```

مرحله ۳: استفاده از عامل‌های ارسال از طریق فراخوانی سیستم

بعد از ایجاد عامل‌های ارسال و لایه کاربرد و اتصال آنها به یکدیگر، لایه کاربرد با کمک چندین فراخوانی سیستم قادر به استفاده از سرویس‌های عامل ارسال می‌گردد. این فراخوانی‌های سیستم عبارتند از:

- send (int nbytes)

ارسال nbytes داده به طرف مقابل.

- sendmsg (int nbytes, coNSt char *flags)

مشابه دستور send می‌باشد که علاوه بر داده‌ها پرچم‌های خاصی نیز فرستاده می‌شود.

- close ( )

درخواست از عامل برای قطع اتصال (فقط برای عامل‌های TCP استفاده می‌شود).

- listen ( )

درخواست از عامل برای گوش دادن به ورود درخواست‌های جدید (فقط برای عامل FullTCP استفاده می‌شود).

- set pkttype(int pkttype)

این دستور متغیر type عامل را برابر با pkttype قرار می‌دهد.

مرحله ۴: Upcalls کردن عامل‌ها به لایه کاربرد

از آنجایی که در شبیه‌ساز NS هیچگونه داده حقیقی مبادله نمی‌شود، عامل‌ها برای مطلع ساختن لایه کاربرد از وقوع یک رخداد، از upcalls استفاده می‌نمایند. در شبیه‌ساز NS از upcallsهای زیر استفاده می‌شود:

- recv (int nbytes)

اعلام کند که nbytes داده توسط عامل ارسال دریافت شده است.

- resume( )

عامل ارسال به لایه کاربرد خبر می‌دهد که تاکنون تمام داده‌هایی را که برای آن فرستاده است، از خود خارج نموده است.

مثال ۲-۷: مثال زیر نحوه استفاده از API برای ایجاد یک لایه کاربرد FTP بالای یک اتصال از نوع FullTCP را نشان می‌دهد.

```
set src [new Agent/TCP/FullTCP]
set sink [new Agent/TCP/FullTCP]
$NS attach-agent $node۱ $src
$NS attach-agent $node۲ $sink
$NS connect $src $sink
$sink listen
$src set window ۱۰۰
set ftp۱ [new application/FTP]
$ftp۱ attach-agent $src
$NS at ۰,۰ "ftp۱ start"
```

در مثال فوق ابتدا دو عامل FullTCP، به نام‌های src و sink ایجاد شده‌اند و به نودهای node۱ و node۲ اتصال یافته‌اند، عامل sink به وضعیت listen می‌رود و منتظر دریافت اطلاعات می‌گردد. طول پنجرهٔ ارسال عامل src برابر با ۱۰۰ قرار داده شده است. یک برنامه کاربردی از نوع FTP و با نام ftp۱ ایجاد شده است.

۲-۵ دستورات NS برای ایجاد و کنترل انیمیشن‌های nam

برای ایجاد و کنترل انیمشن‌های nam از دستورات زیر استفاده می‌شود.

۲-۵-۱ دستورات مربوط به نود

```
$node color <color>
```

این دستور رنگ نود را هنگام نمایش به وسیله nam، به رنگ نشان داده شده color قرار می‌دهد.

```
$node shape <shape>
```

این دستور شکل نود را هنگام نمایش بر روی صفحه نمایش nam مشخص می‌کند. شکل نود یکی از سه صورت دایره، مربع و هشت‌ضلعی می‌باشد.

```
$node label <label>
```

این دستور برای نود node بر چسب label را قرار می‌دهد.

```
$node label-color <lcolor>
```

به وسیله این دستور، می‌توان رنگ برچسب نود node را برابر با lcolor قرار داد.

$node label-at <ldirection>

با کمک این دستور می‌توان برچسب نود node را در مکان ldirection در صفحه نمایش nam قرار داد.

۲-۵-۲ دستور مربوط به لینک و صف

$NS duplex-link-op <attribute> <value>

از این دستور برای تعیین مشخصه‌های لینک استفاده می‌شود. مشخصه attribute یکی از مقادیر orient,queuePos:color می‌تواند باشد. Orient مشخص کنندهٔ زاویه بین لینک و خط افق، هنگام نمایش آن برروی صفحه nam می‌باشد. مقادیر Orient به صورت درجه و یا به صورت متنی به صورت زیر قابل بیان هستند.

right (زاویه صفر درجه)

right-up (زاویه ۴۵+ درجه)

right-down (زاویه ۴۵ - درجه)

left (زاویه ۱۸۰+ درجه)

left-up (زاویه ۱۳۵+ درجه)

left-down (زاویه ۱۳۵- درجه)

up (زاویه ۹۰+ درجه)

down (زاویه ۹۰- درجه)

مشخصه queuePos، محل قرار گرفتن صف را برحسب زاویه آن با خط افق برروی صفحه نمایش nam مشخص می‌نماید.

۲-۶ چند مثال شبیه‌سازی در NS

همان‌طور که قبلاً نیز گفته شد، برنامه‌های شبیه‌سازی در NS، در یک فایل telscript با پیوند tcl نوشته می‌شوند و سپس برای اجرای آن از دستور زیر استفاده می‌شود.

NS <tclscript>

البته باید قبلاً مسیری که فایل telscript آنجا قرار دارد در Path آورده شده باشد. می‌توان بدون آن که دستورات شبیه‌سازی را در فایل telscript نوشت، در محیط tel shell دستورات را یکی یکی وارد نمود، که روش چندان مناسبی نمی باشد. در این بخش از کتاب به ذکر چند مثال و توصیف عملکرد هر یک می‌پردازیم.

۲-۶-۱ ایجاد توپولوژی شبکه

در این مثال به طور خیلی ساده و ابتدایی به نحوه ایجاد توپولوژی شبکه شامل نودها، لینک‌ها و نحوه مونیتور نمودن صف‌ها و مشاهده آن در nam می‌پردازیم. اولین قدم، ایجاد یک object از شبیه‌سازی می‌باشد که این امر به صورت زیر انجام می‌شود:

```
set NS [new Simulator]
```

جهت ثبت و ذخیره‌سازی نتایج حاصل از شبیه‌سازی و ردیابی نمودن صف و استفاده از آن در nam، فایل out nam به صورت زیر باز می‌شود.

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

مطابق با دستورات فوق، فایل out.nam جهت نوشتن اطلاعات شبیه‌سازی باز می‌شود و اشاره‌گر فایل nf به آن اختصاص می‌یابد و در مدت شبیه‌سازی، تمام داده‌های شبیه‌سازی که به نحوی به nam وابسته می‌باشند در این فایل ذخیره می‌شوند.

در مرحله بعدی یک روال پایان به نام finish ایجاد می‌شود، که هنگام اتمام شبیه‌سازی دستورات موجود در این روال اجرا می‌شود. روال فوق به صورت زیر می‌باشد:

```
proc finish()
        {
    global NS nf
    NS flush-trace
        close $nf
exec nam out.nam &
exit
        }
```

در لحظه‌های خاتمه شبیه‌سازی، روال فوق اجرا می‌شود که باعث می‌شود تمام داده‌های شبیه‌سازی مربوط به nam در فایل out-nam که دارای اشاره‌گر nf می‌باشد، ذخیره شده و سپس فایل فوق بسته می‌شود به دنبال آن nam اجرا می‌شود و نتایج شبیه‌سازی که در فایل out.nam نوشته شده بود، به صورت انیمیشن نشان داده می‌شود.

زمان اجرای روال finish که در بالا آورده شده است، با کمک دستور at و به صورت زیر قابل تنظیم است:

```
$NS at ۵.۰ "finish"
```

دستور فوق موجب می‌شود که روال finish در زمان ۵.۰ اجرا گردد.

در نهایت برای اجرای شبیه‌سازی باید از دستور زیر استفاده کرد:

```
$Ns run
```

مطالبی که در بالا ذکر گردید، قسمت‌های اصلی تشکیل‌دهنده هر فایل tclscript برای انجام شبیه‌سازی می‌باشد. جهت ایجاد توپولوژی شبکه شامل: نودها، لینک‌ها اتصال‌دهنده نودها، عاملان

ارسال و اتصال آنها به نودها، باید از دستورات خاص NS استفاده کرد. که بهتر است که دستورات فوق، قبل از دستور "finish" ۵٫۰ $Ns at و بعد از توصیف روال finish آورده شود.

به عنوان مثال برای ایجاد یک توپولوژی شبکه شامل دو نود به نام‌های n۰ و n۱، از دستورات زیر استفاده می‌شود:

```
set n۰ [$NS node]
set n۱ [$NS node]
```

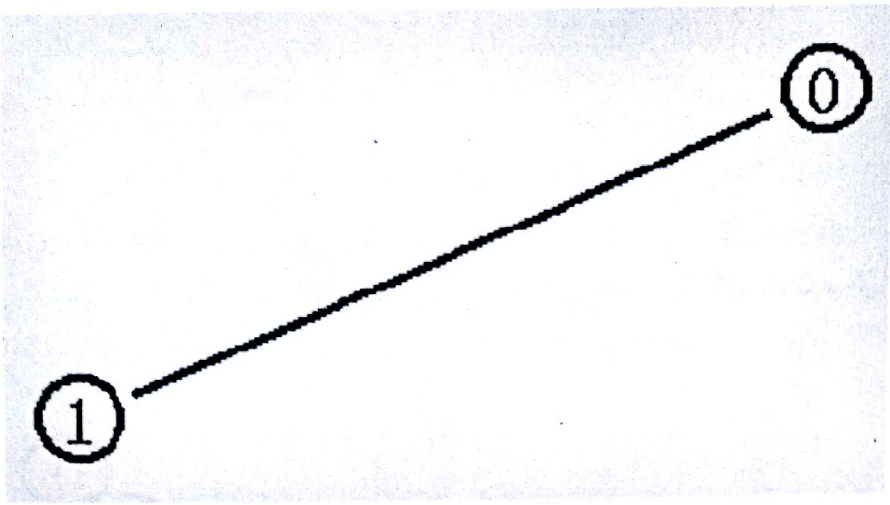
جهت اتصال دو نود فوق به یکدیگر، از طریق لینک دو طرفه با سرعت ۱Mb/s، تأخیر ۱۰ms میلی ثانیه و بافر از نوع FIFO ، از دستور زیر استفاده می‌شود:

```
$NS duplex-link $n۰ $n۱ ۱Mb ۱۰ms DropTail
```

در زیر کد کامل برنامه شبیه‌ساز فوق آورده شده است:

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

#Open the nam trace file
set nf [open out.nam w]
$NS namtrace-all $nf
#Define a 'finish' procedure
proc finish {} {
        global NS nf
        $NS flush-trace
         #Close the trace file
        close $nf
         #Execute nam on the trace file
        exec nam out.nam &
        exit ۰
}

#Create two nodes
set n۰ [$NS node]
set n۱ [$NS node]

#Create a duplex link between the nodes
$NS duplex-link $n۰ $n۱ ۱Mb ۱۰ms DropTail

#Call the finish procedure after ۵ seconds of simulation time
$NS at ۵٫۰ "finish"

#Run the simulation
$NS run
```

بعد از اجرای مثال فوق، چنانچه در محیط گرافیکی Xwindows باشیم، خروجی مطابق با شکل ۲-۶ در روی صفحه مونیتور مشاهده خواهد شد.



شکل ۲-۶: خروجی مثال ۲-۶-۱ در محیط گرافیکی Xwindows

۲-۶-۲ ایجاد عامل ارسال و اتصال آن به نودهای شبکه (بدون اتصال لینک‌ها)

در مثال ساده قبل، فقط دو نود شبکه ایجاد گردیدند و به کمک یک لینک به یکدیگر متصل شدند. در مثال فوق هیچگونه داده‌ای بین نودها ایجاد و ارسال نگردید. برای ارسال داده، ابتدا باید عاملان مناسب تعریف شوند و به نودهای شبکه متصل گردند. در این مثال به تکمیل مثال قبل می‌پردازیم.

جهت ایجاد عامل ارسال و اتصال آن به نودهای شبکه دستورات زیر به کار می‌رود:

```
set cbr۰ [new Agent/CBR]
$NS attach-agent $n۰ $cbr۰
$cbr۰ set packetSize ۵۰۰
$cbr۰ set interval ۰,۰۰۵
```

دستورات فوق، باعث ایجاد یک منبع ترافیکی CBR با نام cbr۰ (که به نود n۰ متصل است)، می‌شود. مشخصه طول بسته (packetsize) و فاصله زمانی بین بسته‌ها (interval)، به ترتیب برابر با ۵۰۰ بایت و ۰،۰۰۵ ثانیه (معادل سرعت ۲۰۰ بسته در ثانیه) قرار داده شده است.

جهت ایجاد یک عامل Null به نام null۰، که در حکم دریافت کننده ترافیک عمل می‌کند و اتصال آن به نود n۱ دستورات زیر استفاده می‌شود:

```
set null۰ [new Agent/Null]
$NS attach-agent $n۱ $null۰
```

بعد از ایجاد عاملان ارسال و دریافت، باید اتصال بین این دو عامل برقرار گردد. این کار به صورت زیر انجام می‌شود.

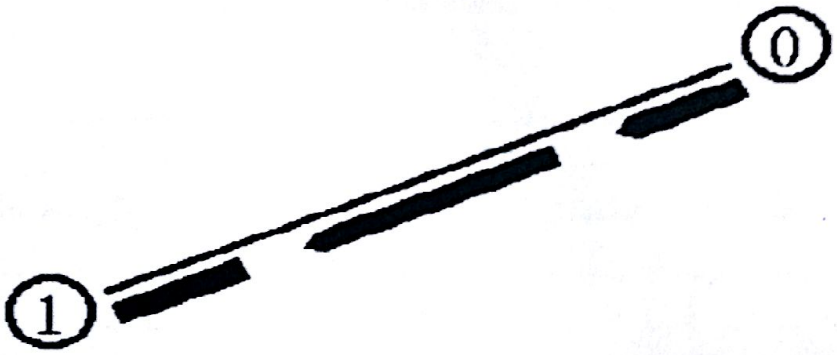
```
$NS connect $cbr۰ $null۰
```

جهت تعیین زمان شروع به کار و پایان ارسال منابع ترافیکی، از دستور at استفاده می‌شود. طبیعی است که زمان پایان ارسال منابع ترافیکی، باید قبل از زمان پایان شبیه‌سازی که قبلاً برابر با ۵،۰ در نظر

گرفته شده بود، باشد. دستورات زیر، زمان شروع و پایان منبع ترافیکی cbr۰ را به ترتیب برابر با ۰.۵ و ۴.۵ قرار می‌دهد.

```
$NS at ۰.۵ "cbr۰ start"
$NS at ۴.۵ "cbr۰ stop"
```

بعد از اجرای مثال فوق، محیط nam ظاهر می‌شود. چنانچه کلید play را فعال سازیم، در این صورت بعد از گذشت نیم ثانیه از آغاز شبیه‌سازی، نود n۰ شروع به ارسال بسته‌های اطلاعاتی به مقصد نود n۱ می‌کند که در nam به صورت شکل ۲-۷ قابل رؤیت می‌باشد.



شکل ۲-۷ : خروجی حاصل از مثال ۲-۶-۲