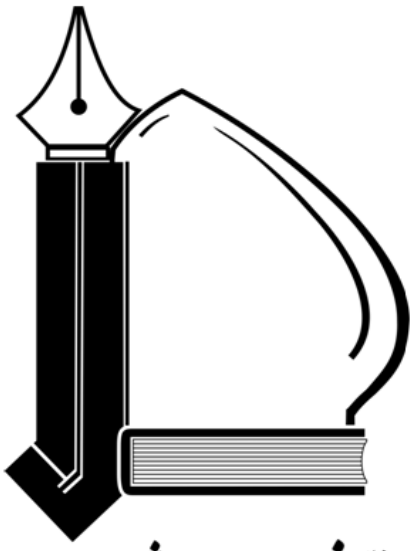


دانشگاه فردوسی مشهد

دانشکده مهندسی - گروه مهندسی کامپیوتر

جزوه آموزشی NS

(**N**etwork **S**imulator)

تهیه و تنظیم :

دکتر محمد حسین یغمائی مقدم

مهندس حسین کاری

فصل ۱

معرفی اجمالی NS:

شبیه‌ساز NS، یک نرم‌افزار جامع شبیه‌سازی شبکه‌های مخابراتی و رایانه‌ای با قابلیت پشتیبانی از پروتکل‌های مختلف شبکه می‌باشد. شبیه ساز فوق شاخه‌ای از پروژه REAL Network Simulator می‌باشد، که از سال ۱۹۸۹ آغاز شد و در طی چند سال اخیر تکمیل و توسعه یافته است. NS بر اساس تکنیک شبیه‌سازی رخدادگرا[۱] طراحی شده است. شبیه ساز NS تعداد زیادی از برنامه‌های کاربردی، پروتکل‌ها، انواع شبکه، اجزای شبکه و مدل‌های شبکه که آنها را اشیا شبیه‌سازی شده می‌نامیم، پوشش می‌دهد. این متن دو هدف را دنبال می‌کند: یادگیری استفاده از شبیه ساز NS ودیگری آشنایی و درک کارکرد برخی از اشیا شبیه‌سازی شده با استفاده از شبیه‌سازی‌های NS. این جزوه نه تنها برخی از اصول و توصیفات شبیه‌ساز NS را فراهم می‌نماید بلکه از اشیا شبیه‌سازی شده نیز برخی از موارد را توضیح می‌دهد.

این جزوه برای کمک به دانشجویان،مهندسین و محققین که پیش زمینه‌ای عمیق از برنامه نویسی ندارند ونیز به آنان که می‌خواهند از طریق مثال‌های ساده چگونگی بررسی برخی از اشیا شبیه‌سازی شده را متوجه شوند نگارش‌یافته است. بدین منظور تعداد زیادی از اسکریپت های به زبان tcl که می‌توانند توسط خوانندگان جهت شروع فوری برنامه‌نویسی استفاده شوند، فراهم شده است. البته برای خوانندگانی که علاقه‌مند هستند مثال‌های بیشتری می‌باشند در نظر داشته باشند که تعداد زیادی از این مثال‌ها هم اکنون در بسته‌های شبیه ساز NS موجود می‌باشد. سایر خودآموزهای الکترونیکی که مثال‌هایی در خود دارند : خودآموز Marc Greis ودیگری خوآموز Mark Claypool وJae Chung می‌باشند.

---

[۱] Event driven

برای یک مطالعه عمیق‌تر از شبیه‌ساز NS باید به کتاب راهنمای NS که بصورت به‌روز شده در www.isi.edu/nsnam/ns/ نگهداری می‌شود،مراجعه نمود.

ما در این کتاب بسیاری سناریوی ساده اما کاربردی را برای شبیه‌سازی ارائه می‌نماییم. شبیه‌سازی‌هایی که ممکن است در ابعاد زیادی با هم متفاوت باشند: برنامه کاربردی، ساختار شبکه،پارامترهایی اشیاء شبکه (لینکها ونودها) و پروتکل مورد استفاده و غیره. در ابتدای کار ما به دنبال جامعیت مثال‌ها نبوده و خود را درگیر جزئیات نمی‌کنیم در عوض آنچه را که ارائه می‌نماییم در قالب یک مثال عمومی در نظر می‌گیریم. اگر توصیف جزئیات را در مواردی از ns نیاز باشد می‌توانید آن را از راهنمای NS بخواهید. که می‌توانید به آدرس www.isi.edu/nsnam/ns/ns-documentation.html برای دسترسی به آن مراجعه نمایید.به عنوان یک راه جایگزین و ساده جهت اطلاع از سایر امکانات برای انتخاب اجزای شبکه، پروتکل‌های شبکه یا پارامترهای آنان و پارامترهای برنامه‌های کاربردی و سایر موارد می‌توانید مستقیماً به فایل‌های کتابخانه‌ای[1] که آنان را تعریف می‌نماید، بیندازید(در آدرس /ns2.x/tcl/lib…) به عنوان مثال تعریف نودهای موبایل را ممکن است در فایل ns-mobilenode.tcl پیدا کنید و یا توصیفات ترتیب صف‌بندی و پارامترها در فایل ns-queue.tcl می‌باشد. مقادیر پیش فرض پارامترها را می‌توانید در فایل ns-default.tcl بیابید به خاطر داشته باشید دانستن آنکه کدام شی پیش‌فرض مربوطه به کدام فرمان می‌باشد لازم است فایل ns-queue.tcl را چک‌نمایید. که البته در یک مثال در بخش ۲،۲ ان شاا... ملاحظه خواهید نمود.

۱-۱ : دور نما و پس زمینه شبیه ساز NS :

شبیه ساز NS مبتنی بر ۲ زبان است: یک زبان شی‌گرا که همان C++ می‌باشد یک مفسر OTCL که توسعه شی‌گرای TCL می‌باشد که دومی جهت اجرای اسکریپت دستورات کاربر استفاده می‌شود. NS دارای یک کتابخانه غنی از اشیا و شبکه و پروتکل‌ها می‌باشد. ۲ کلاس ارث‌بری و سلسله مراتبی در

[1] - Library

NS وجود دارد: سلسله مراتب کدهای کامپایل شده C++ و دیگری سلسله مراتب کدهای مفسری OTCL که یک تناظر یک‌به‌یک میان آن ۲ موجود می‌باشد. سلسله مراتب کامپایل شده به ما توانایی برخورداری از بازدهی و کارایی را در شبیه‌سازی داده و نیز امکان اجرای سریعتر شبیه‌سازی را فراهم می‌نماید. این وجه مخصوصاً از نظر تعاریف با جزئیات زیاد و عملکرد پروتکل‌ها مفید است. این ویژگی باعث کاهش زبان پردازش رخدادها و بسته‌ها می‌شود.

اما در اسکریپت OTCL ایجاد شده توسط کاربر می‌توان ساختار شبکه، پروتکل خاص مورد کاربرد را نیز می‌توان تعریف نمود. در مورد برنامه مورد کاربردی مورد استفاده نیز که روی آن پروتکل خاص به ارسال ترافیک می‌پردازد، نوع آن در اسکریپت کاربر مشخص می‌شود که به همراه نام فایل آن مشخص می‌گردد.

در طراحی شبیه ساز فوق، از زبان برنامه نویسی C++ استفاده گردیده است و همچنین از زبان otcl[1] نیز به‌عنوان واسط و مترجم فرامین استفاده می‌شود.

در شبیه ساز NS، از دو زبان C++ و otcl همزمان با هم استفاده می‌شود. به علت سرعت بالای C++ از آن برای پیاده سازی پروتکل‌ها و پردازش بسته‌های اطلاعاتی ورودی استفاده می‌شود. اما برای شبیه‌سازی ساختار و توپولوژی شبکه از زبان otcl استفاده می‌گردد.

OTCL از طریق یک پیوند[۲] می‌تواند از اشیا کامپایل شده C++ استفاده نماید که این کار از طریق tclcl انجام می‌پذیرد که یک مفسر TCL/C++ است و یک تطابقی میان اشیا OTCL و C++ را فراهم می‌نماید. NS یک شبیه‌ساز رویداد و رخدادها از نوع دقیق می‌باشد و پیش‌برد و جلو رفتن زمان بستگی به زمانبندی رویدادها که توسط زمانبند نگهداری می‌شود اتفاق می‌افتد. در سلسله مراتب C++ یک رویداد یک شی با شناسه یکتا می‌باشد که یک زمان تنظیم شده و یک اشاره‌گر به یک شی

---

[1] Object Tool Command Language

[2] - linkage

که با رویداد سروکار دارد را نیز همراه خواهد داشت. زمانبند یک ساختار داد مرتب را (که اغلب از لیست پیوسته استفاده می کند)

با رویدادهایی که باید اجرا شود را نگهداری می‌نماید و آن‌ها را یک به یک اجرا نموده و آن رویداد را در آن زمان فراخوانی می‌نماید.

محیط مورد نیاز و نحوه نصب NS:

شبیه‌ساز NS بر روی سیستم‌های عامل مختلف یونیکس مانند Free BSD UNIX, Linux, Sunos, Solaris قابل نصب می‌باشد. البته می‌توان از سیستم عامل ویندوز نیز استفاده نمود، ولی NS ذاتاً برای محیط‌های یونیکس طراحی شده است.

برای نصب NS کافی است که بعد از باز کردن فایل فشرده شده NS، در بالاترین شاخه دستورinstall/. اجرا گردد. این دستور باعث نصب NS برروی سیستم(در محیط یونیکس) می‌گردد. جهت آن‌که شبیه‌ساز NS به‌طور صحیح اجرا گردد، باید از Tcl نسخه v7.5 یا بالاتر استفاده شود. همچنین چنانچه از gcc به‌عنوان کامپایلر C++ استفاده می‌شود، باید از نسخه 2.7.2 یا بالاتر از آن استفاده نمود.

بعد از نصب NS، برای اطمینان از صحت عملیات نصب و تست کردن سیستم، می‌توان در بالاترین شاخه دستور.validate/ را اجرا نمود. این دستور یک‌سری تست‌های استاندارد از قبل تهیه شده برای NS را اجرا می‌کند و گزارشی از خروجی هر تست بر روی صفحه نمایشگر اعلام می‌شود.

۱-۲: برنامه نویسی TCL و OTCL:

TCL توسط میلیون ها نفر در جهان مورد استفاده قرار می‌گیرد. TCL یک زبان با گرامر ساده می‌باشد و اجازه ائتلاف و مجتمع شدن بسیار ساده و راحت را با سایر زبان‌ها می‌دهد. این زبان توسط Jhon Ousterhout بوجود آمد، از خصوصیات این زبان عبارتست از:

- اجازه توسعه سریع را می‌دهد
- یک واسط گرافیکی را می‌تواند ایجاد نماید
- با بسیاری از Platform ها سازگار است
- جهت مجتمع شدن بسیار از خود انعطاف نشان می‌دهد
- جهت استفاده بسیار راحت است
- یک نسخه free است

در اینجا برخی از اصول برنامه نویسی tcl وotcl را اشاره می‌نماییم:

انتساب یک مقدار به یک متغیر از طریق دستور "Set" انجام می‌پذیرد; برای مثال "Set b 0" مقدار 0 را به متغیر منتسب می‌کند و این معادل "b =0" در زبان C می‌باشد.

وقتی می‌خواهیم از مقدار منتسب شده به متغیر استفاده نماییم باید از علامت "$" قبل از متغیر استفاده نماییم. برای مثال اگر قصد داریم مقداری را که در متغیر a وجود دارد را به متغیر x نسبت دهیم باید بنویسیم:

"Set x $a"

عملیات ریاضی از طریق دستور expression انجام می‌شود به عنوان مثال اگر ما بخواهیم به متغیر X مجموع ۲ متغیر a و b را منتسب نماییم، ما باید بنویسیم:

" Set X [expr $a+$b] "

در Tcl متغیرها نوع‌بندی نشده‌اند بنابراین یک متغیر می‌تواند از نوع رشته[1] یا مقدار صحیح[2] بسته به مقداری که شما بدان منتسب می‌نمایید، باشد برای مثال فرض کنید شما می‌خواهید حاصل تقسیم 1/60 را در خروجی داشته باشید. اگر بنویسیم:

Puts"[expr 1/60]"

---

[1] - string
[2] - Integer

نتیجه صفر خواهد بود. برای داشتن حاصل صحیح لازم است که مشخص نماییم که ما با اعداد صحیح کار نمی‌کنیم لذا می‌نویسیم:

Puts"[expr 1.0/60.0]"

علامت # که در ابتدای یک خط وجود داشته باشد، آن خط جزیی از برنامه محسوب نخواهد شئ و مفسر tcl آن خط را اجرا نخواهدکرد.

جهت ایجاد یک فایل یکی آنکه باید یک نام بدان اختصاص دهیم مثلا "filename" وسپس یک اشاره گر به آن که از میان برنامه tcl به‌منظور دسترسی به آن فایل استفاده می‌شود مثلا "file1" و این موارد با کمک دستور: Set file1 [open filename w] انجام می‌شود.

دستور puts جهت ارسال یک خروجی برای چاپ چه به صفحه نمایش و چه در یک فایل استفاده می‌شود. به خاطر داشته باشید با هر بار استفاده از این دستور یک خط جدید آغاز می‌شود. برای اجتناب از این مورد یک روش اضافه کردن nonewline – پس از دستور "puts" می‌باشد. اگر ما قصد چاپ کردن خروجی در یک فایل را داشته باشیم (که دربالا روش ایجادیک فایل را دیدیم) خواهیم نوشت: "puts filet "text". جدول بندی با اضافه کردن t\ انجام می‌شود. برای مثال اگر متغیر x مقدار Q داشته باشد و بنویسیم "x $ X" Puts $ file1، این دستوریک خط را در فایلی به نام "filename" ایجاد می‌نماید که دارای ۲ عضو : "X" و"۲" می‌باشد و توسط یک کاراکتر فضای خالی[1] ازیکدیگر جدا شده‌اند.

اجرای یک دستور سیستم عامل unix ازطریق نوشتن "exec" وسپس آن دستور امکان‌پذیر است. برای مثال می‌خواهیم ns منحنی نمودار اطلاعاتی را که در ۲ ستون در فایلی به نام data که از طریق شبیه‌سازی بدست آمده نمایش دهد. این اتفاق با استفاده از دستور Xgraph رخ داده و به صورت زیر نوشته می شود:

[1] - Space

exec Xgraph data &

(به خاطر داشته باشید که علامت & جهت اجرا شدن دستور در پس زمینه استفاده می‌شود)

-ساختاریک دستور if به صورت زیر است:

```
If {expression} {
   <excute some commands>
   } else {
   < excute some commands>
}
```

دستور if می تواند باسایر ifها وسایر elseها استفاده شود که در بخش "<excute some commands>" می‌توانند ظاهر شوند. به خاطر داشته باشید در زمان تست‌کردن برابری ما باید از "==" استفاده نماییم و نه از "=". وبرای تست‌کردن نابرابری از "!" استفاده می‌نماییم.

-حلقه‌ها در ns به فرم زیر هستند:

```
For {Set i 0 } { $i < s} { incr i } {
}        <excute some commands>
```

در این مثال دستورات درون حلقه ۵ مرتبه انجام می‌شوند پس از دستور for عبارت " { Set i 0 }" اعلام می‌نماید که متغیرi برای شمارش استفاده می‌شود و مقدار اولیه آن ۵ است. بخش دوم بین {} شرط ادامه‌یافتن حلقه را مشخص می‌نماید و می‌گوید حلقه را تا زمان کمتر بودن متغیر i از عدد ۵ ادامه بده و آخرین بخش از تعریف‌کردن عبارت for تغییر متغیر شمارنده می‌باشد.

در این جا ما متغیرi را یک‌به‌یک افزایش می‌دهیم ولی می‌توان مقدار عددی آن را کاهش داد (شمارش نزولی) و یا حتی از عبارات ریاضی برای افزایش و کاهش شمارشگر استفاده نمود.

tcl اجازه می‌دهد تا رویه‌ها[1] ایجاد شوند. در صورتی که آنها دارای دستور "return" باشند می‌توانند برخی مقادیر را برگردانند. فرم عمومی‌یک روال به نام blue را ملاحظه می نمایید:

```
Pro  blue {par 1 par 2 …} {
   Global   var 1  var 2
```

[1] - procedure

```
    <commands>
   Return  $  something
  }
```

روال برخی پارامتر را که می‌توانند اشیا فایل‌ها یا متغیر باشند، دریافت می‌نماید. در مورد ما این پارامترها نام par1 وpar2 وغیره می‌باشند. این پارامترها درون این روال با همین نام‌ها مورد استفاده قرار می‌گیرند. این روال به صورت .... blue x y فراخوانی می‌شود در جایی که مقادیر x و y توسط روال برای par1 و par2 استفاده می‌شود. اگر متغیر xوy در بین روال تغییر کند این تغییرات روی خود متغیرهای xوy تأثیری نخواهد داشت. به عبارت دیگر اگر بخواهیم روال قادر به تاثیرگذاری مستقیم روی متغیرهای خارجی باشد باید آن متغیرها را "global" تعریف نماییم.

```
# create  a procedure
Proc  test  { }  {
    Set a 43
    Set b 27
    Set c [ expr $a + $b ]
    Set d [ expr [ $a - $b ] * $c ]
    Puts  "c = $c   d  = $d"
    For { set    k 0 }    {$ 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 procedure
Test
```

جدول ۱-۱ :برنامه tcl برای انجام اعمال ریاضی

در مثال بالا آن متغیرها Var2وVar1 می‌باشند. در جدول ۱،۱یک مثال را مشاهده می‌نمایید که بسیاری از عملکردهای ریاضی را در tcl به نمایش می‌گذارد. عبارت Pow در این مثال متغیر d به توانk را به ما خواهد داد. در جدول ۱،۲یک مثال از برنامه tcl برای محاسبه اعداد اول نشان می‌دهد که چگونه از دستور if، زیر روال و حلقه استفاده نمایید متغیر argc شامل تعداد پارامترهایی است که

به برنامه ارسال می‌شود. متغیر argv یک بردار شامل متغیرهای ارسال شده به برنامه می‌باشد. (argc طول بردار argv می‌باشد). و دستور lindex به ما اجازه می‌دهد که موقعیت بردار مورد اشاره را در مکان مورد اشاره توسط پارامتر دوم را در اختیار بگیریم. بنابراین خط Set j [lindex $argv] مقدار موجود در اولین پارامتر ارسال‌شده به برنامه که در متغیر argv ذخیره شده است را به متغیر j منتسب می‌نماید.

```
#Usage :ns prime.tcl NUMBER
# NUMBER  is the number  up  to we want   to obtain  the   prime  numbers
#
If   {$argee!= 1 } {
#  Must  get  a single argument  or program  fails.
Puts  stderr  " ERROR! Ns  called  with  wrong number  of  argument ! ($argc)"
Exit 1
}else {
Set  j [lindex  $argv 0]
}

          Proc  prime  {j} {
             #computes all  the prime  numbers  till  j
            For  { set  a  2 {$a <=   $j }   {
               Set d  [exper  fmod  ($a ,$i)]
              If {$d==0} {
                Set  b  1 }
             }
            If  {$b ==1} {
                Puts  "$a  is  not  a prime number "
              } else{
               Puts "$a is a prime  number "
               }
            }
         }
Prime $j
```

جدول ۲-۱: برای محاسبه اولیه اعداد

```
# Usage ns fact.tcl  NUMBER
#  NUMBER is the number we mant  to obtain the factorial
#  if {$arge   !=1}  {
#  Must  get a single  argumet  or program  fails.
 Puts  stderr  "ERROR!  Ns called  with  wrong  number  of  arguments ! ($argc)"
Exit  1
```

```
} else  {
Set  f [lindex $argv 0 ]
}

Proc  Factorial  {x} {
For  {set resuit 1 }  {$x>1} {set  x [expr  $x -  1 ] }  {
Set  result  [expr  $result * $x ]
}
Return  $result
}
Set res" Factorial  of $f  is  $res"
```

جدول ۳-۱: برنامه ساده TCL برای محاسبه فاکتوریل یک عدد

نمونه برنامه ای برای محاسبه فاکتوریل یک عدد را نیز در جدول ۳-۱ ملاحظه می‌نمایید.

ما به طور خلاصه از طریق یک مثال مدل برنامه نویسی شی گرا رادر OTCL نیز مشکل خواهید داشت لذا لازم است قبلا با شی‌گرایی آشنا باشید. کلمه رزرو شده Class توسط نام کلاس تعریف شده در OTCL متابعت می‌شود. متدهای تعریف شده در کلاس‌ها با استفاده از کلمه instproc که پس از نام کلاس می‌آید و پس از آن نام آن متد و پارامترهایش می‌آید، تعریف می‌شوند. متد init همان سازنده (constructor) کلاس می‌باشد.

متغیر self یک اشاره‌گر از شی به خودش می‌باشد مانند متغیر this در C++ یا java. برای تعریف کردن نمونه متغیر در OTCL از واژه instvar استفاده می‌شود. لغت super class- برای تعریف کردن ارث‌بری یک کلاس از کلاس دیگر استفاده می‌شود. به عنوان مثال کلاس اعداد صحیح از کلاس اعداد حقیقی ارث می‌برد. در جدول ۴-۱ این موضوع را ملاحظه می‌نمایید.

```
Class Real

Real  instproc init  {a}  {
        $self instvar value –
         Set  value -$a
}
Real  instproc  sum {x}  {
```

```
            $self  instvar _+  [$x  set  value _]  = \t”
            Set  value _  [expr  $value  _+  {$x  set  value  _]]
             Puts  “$op  $value _”
}
Real  instproc  multiply {x} {
            $self instvar  value_
            Set op “$value _* [$x set value_] =\t”
            Set  value _[expr $value _*[$x set value _]]
            Puts “$op $value _ “
}
Real instproc divide {x}  {
           $self instvar value_
           Set op “$value _ / [ $x set value _]  = \t”
           Set  value _[ expr $value _/ [$x set value _]]
           Puts  “$op $value _”
}
Class Integer –superclass Real
  Integer instproc  divide {x} {
            $self instvar value_
            Set op “$value _ / [$x  set value _] =\t”
            Set d [expr $value _ / [ $x set value _]]
            Set value _ [expr round ($d) ]
            Puts  $op $value_”
            }

            Set  real A [new Real 12.3]
            Set realB [ new Real 0.5 ]

            $real A sum  $real B
            $real A multiply $realB
            $real A divide $real B
            Set integer A [new Integer 12]
            Set integer B[new Integer 5]
            Set integer C [ new Integer 7]

            $integer A multiply $integer B
            $integer B divide $integer C
```

جدول ۱-۴:استفاده واقعی برنامه ساده OTCL واشیا عدد صحیح

فصل ۲

۲-۱ -مقدمات شبیه‌ساز NS :

در این بخش گامهای اولیه را که شامل:

شروع به کارکردن و خاتمه‌دادن شبیه‌سازی

تعریف توپولوژی شبکه شامل صف‌ها،

تعریف لینک‌ها و نودهای شبکه،

تعریف برنامه کاربردی[۱] وعامل[۲]،

ابزار مشاهده شبیه‌سازی (NAM)،

و نیز ابزار Tracing (ثبت رویداد و وقایع شبیه سازی) می‌باشد ارائه می‌نماییم.

مجموعه‌ای از مثال‌های ساده که ما را قادر می‌سازد تا این گام‌ها را با شبیه‌ساز NS به انجام برسانیم نیز ارائه می‌شود.

[1] - Application
[2] - Agent

شروع و خاتمه ns :

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 مورد نظر استفاده نمود. مهم‌ترین این ویژگی‌ها عبارتند از :

- تعداد نودها
- مشخصه هر نود
- مشخصه لینک‌ها اتصال دهنده 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

در اولین قدم برای انجام شبیه‌سازی، باید یک شیء ازکلاس 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. به‌عنوان 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.

لذا یک شبیه‌ساز ns با این دستور آغاز می‌شود

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

لذا همین دستور اولین خط در متن[1] شبیه‌سازی می‌باشد. این خط یک متغیر جدید ns را با استفاده از دستور set تعریف می‌نماید. البته به هر شکلی که بخواهید می‌توانید آن را نامگذاری نمائید اما از آنجا که نمونه‌ای از کلاس Simulator و لذا یک شی از آن می‌باشد، پس عموماً کاربران آن را به همین شکل نامگذاری می‌نمایند.

کد [new Simulator] در واقع یک نمونه‌سازی از کلاس Simulator با استفاده از کلمه رزرو شده new می‌باشد. لذا با استفاده از این تغییر جدید ns می‌توانیم از تمامی توابع و متدهای کلاس Simulator آن‌گونه که خواهیم دید استفاده نماییم.

به‌منظور ایجاد فایل خروجی حاوی اطلاعات شبیه‌سازی (فایلtrace) یا فایلی که برای مشاهده شبیه‌سازی استفاده می‌شود (فایلهای nam) از دستور open استفاده می‌نماییم.

```
# open the tarce file
Set tracefile 1[ open out. tr w]
# open the  NAM  trace file
Set name file  [open out. nam w]
$ns namtarce-all $nam file
```

قطعه کد بالا یک فایل داده ازنوع trace بنام out.tr و یک فایل مشاهده ازنوع nam (برای استفاده توسط ابزار NAM)بنام out.nam ایجاد می‌نماید. درون کدهای tcl شبیه‌سازی این فایلها را از طریق نامشان به‌کار نمی‌گیریم و عوض آن از اشاره‌گرهایی که در بالا در حین ایجاد این فایل‌ها به‌وجود آورده‌ایم، tracefile1 وnam file استفاده می‌نماییم.

خطوط اول و چهارم مثال بالا فقط جهت توضیحات[2] می‌باشند و جزیی است دستورات شبیه‌سازی نیستند. بخاطر داشته باشید که این توضیحات با علامت # آغاز می‌شوند. خط دوم فایل out.tr

---

[1] - script
[2] - comment

را که برای نوشتن استفاده می‌شود، باز می‌کند. حرف W نشان می‌دهد که این فایل به‌منظور نوشتن ایجاد شده است. سومین خط از یک متد Simulator بنام trace-all که دارای پارامتری است که نام فایلی می‌باشد که اطلاعات trace درون آن نوشته خواهد شد، استفاده می‌نماید. با این دستور از کلاس Simulator، تمامی وقایعی را که در شبیه‌سازی رخ می‌دهد با قالب مخصوص که در ادامه مطالب توضیح داده خواهد شد، ردیابی می‌شوند و اگر فایلی مشخص کرده‌باشیم به همین قالب تمامی رویدادهای شبیه‌سازی از اول تا آخر شبیه سازی در آن فایل نوشته می‌شوند.

آخرین خط به شبیه‌ساز می‌گوید همه وقایع شبیه‌سازی را در قالب ورودی NAM ثبت نماید. این دستور همچنین نام فایلی را که وقایع توسط دستور $ns flush-trace درون آن نوشته خواهد شد نیز می‌دهد (در زیر برنامه finish) در مورد مثال ما این فایل توسط اشاره‌گر $nam file که همان فایل out.tr می‌باشد مشخص است. به یاد داشته باشید که استفاده از دستورات nam trace-all و trace-all ممکن است منجر به ایجاد فایل‌های بسیار بزرگی شوند. اگر ما قصد نگهداری و حفظ فضای ذخیره‌سازی‌مان را داشته باشیم باید از دستورات دیگر tracing استفاده نماییم تا زیر مجموعه ای از وقایع شبیه‌سازی شده را که مورد نیاز ما است دربر بگیرد. چنین دستوراتی را در ادامه مطالب خواهیم دید. خاتمه یک برنامه شبیه‌سازی توسط اجرای زیر برنامه finish انجام می‌پذیرد.

```
# Define a ‘finish’ procedure
Proc finish {}   {
        Global ns  tarce file 1 namfile
        $ns flush-trace
        Close $tracefile1
        Close $ namefile
        Exec nam out. nam  &
        Exit 0
}
```

کلمه proc یک زیربرنامه را که در این مورد نامش finish و بدون آرگومان می‌باشد، تعریف می‌نماید. کلمه global استفاده می‌شود تا بگوید از متغیرهایی استفاده خواهیم نمود که خارج از این زیربرنامه تعریف شده‌اند. متدflush-trace که متعلق به کلاس Simulator می‌باشد اطلاعات trace را

به فایل‌های مرتبط به آن‌ها، تخلیه می‌نماید. دستورtcl به نامclose، فایلهای trace را که قبلاً تعریف شده‌اند را می‌بندد و دستور exec برنامه nam را برای مشاهده شبیه‌سازی اجرا می‌نماید. متوجه باشید که ما نام واقعی فایل‌های nam وtrace را برای اجرا استفاده می‌نماییم و نه اشاره‌گر به آنها را چرا که آن (دستورnam)یک دستور خارجی است (خارج از برنامه ما). دستور exit اجرای شبیه‌سازی را پایان می‌دهد و عدد صفر را برمی‌گرداند که نشانه‌ای از وضعیت خاتمه است که به سیستم اعلام می‌نماید. مقدار صفر برای یک خاتمه صحیح مقدار پیش فرض است. سایر مقادیر می‌توانند استفاده بشوند که بگویند خاتمه‌یافتن اجرای شبیه‌سازی به‌علت رویداد برخی از خطاها می‌باشد.

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

به‌عنوان مثال دستوراتی که در زیر آمده است، باعث می‌شود که منبع ترافیکی از قبل تعیین شده $cbr0 در زمان 0.5 شروع به ارسال ترافیک نماید و در زمان 4.5 متوقف گردد.

```
$NS at 0.5 "$cbr0 start"
$NS at 4.5 "$cbr0 stop"
```

در پایان برنامه شبیه‌سازی‌مان باید زیربرنامه finish را فراخوانی نماییم و تعیین نماییم چه زمانی این خاتمه باید اتفاق بیفتد. برای مثال

```
$ns at 125.0 ″ finish″
```

برای صدازدن زیربرنامه در ثانیه ۱۲۵ ام استفاده می‌شود.

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

۲-۲- تعریف نودها(گره ها) ولینکهای شبکه :

در نرم‌افزار شبیه سازNS ، هر نود با کمک دستور زیر ایجاد می‌شود:

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

که در دستور فوق node_name نام نودشبکه 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 نشان دهنده نود و ۸ بیت بعدی نشان دهنده عامل ارسال در نود فوق می‌باشد. بنابراین با توجه به محدودیت ۸ بیتی در تعیین آدرس نودها، نمی‌توان در NS بیشتر از ۲۵۶ نود ایجاد نمود و اگر چنانچه تعدادنودها از ۲۵۶ بیشتر باشد، باید فیلد آدرس نود توسعه یابد. بدین منظور از دستورزیر استفاده می‌شود:

```
Node expander
```

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

حالت پیش فرض عملکرد هر نود در NS، به‌صورت یک‌پراکنی 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 Addr classifier بررسی می‌گردد و چنانچه مقصد بسته یک نود دیگر باشد، بسته دریافتی به یکی ازلینکها 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 وتحلیل شده و سپس بسته ورودی عامل ارسال مربوطه انتقال می‌یابد.

درهنگام استفاده از مسیریابی دینامیکی، بالاترین بیت در ناحیه آدرس، نشان دهنده چندپراکنی یا یک‌پراکنی بودن آدرس می‌باشد. چنانچه بیت فوق صفر باشد، آدرس دهی از نوع یک‌پراکنی است و درغیراین‌صورت آدرس از نوع چند پراکنی می‌باشد. بنابراین در حالت چندپراکنی حداکثر تعداد نودها برابر با ۱۲۸ است.

برای اتصال یک عامل جدید به نود و یا حذف عامل قبلی از نود مورد نظر، 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 agent در نود node می‌شوند.

- **لینک‌ها در** NS

در این بخش، یکی دیگر از المان‌ها تشکیل دهنده توپولوژی شبکه را که لینک می‌باشد، بررسی می‌کنیم. فقط لینک‌های نقطه به نقطه مورد بررسی قرار می‌گیرند. البته شبیه‌سازNS علاوه بر لینک‌ها نقطه به نقطه، سایر لینک‌ها متداول مانند : لینک‌ها بادسترسی 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 شیء لینک می‌باشد.

-Queue: نشان دهنده آدرس صف موجود در لینک است. در لینک‌های کاملاً یک‌طرفه یک صف وجود دارد، درحالی‌که در سایر لینک‌ها امکان وجود بیشتر از یک صف می‌باشد.

-Link : اشاره کننده به المانی است که لینک را با مشخصه های داده شده مدل سازی می‌کند.

-Ttl : اشاره‌گر به المانی است که مقدار زمان زندگی (TTL) هر بسته را نگهداری می‌کند.

-Drophead : اشاره‌گر به شیء است که اتلاف در لینک را پردازش می‌کند.

چنانچه متغیر $traceAllFile تعریف شده باشد، در این صورت امکان ردیابی بسته ها از لحظه ورود به صف تا لحظه خروج از آن وجود دارد.

دستورات زیر برای تعریف لینک‌ها در NS به‌کار می‌روند.

$NS simplex-link n1 n2 bw delay q_type

دستور زیر برای ایجاد یک لینک یک طرفه بین دو نود n1 و n2 به‌کار می‌رود. متغیرهای دستور فوق به‌صورت زیر تعریف می‌شوند:

-bw : میزان پهنای باند لینک اتصال دهنده نودهای n1 و n2

-delay : مقدار تأخیر لینک اتصال دهنده نودهای n1 و n2

-q_type : نوع مکانیسم صف بندی در بافر لینک که بعداً بیشتر در مورد آن توضیح خواهیم داد

$NS duplex-link n1 n2 bw delay q_type

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

$NS simplex-link-op n1 n2 op args

از این دستور برای تعیین برخی مشخصه‌های خاص مانند: جهت لینک در هنگام نمایش در nam (نرم‌افزاری که به‌صورت 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 n1 و n2 به‌کار می‌رود.

$NS duplex-link-op n1 n2 op args

مشابه دستور قبلی می‌باشد، با این تفاوت که برای لینک‌های دوطرفه به‌کارمی‌رود.

$link cost cost-value

مقدار ارزش لینک را برابر با cost-value قرار می‌دهد. در حالت پیش فرض 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.

وقتی که چندین گره ایجاد کرده باشیم می توان لینکهایی را که متصل کننده آنها به هم می باشند، ایجاد نماییم به عنوان مثال می توانید لینک را به ان شکل تعریف کرد :

$ ns duplex-link $n0 $n2 10Mb l0ms DropTail

بدین معناست که گره های $n0و$n2 با استفاده ازیک اتصال دو طرفه که دارای تاخیر انتشار ۱۰میلی ثانیه وظرفیت ۱۰ مگا بیت بر ثانیه برای هر جهت می‌باشند بهم متصل می‌شوند.جهت تعریف یک لینک یکطرفه بجای یک لینک دوطرفه باید عبارت duplex-link را با عبارت simplex linkجایگزین نماییم.

در NS صف خروجی متعلق به یک گره بعنوان بخشی از هر لینک که ورودی اش آن لینک است پیاده سازی می‌شود.تعریف لینک مشمول راهی برای مورد سرریز (over flow) روی آن صف نیز می‌باشد. در مورد مثال ما اگر ظرفیت با فرصف خروجی اشباع گردد آنگاه آخرین بسته‌ای که وارد می‌شود، حذف می گردد(با گزینه و انتخاب DropTail). البته بسیاری از گزینه‌های جانبی دیگری نیز وجود دارند مانند RED(Random Early Discard) و نیزFQ(Fair Queueing)، DRR(Deficit Round Robin)،SFQ(stochastic Fair Queueing)، CBQ که شامل یک اولویت ویک زمانبندی نوبت گردشی Round-Robin می‌باشد.

البته ما باید ظرفیت بافر صف مرتبط با هر لینک رانیز تعریف نماییم بعنوان مثال خواهیم داشت:

```
# Set Queue Size Of Link (no-n2) To 20
 $ns queue-limit $no $n2 20
```

سر ریزصف به شکل ارسال بسته های حذف شده به یک عامل تهی (Null Agent) پیاده‌سازی می‌شود شیTTL(Time to Live)پارامتر زمان حیات را برای آن بسته که دریافت شده است، محاسبه می نماید.

بسته‌ها بر چسب‌هایی دارند که در شبکه به‌روز رسانی می‌شوند و این برچسب TTL مشخص می‌نماید چه مدت زمانی بسته‌ها می‌توانند قبل از آن‌که به مقصد برسند، در شبکه وجود داشته باشند. یک لینک ساده دارای فرمی است که در تصویر زیر مشاهده می‌نمایید:

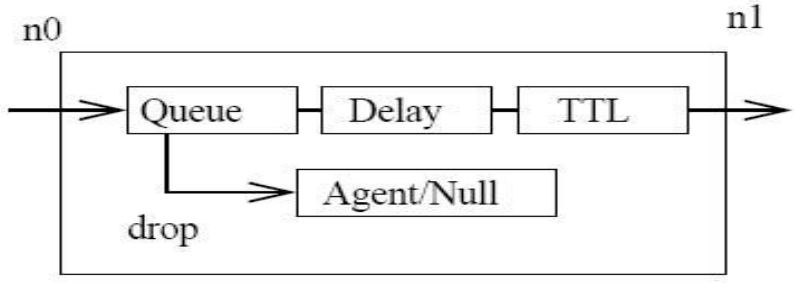


Figure 2.1: A simplex link

یک لینک دو طرفه از دو لینک موازی یک‌طرفه ایجاد می‌شود.

به‌عنوان یک مثال از یک شبکه ساده تصویری را که در زیر نمایش داده شده را در نظر بگیرید

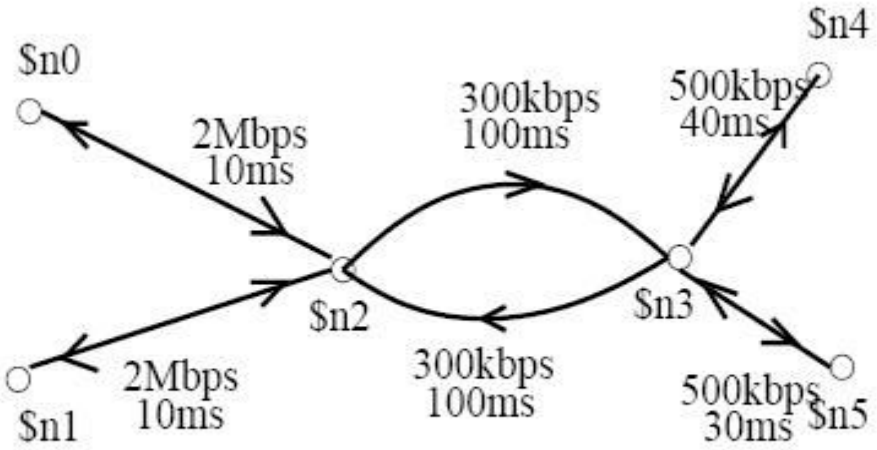


Figure 2.2: Example of a simple network

توپولوژی این شبکه از طریق اجرای کدهای زیر تعریف شده است.

```
# create six nodes
Set n0 [$ns node]
Set n1 [$ns node]
Set n2 [$ns node]
Set n3 [$ns node]
Set n4 [$ns node]
Set n5 [$ns node]
#create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms droptail
$ns duplex-link $n1 $n2 2mb 10ms droptail
$ns simplex-link $n2 $n3 0.3Mb 100ms droptail
$ns simplex-link  $n3 $n2 0.3Mb 100ms droptail
$ns duplex-link  $n3 $n4 0.5Mb 40ms droptail
$ns duplex-link $n3 $n5 0.5Mb 30ms droptail

#set Queue size of link (n2-n3) to 20
$ns queue-limit $n2 $n3 20
```

به یاد داشته باشید که ظرفیت بافر مرتبط بایک لینک تعریف شده است (لینک متصل کننده نودهای n2, n3) صفوف مرتبط با سایر لینک‌ها مقدار از پیش تعریف شده ۵۰ را دارا می‌باشد. این مقدار پیش فرض را می‌توانید در فایل ns-default.tcl در دستور Queue set limit _50 چک نمایید.

این مقدار پیش فرض را چگونه بیابیم و تغییر دهیم؟

با پیداکردن زیربرنامه Queue-limit درفایل ns-lib.tcl :

```
Simulator insproc queue – limit {n1 n2 limit}
$self   instvar   link
[$link_( [ $n1   id  ] :[ $n2   id ] ) queue] set limit_$limil
```

در اینجا ما می‌بینیم که queue limit در واقع یک متد از کلاس Simulator است که ۳پارامتر ورودی دارد :۲گروه که لینک را تعریف می‌نماید و محدوده و اندازه بافر صف. مقدار ارجاع داده شده به متغیرlimit داده می‌شود.

```
#setup a TCP connection
Set tcp [new agent/TCP]
$ns attach-agent $n0 $tcp
Set sink [new agent/TCPsink]
$ns attach-agent $n4 $sink
$tcp set fid_1
$tcp set packetsize_552

#setup a FTP over TCP connection
Set ftp [new application/FTP]
$ftp attach-agent $tcp
```

۲–۳– عامل ها و برنامه های کاربردی: (Agents and Applications)

پس از تعریف‌کردن توپولوژی شبکه متشکل از نودها و لینک‌ها اکنون لازم است یک جریان ترافیک ایجاد کرده، ازطریق آن توپولوژی ارسال نماییم.

برای رساندن این جریان به مقصد لازم است برنامه‌های کاربردی و عامل‌ها را مسیردهی نماییم تا از آن توپولوژی استفاده نمایند. در مثال گذشته ممکن بود ما بخواهیم یکFTP را بین گره-های $n0 و $n4 اجرا نماییم و نیز یک برنامه‌های کاربردی از نوع CBR (Constant Bit Rate) را بین نودهای $n1 و$n5 پروتکل لایه شبکه‌ای که توسط FTP استفاده می‌شود TCP/IP می‌باشد و این در حالی است که CBR از UDP بهره می‌گیرد. ابتدا باید یک عامل از نوع TCP را بین گره مبدا $n0 وگره مقصد $n4 تعریف نماییم پس FTP از آن استفاده خواهد نمود. سپس در قطعه کد دوم یک عامل UDP بین نود مبدا $n1 و مقصد $n5 ایجاد می‌نماییم که CBR از آن استفاده خواهد نمود.

```
#setup a TCP connection
Set tcp [new agent/TCP]
$ns attach-agent $n0 $tcp
Set sink [new agent/TCPsink]
$ns attach-agent $n4 $sink
$tcp set fid_1
$tcp set packetsize_552
#setup a FTP over TCP connection
Set ftp [new application/FTP]
$ftp attach-agent $tcp
```

FTP over TCP -۱-۳-۲

TCP یک پروتکل کنترل ازدحام قابل اعتماد ار نوع دینامیک است که از بسته‌های تصدیق و اعلام وصول[1] تولید شده توسط مقصد برای اطمینان از دریافت بسته‌ها استفاده می‌نماید.

بسته‌های از دست رفته به عنوان سیگنال ازدحام تعبیر می‌شوند. با این وصف وجود لینک‌های دو طرفه برای TCP برای برگرداندن تصدیق‌ها به مبدا ضروری خواهد بود. در خط اول قطعه کد بالا داریم:

```
Set   tcp [ new Agent / Tcp ]
```

[1] - Acknowledgement

این دستور علاوه بر بیان نوع عامل یک اشاره‌گر بنام tcp را به عامل TCP می‌دهد که شی در NS می باشد. دستور $ns attach –agent $n0 $tcp نود مبدا اتصال TCP را تعریف می کند و دستور Set Sink [new agent/Tcp Sink] رفتار نود مقصد TCP را تعریف می‌کند و اشاره‌گری بنام Sink را بدان برمی‌گرداند. بخاطر داشته باشید در TCP، نود مقصد نقش مهمی را در تولید تصدیق‌ها واعلام و وصول بسته‌ها به‌منظور تضمین‌کردن همه بسته‌ها در رسیدن به مقصد ایفا می‌کند.

```
#setup a UDP connection
Set upd [new agent/UDP]
$ns attach-agent $n1 $udp
Set null [new agent/UDP]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2

#setup a CBR over UDP connection
Set cbr [new application /traffic/CBR]
$cbr attach-agent $udp
$cbr set packetsize _1000
$cbr set rate _0.01Mb
$cbr set random_false
```

دستور $ns attach – agent $n4 $sink گره مقصد را تعریف می نماید و دستور $ns connect $tcp $sink اتصال TCP بین نودهای مبدا و مقصد را ایجاد می‌نماید.TCP پارامترهای زیادی با مقادیر پیش فرض مقداردهی شده دارد که اگر بخواهید به وضوح می‌توانید آنان را تغییر دهید. برای مثال مقدار پیش‌فرض اندازه بسته در TCP مقدار ۱۰۰۰ بایت است و می‌تواند به مقدار دیگری مثلا ۵۵۲ بایت تغییر یابد این اتفاق با دستور $tcp set packet size _552 رخ می‌دهد.

وقتی ما چندین جریان ترافیکی داریم ممکن است بخواهیم میان آنان تمایز قایل شویم به عنوان مثال با رنگ‌های متفاوت در مشاهده آنان و از طریق اجرای دستور $tcp set fid _1 که یک شناسه جریان"۱" را به اتصال TCP نسبت می‌نماید، انجام می‌پذیرد. قصد داریم که در ادامه مطالب به

جریان UDP، شناسه "۲" را اختصاص دهیم پس از آنکه اتصال TCP تعریف شد، برنامه کاربردی FTP روی آن تعریف می شود و این عمل در سه خط انتهایی قطعه کد فوق انجام می‌شود.

به خاطر داشته باشید که هر دو عامل TCP و برنامه کاربردی FTP دارای اشاره‌گرهایی هستند اولی را tcp ودومی را ftp نام گذارده‌ایم اما دقت نمایید که نام‌های دیگری را نیز می‌توانیم انتخاب نماییم.

۲-۳-۲- CBR Over UDP :

پس از تعریف اتصال UDP و برنامه کاربردی CBR بر روی آن، یک UDP در مبدا (Agent/ UDP) و یک مقصد تهی (Agent/Null) به‌همان شیوه‌ای که برای TCP گذشت، تعریف می‌شود:

```
#setup a UDP connection
Set upd [new agent/UDP]
$ns attach-agent $n1 $udp
Set null [new agent/UDP]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2

#setup a CBR over UDP connection
Set cbr [new application /traffic/CBR]
$cbr attach-agent $udp
$cbr set packetsize _1000
$cbr set rate _0.01Mb
$cbr set random_false
```

برای CBR که از CDP استفاده می‌نماید قطعه کد فوق چگونگی تعریف نرخ ارسال اطلاعات و نیز اندازه بسته را نشان می‌دهد. بجای تعریف نرخ ارسال در دستور $cbr Set rate_0.01 راه دیگر تعریف کردن فاصله زمانی[1] بین ارسال بسته‌ها با استفاده ازدستور

```
$cbr Set interval_0.005
```

[1] - Interval

می‌باشد از سایر خصوصیات CBR،گزینه و متغیر _random می باشد که یک پرچم جهت مشخص کردن وجود یا عدم وجود پارازیت تصادفی در مدت زمان ارسال ترافیک می‌باشد به صورت پیش‌فرض این گزینه غیر فعال می‌باشد و می‌تواند با کد

$cbr Set random_1

فعال گردد. اندازه بسته‌ها نیز با دستور

$cbr Set pachet size _<size>

می‌تواند تنظیم(به بایت) شود

۲-۳-۳-UDP with other traffic sources :

ممکن است با سایر ترافیک‌های کاربردی که از پروتکلUDP استفاده می‌نماید بخواهیم شبیه‌سازی را انجام دهیم:

منبع ترافیکی Exponential on-off , pareto on-off ویک منبع متنی برفایل trace. دو منبع اولی به ترتیب به شکل زیر تعریف می شوند:

```
Set source [new Application /Traffic  Exponential
Set source [new Application /Traffic pareto]
```

این منابع ترافیکی پارامترهایی همچون اندازه بسته (به بایت)packet size ، _burst_time که متوسط زمان"on" و فعال‌بودن را مشخص می‌نماید را دریافت می‌کنند. در منبع pareto یک پارامتر شکل دهی _shap را نیز تعریف می‌کنیم و در زیر مثالی از یک pareto on/off را ملاحظه می‌نمایید:

```
Set source[new application/traffic/pareto]
$source set packetsize_500
$source set burst_time_200ms
$source set idle_time_400ms
$source set rate_100k
$source set shape_1.5
```

برنامه کاربردی ترافیک مبتنی بر trace به صورت زیر تعریف می‌شود: ابتدا یک فایل trace را تعریف می‌نماییم:

Set tracefile[new trace file]
$trace file file name <file >

سپس برنامه‌های کاربردی را تعریف می‌نماییم و آن را با فایلی که باید دارای قالب دودویی باشد، متصل نماییم:

Set src [new Application / Traffic/Trace]
$src attach –tracefile $ tracefile
فایل مزبور باید شامل فاصله زمانی ارسال بسته‌ها و اندازه بسته به بایت باشد.

۲-۴-رویدادهای زمانبندی :

NS یک شبیه‌ساز مبتنی بر رویدادهای جدا از هم می‌باشد. درکدهای شبیه‌سازی تعریف می‌شود که رویدادها چه موقع رخ دهند. دستور ابتدایی Set ns [new Simulator] یک زمانبند رویداد ایجاد می‌نماید و رویدادها پس از آن با استفاده از قالب

$ns at <time> <event>

زمانبندی می‌شوند. زمانبند در زمان اجرایی ns از طریق دستور $ns run آغاز به‌کار می‌کند در مثال ساده ما، باید آغاز بکار و خاتمه FTP و CBR را زمانبندی نماییم و این می‌تواند از طریق دستورهای زیر انجام شود:

```
$ns  at  0.1  "$clor  start"
$ns  at  1.0  "$ftp  start"
$ns  at  124.0  "$ftp  stop"
$ns  at  124.5  "$clor  stop"
```

بنابراین FTP در بازه زمانی ۱ تا ۱۲۴ فعال می باشد CBR در ثانیه‌های ۰،۱ تا ۱۲۴،۵ فعال و در حال ارسال ترافیک خواهد بود.

ما هم اکنون آماده هستیم که همه شبیه‌سازی را اگر دستورات ما در فایلی بنامex1.tcl نوشته شده باشند برای اجرا به سادگی تایپ می کنیم:

ns ex1.tcl

به یاد داشته باشید که در قطعه کد زیر، زیربرنامه دیگری در پایان اضافه شده‌است. که یک فایل خروجی را با مقادیر لحظه‌ای اندازه پنجره ارسال TCP در فواصل زمانی ۵۰۱ ثانیه می‌نویسد. در مثال نام فایل خروجی winfile می‌باشد. این زیربرنامه از نوع بازگشتی است که در هر ۵۰۱ ثانیه خود را فراخوانی می‌کند. آن یک پارامتر منبع TCP و فایلی که ما می‌خواهیم به عنوان خروجی در آن بنویسیم را دریافت می‌نماید.

```
Set  tim#set queue size of link (n-n3)to 20
$ns queue –limit  $n2 $n3 20
Setup a TCP connection
Set tcp [new agent/TCP]
$ns attach-agent $n0 $tcp
Set sink [new agent/TCPS ink]
$ns attach-agent $n4 $sink
$tcp set fid_ 1
$tcp set packetsize _ 552


#setup a FTP over TCPconnection
Set ftp [new application /FTP]
$ftp attach –agent $tcp

#setup a UDPconnection
Set udp [new agent/UDP]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2

#setup a CBRover UDP connection
Set cbr [new application /traffic/CBR]
$cbr attach-agent $udp
$cbr set packetsize_ 1000
$cbr set rate_0.01Mb
$cbr set random_false

$ns at 0.1  “$cbr start”
$ns at 1.0 “$ftp start”
$ns at 124.0 “$ftp stop”
$ns at 124.5 “$cbr stop”

#procedure for plotting window size.gets as arguments the name
# of the tcp source nodded (called “tcp source “)and of output file.
```

```
Proc plotwindow {tcpsource file} {
Global ns e0.1
Set now [$ns now]
Set cwnd [$tcpsource set cwnd_]
Puts $file "$now $cwnd"
$ns at {exper $now+$time]  "platwindow $tcp source $file"
}
$n at 0.1 "plotwindow $tcp $winfile"
$ns at 125.0 "finish"
$ns run
```

۲-۵-استفاده از nam :

وقتی ما مثال قبلی را اجرا می‌نماییم، nam که ابزار مشاهده می‌باشد یک شبکه شامل ۶ گره را نمایش می‌دهد. محل استقرار گره‌ها می‌تواند به طور تصادفی انتخاب شود. برای آنکه مکان گره‌ها را به صورت تصویر زیر مقداردهی نماییم،

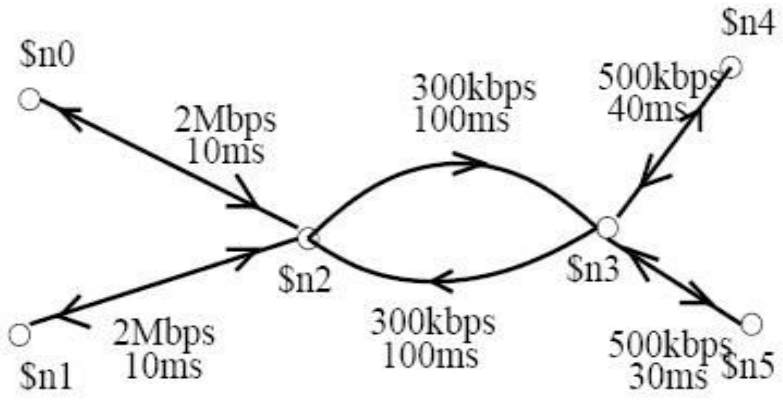


Figure 2.2: Example of a simple network

کد زیر را اضافه می‌نماییم:

```
# Give node position (for NAM)
$ns duplex-link-op $n0 $n2  orient right-up
$ns duplex-link-op $n1 $n2 orient right-up
$ns simplex-link-op $n2 $n3 orient right
$ns simplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n3 $n5 orient right-down
```

نکته: اگر استقرار گره‌ها به صورت تصادفی انتخاب شود و رضایت بخش نباشد با فشردن دکمه "re- layouf" با انتخاب یک استقرار تصادفی دیگر می‌توانید بخت خود را بیازمایید. همچنین می‌توان مکان گره‌ها را با کلیک‌کردن Edit/view و کشیدن هر گره و بردن آن به مکان دلخواه بوسیله موس، تنظیم نمود. یادآوری می‌نماییم که نمایشگر nam به صورت متحرک بسته‌های CBR را که از نود ۱ به ۵ می‌روند به رنگ قرمز و بسته‌های TCP را که از گره ۰ به ۴ می‌روند را به رنگ آبی نمایش می‌دهد. بسته‌های اعلام وصول TCP که مسیر عکس بسته‌های TCP را می‌پیمایند نیز به رنگ آبی نمایش داده می‌شوند اما با ابعاد بسیار کوچکتر چرا که بسته‌های تصدیق ۴۰ بایتی هستند در حالی‌که بسته‌های TCP، ۵۵۲ بایت می‌باشند. برای تنظیمات رنگ‌ها درشروع کد برنامه مثال قبلی تعریف می‌کنیم:

```
$ns  color 1 Blue
$ns  color 2 Red
```

به خاطر داشته باشید اگر ما هم‌اکنون یک فایل nam داشته باشیم مجبور نیستیم برای مشاهده آن لزوماً ns را اجرا نماییم بلکه می‌توانیم به جای آن مستقیماً دستور< نام فایل nam> nam را تایپ نماییم.

یک عکس فوری[1] از نمایشگر nam می‌تواند توسط یک چاپگر و یا درون یک فایل چاپ شود این عمل را در منوی بالای nam و در گزینه File می‌توانید محقق سازید.

برخی از توانمندی‌های NAM :

رنگ کردن گره‌ها : به عنوان مثال اگر بخواهیم n0 در رنگ قرمز نمایش داده شود خواهیم نوشت:

```
$n0  color  red
```

[1] - Snap shot

شکل گره‌ها: به صورت پیش فرض گره‌ها، دایره ای هستند. اما می‌توانند به اشکال مختلفی نمایش داده شوند به عنوان مثال می‌توان با نوشتن $n1 shape box و یا به جای لغت box از hexagon برای ۶ ضلعی به جای مربعی و یا circle برای کروی بودن.

- رنگ کردن لینک‌ها:

```
$ns  duplex-link-op   $n0  $n2 color "green"
```

اضافه کردن و حذف کردن علامت: می‌توانیم در زمان خاصی یک گره را علامت بزنیم (مثلاً در زمانی که یک منبع ترافیکی را در آن زمان فعال می‌نماییم) مثلاً:

```
$ ns  at  2.0 " $n3   add-mark  m3  blue box "
$ ns  at  30.0 " $n3 delete-mark  m3"
```

این باعث می‌شود علامت آبی رنگ اطراف گره ۳ در فاصله زمانی [2,30] ایجاد شود.

اضافه کردن برچسب: یک گره می‌تواند ازیک زمان به بعد با یک برچسب مثلا "active node" نمایش داده شود مثلاً: برای زمان ۱۰۲ به بعد با همان برچسب داریم:

$ns at 1.2 "$n3 label \"active node\"" وبرای برچسب زدن با "Tcp Input Link" به لینک n0-n2 تایپ می‌کنیم:

```
$ns duplex –link –op $n0 $n2  label "Tcp  input link "
```

اضافه کردن متن: در پایان پنجره NAM می‌تواند یک متن در زمان خاص ظاهر شود. این متن می‌تواند توصیف‌کننده رویداد خاصی در آن زمان باشد. مثال:

```
$ns  at  5  $ns trace-annotate  $n2   \"packet drop\"
```

یک امکان دیگر از nam، اضافه کردن یک مانیتور اندازه‌صف می‌باشد برای مثال جهت مانیتور و مشاهده‌کردن صف ورودی لینک n2-n3 داریم:

```
$ns  simplex - link – op $n2 $n3  queuepos  0.5
```

واسط گرافیکی NAM را در تصویر زیر ملاحظه می‌نمایید:

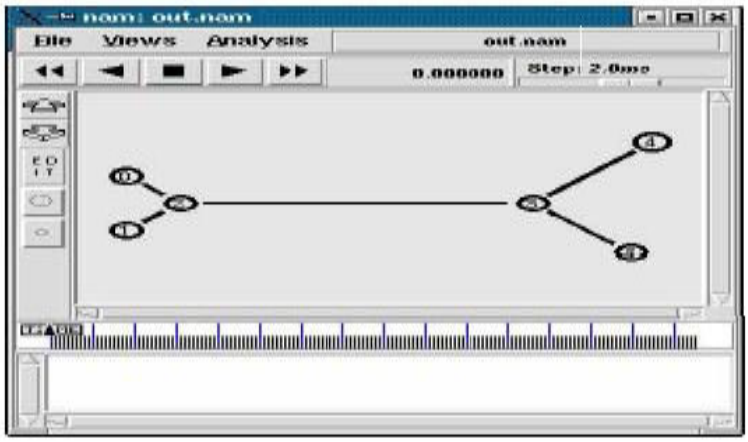


Figure 2.3: NAM graphic interface

۲-۶-Tracing[1]:

۲-۶-۱-Tracing object:

شبیه‌ساز NS می‌تواند فایل‌های Trace را به هر دو شکل تصویری و اسکی[2]ایجاد نماید که هر دو فایل منطبق بر رویدادهای رخداده در شبکه می‌باشند. وقتی از tracing استفاده می‌نماییم آن‌چنان‌که در تصاویر قبلی از ساختار یک لینک مشاهده نمودیم در اینجا ۴ شیء جدید در لینک که در تصویر زیر می بینید: DrpT ,RecT,DeqT, EnqT

[1]- ثبت رویدادها
[2] - ascii

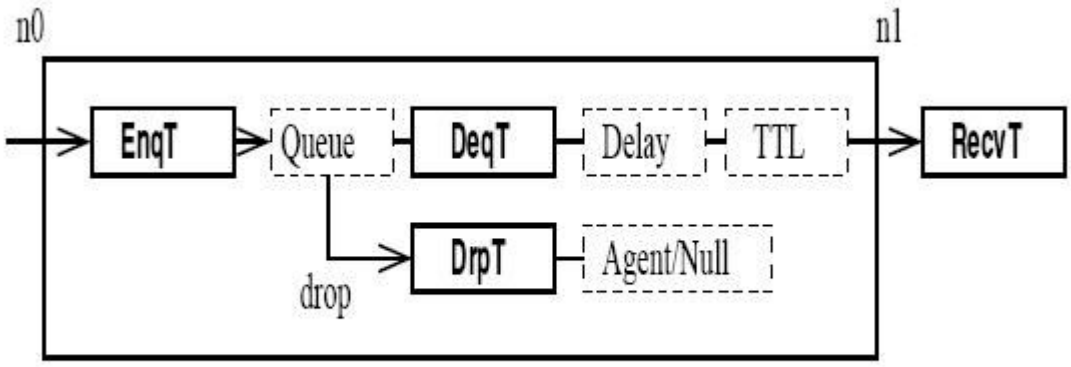


Figure 2.4: Tracing objects in a simplex link

EnqT، اطلاعات مربوط به بسته وارد شده و داخل صف ورودی لینک قرار گرفته را ثبت می‌نماید. اگر صف دچار Overflow (سر ریز) شود اطلاعات این بسته را شی DeqT ثبت می‌نماید. DeqT اطلاعات بسته‌های موجود در صف را ثبت می‌نماید. در نهایت RecvT اطلاعاتی را درباره بسته‌هایی که به خروجی لینک رسیده‌اند را به ما ارائه می‌کند. NS با ما اجازه می‌دهد اطلاعات بیشتری را از طریق tracing بدست آوریم. یکی از آن راهها Queue Monitoring و یا مشاهده پشت رویدادهای رخداده روی یک صف می‌باشد

۲-۶-۲-ساختار فایلهای trace :

وقتی اطلاعات ثبت شده از رویدادهای شبکه را دریک فایل اسکی قرار می‌دهیم در ۱۲ فیلد سازماندهی می‌شود که تصویر آن را ملاحظه می‌نمایید:

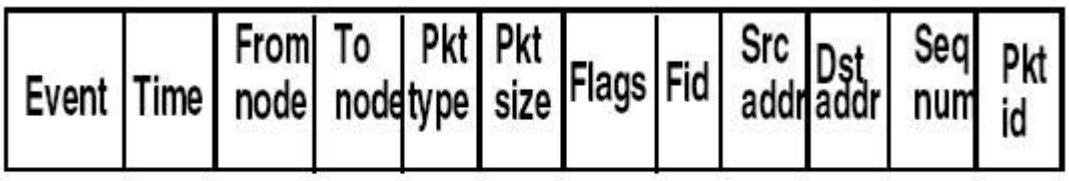


Figure 2.5: Fields appearing in a trace

مفهوم فیلدها به شرح ذیل می‌باشند:

فیلد۱: اولین فیلد نوع رویداد را بیان می‌کند که با استفاده از ۴گزینه: d,-,+,r که به ترتیب به دریافت در خروجی لینک، واردشدن به صف، خروج از صف و حذف آن بسته دلالت می‌نمایند.

۲. دومین فیلد زمان رخداد آن رویداد را نشان می‌دهد.

۳.گره ورودی لینکی که رویداد در آن اتفاق افتاده است را نمایش می‌دهد.

۴. گره خروجی لینک حاصل رویداد را نمایش می‌دهد.

۵. نوع بسته را برای مثال CBR و یا TCP نمایش می‌دهد نوع به نامی که ما بدان Application ترافیکی داده‌ایم دلالت می‌نماید. برای مثال TCP Application در مثال قبلی را "tcp" می‌نامد.

۶. اندازه بسته

۷. چندنوع پرچم در این فیلد موجود می‌باشد.

۸. این فیلد شناسه جریان ترافیکی ومتعلق به IP_{v6} می‌باشد که کاربر می‌تواند هر جریان را که در کد OTCL ورودی تنظیم نماید. این شناسه برای تجزیه وتحلیل بیشتر می‌تواند مورد استفاده واقع شود. این امکان همچنین برای زمانی که رنگ جریان ترافیکی برای نمایش توسط NAM مشخص می‌شود، می‌تواند مورد استفاده قرار بگیرد.

۹.این آدرس مبدا به فرم"پورت. نود"

۱۰. آدرس مقصد است به همان فرم مشابه

۱۱.شماره توالی بسته متعلق به پروتکل لایه شبکه می‌باشد. اگرچه در پیاده‌سازی واقعی UDP از شماره توالی استفاده نمی‌شود، اما NS توالی UDP را نیز ثبت می‌نماید تا برای اهداف تجزیه وتحلیل و مطالعات استفاده شود.

۱۲. آخرین فیلد شماره یکتای آن بسته را مشخص می‌نماید به عنوان یک مثال خطوط اولیه یک trace را که توسط کدهای مثال قبلی تولید شده است را ملاحظه می‌نمایید:

```
+0.1 1 2 cbr 1000 ------ 2 1.0 5.0 0 0
- 0. 1 1 2 cbr 1000 ------ 2 1.0 5.0 0 0
R 0.114 1 2 cbr 1000 ------ 2 1.0 5 0 0 0
```

+ 0.114 2 3 cbr 1000 ------ 2 1.0 5.0 0 0
- 0.114 2 3 cbr 1000 ------ 2 1.0 5.0 0 0
R 0.240667 2 3 cbr 1000 ------ 2 1.0 5.0 0 0
+ 0.240667 3 5 cbr 1000 ------ 2 1.0 5.0 0 0
- 0.240667 3 5 cbr 1000 ------ 2 1.0 5.0 0 0
R 0.240667 3 5 cbr 1000 ------ 2 1.0 5.0 0 0
+ 0.9 1 2 cbr 1000 ------- 2 1.0 5.0 1 1
- 0.9 1 2 cbr 1000 ------- 2 1.0 5.0 1 1
R 0.914 1 2 cbr 1000 ------- 2 1.0 5.0 1 1
+ 0.914   2 3 cbr 1000 ------ 2 1.0 5.0 1 1
- 0.914   2   3 cbr 1000 ------ 2 1.0 5.0 1 1
+ 1 0 2   tcp   40 -------- 1 0.0 4.0 0 2
- 1 0 2   tcp   40 -------- 1 0.0 4.0 0 2
R 1.01016 0 2 tcp 40 ------- 1 0.0 4.0 0 2
+ 1.01016   2 3 tcp 40 ------- 1 0.0 4.0 0 2
- 1.01016   2 3 tcp 40 ------- 1 0.0 4.0 0 2
R 1.040667 2 3 cbr 1000------- 2 1.0 5.0 1 1
+ 1.040667 3 5 cbr 1000------- 2 1.0 5.0 1 1
- 1.040667 3 5 cbr 1000------ 2 1.0 5.0 1 1
R 1.086667 3 5 cbr 1000 -------- 2 1.0 5.0 1 1
R 1.111227 2 3 tcp 40 --------- 1 0.0 4.0 0 2
+ 1.111227 3 4 tcp 40 --------- 1 0.0 4.0 0 2
- 1.111227 3 4 tcp 40 --------- 1 0.0 4.0 0 2
R 1.151867 3 4 tcp 40 --------- 1 0.0 4.0 0 2
+ 1.251867 4 3 ack 40 --------- 1 4.0 0.0 0 3
- 1.251867 4 3 ack 40 --------- 1 4.0 0.0 0 3
+ 1.251867 4 3 ack 40 --------- 1 4.0 0.0 0 3
- 1.251867 4 3 ack 40 --------- 1 4.0 0.0 0 3
R 1.292507 4 3 ack 40 --------- 1 4.0 0.0 0 3
+ 1.292507 3 2 ack 40 ---------- 1 4.0 0.0 0 3
- 1.292507 3 2 ack 40 ---------- 1 4.0 0.0 0 3
R 1.393573 3 2 ack 40 ---------- 1 4.0 0.0 0 3
+ 1.393573 2 0 ack 40 ---------- 1 4.0 0.0 0 3
- 1.393573 2 0 ack 40 ---------- 1 4.0 0.0 0 3
R 1.403733 2 0 ack 40 ---------- 1 4.0 0.0 0 3
+ 1.403733 0 2 tcp 552 ---------- 1 0.0 4.0 1 4
- 1.403733 0 2 tcp 552 ----------- 1 0.0 4.0 1 4
+ 1.403733 0 2 tcp 552 ----------- 1 0.0 4.0 2 5
- 1.405941 0 2 tcp 552 ----------- 1 0.0 4.0 2 5
R 1.415941 0 2 tcp 552 ----------- 1 0.0 4.0 1 4

۲-۶-۳- Trace کردن زیر مجموعه ای از رویدادها :

در بخش قبلی و تاکنون چگونگی ثبت‌کردن کلیه رویدادهای شبیه‌سازی شده را در نظر گرفته بودیم. اکنون راهی برای ثبت‌کردن تنها جزیی از رویدادها را ارائه می‌دهیم. اولین راه جایگزین‌کردن دستور $ns trace-all< نام فایل > با دستور $ns trace –queue می‌باشد که به عنوان مثال به شکل زیر می‌توان تایپ نمود:

$ ns trace –queue $n2 $n3 $file1

که نتیجه آن یک فایل خروجی از نوع trace می‌باشد که شامل تنها رویدادهایی است روی لینک بین نودهای n2,n3 رخ داده است (این نودها را در مثال قبلی تعریف کرده بودیم) یک دستور مشابه می‌تواند برای nam trace با استفاده از nam – trace queue بجای trace – queue باشد البته trace- queue باید پس از تعریف لینک‌ها واقع شوند. همچنین امکان فیلترکردن رویدادها از طریق کدهای tcl و با استفاده از دستورات سیستم عامل unix نیز وجود دارد.

```
#set queue size of link (n-n3)to 20
$ns queue –limit  $n2 $n3 20
Setup a TCP connection
Set tcp [new agent/TCP]
$ns attach-agent $n0 $tcp
Set sink [new agent/TCPS ink]
$ns attach-agent $n4 $sink
$tcp set fid_ 1
$tcp set packetsize _ 552

#setup a FTP over TCPconnection
Set ftp [new application /FTP]
$ftp attach –agent $tcp

#setup a UDPconnection
Set udp [new agent/UDP]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2

#setup a CBRover UDP connection
Set cbr [new application /traffic/CBR]
$cbr attach-agent $udp
```

```
$cbr set packetsize_ 1000
$cbr set rate_0.01Mb
$cbr set random_false

$ns at 0.1  "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 124.0 "$ftp stop"
$ns at 124.5 "$cbr stop"

#procedure for plotting window size.gets as arguments the name
# of the tcp source nodded (called "tcp source ")and of output file.
Proc plotwindow {tcpsource file} {
Global ns
Set  time0.1
Set now [$ns now]
Set cwnd [$tcpsource set cwnd_]
Puts $file "$now $cwnd"
$ns at {exper $now+$time]  "platwindow $tcp source $file"
}
$n at 0.1 "plotwindow $tcp $winfile"
$ns at 125.0 "finish"
$ns run
```

- **چند مثال شبیه‌سازی در NS:**

همان‌طور که قبلاً نیز گفته شد، برنامه‌های شبیه‌سازی در NS، در یک فایل اسکریپت با پسوند tcl نوشته می‌شوند و سپس برای اجرای آن از دستور زیراستفاده می‌شود:

```
NS  <tclscript>
```

البته باید قبلاً مسیری که فایل tclscript آنجا قرار دارد در path آورده شده باشد. می‌توان بدون آن‌که دستورات شبیه‌سازی را در فایل tclscript نوشت، در محیط tcl shell دستورات را یکی‌یکی وارد نمود، که روش چندان روش مناسبی نمی‌باشد. در این بخش از گزارش اخیر به ذکر چند مثال و توصیف عملکرد هر یک می‌پردازیم.

مثال ۱۳:

در این مثال به‌طور خیلی ساده و ابتدایی به نحوه ایجاد توپولوژی شبکه شامل نودها، لینک‌ها و نحوهٔ مونیتور نمودن صف‌ها و مشاهده آن در nam می‌پردازیم.

اولین قدم، ایجاد یک object از شبیه‌سازی می‌باشد که این امر به‌صورت 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 به‌صورت 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 مرحلهٔ بعدی یک روال پایان به نام finish ایجاد می‌شود، که هنگام اتمام شبیه‌سازی دستورات موجود در این روال اجرا می‌شود. روال فوق به‌صورت زیر می‌باشد:

```
proc finish()
{
global  NS  nf
NS  flush-trace
close  $nf
exec  nam  out.nam  &
exit  0
}
```

در لحظه 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 و به دنبال آن nam اجرا می‌شود و نتایج شبیه‌سازی که در فایل out.nam نوشته شده بود، به‌صورت انیمیشن نشان داده می‌شود.

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

```
$NS  at  5.0  "finish"
```

دستور فوق موجب می‌شود که روال finish در زمان 5.0 اجرا گردد.

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

```
$NS run
```

مطالبی که در بالا ذکر گردید، قسمت‌های اصلی تشکیل دهندهٔ هرفایل tclscript برای انجام شبیه‌سازی می‌باشد. جهت ایجاد توپولوژی شبکه شامل : نودها، لینک‌ها اتصال دهندهٔ نودها، عاملان ارسال و اتصال آنهابه نودها، باید از دستورات خاص NS استفاده کرد، که بهتر است که دستورات فوق، قبل از دستور "$NS at 5.0 "finish و بعد از توصیف روال finish آورده شود.

به‌عنوان مثال برای ایجاد یک توپولوژی شبکه شامل دو نود به نام‌های n0 و n1، از دستورات زیر استفاده می‌شود:

```
set n0 [$NS node]
set n1 [$NS node]
```

جهت اتصال دو نود فوق به یکدیگر، از طریق لینک دو طرفه با سرعت 1Mb/s، تأخیر 10 میلی ثانیه و بافر از نوع FIFO، از دستور زیر استفاده می‌شود:

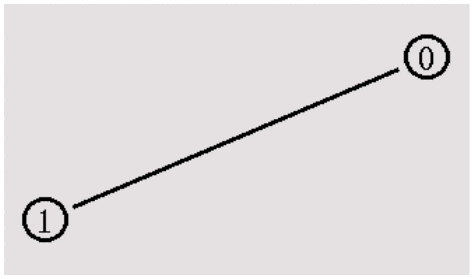
```
$NS duplex-link $n0 $n1 1Mb 10ms DropTail
```

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 0
}

#Create two nodes
set n0 [$NS node]
set n1 [$NS node]

#Create a duplex link between the nodes
$NS duplex-link $n0 $n1 1Mb 10ms DropTail

#Call the finish procedure after 5 seconds of simulation time
$NS at 5.0 "finish"

#Run the simulation
$NS run
```

بعد از اجرای مثال فوق، چنانچه در محیط گرافیکی Xwindows باشیم، خروجی مطابق با شکل۱-۱ در روی صفحهٔ مونیتور مشاهده خواهد شد.



شکل(۱-۱): خروجی مثال۱۳ در محیط nam

مثال ۱۴:

در مثال ساده قبل، فقط دو نود شبکه ایجاد گردیدند و به کمک یک لینک به یکدیگر متصل شدند. در مثال فوق هیچ‌گونه داده ای 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 زیربه‌کار می رود:

```
set  cbr0  [new Agent/CBR]
```

```
$NS  attach-agent  $n0  $cbr0
$cbr0  set packetSize  500
$cbr0  set interval  0.005
```

دستورات فوق، باعث ایجاد یک منبع ترافیکی CBR با نام cbr0 که به نودn0 متصل است، می‌شود. مشخصه طول بسته (packetSize) و فاصله زمانی بین بسته ها(interval)، به ترتیب برابر با ۵۰۰ بایت و 0.005 ثانیه (معادل سرعت ۲۰۰ بسته در ثانیه) قرار داده شده‌است.

جهت ایجاد یک عامل Null به نام null0،که در حکم دریافت کننده ترافیک عمل می‌کند و اتصال آن به نود n1 دستورات زیر استفاده می‌شود:

```
set null0 [new Agent/Null]
$NS attach-agent  $n1 $null0
```

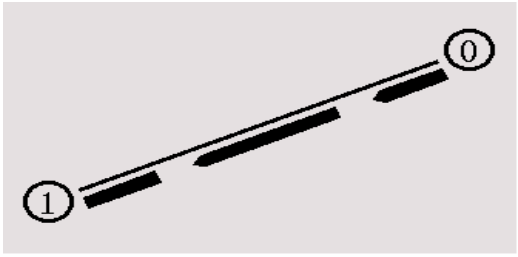
بعد از ایجاد عاملان ارسال و دریافت، باید اتصال بین این دو عامل برقرار گردد. این کار به‌صورت زیر انجام می‌شود.

```
$NS connect $cbr0  $null0
```

جهت تعیین زمان شروع به‌کار 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 5.0 در نظر گرفته شده بود، باشد. دستورات زیر، زمان شروع و پایان منبع ترافیکی cbr0 را به ترتیب برابر با 0.5 و 4.5 قرار می‌دهد.

```
$NS at 0.5 "cbr0 start"
$NS at 4.5 "cbr0 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 n0 شروع به ارسال بسته‌های اطلاعاتی به مقصد نود n1 می‌کند که در nam به‌صورت شکل (۱-۲) قابل رؤیت می‌باشد.



شکل (۱-۲):
خروجی حاصل
از مثال ۱۴

مثال ۱۵:

در این مثال یک شبکه با چهار نود در نظر می‌گیریم، که دو نود شبکه از طریق نود سوم برای چهارمین نود شبکه داده ارسال می‌کنند. در این مثال نحوه جداسازی و تشخیص ترافیک‌های نود های فوق از یکدیگر را توصیف خواهیم کرد. ابتدا برای ایجاد چهار نود به نام‌های n0، n1، n2 و n3 از دستورات زیر استفاده می‌کنیم:

```
set n0 [$NS node]
set n1 [$NS node]
set n2 [$NS node]
set n3 [$NS node]
```

جهت اتصال نودهای فوق به یکدیگر و ایجاد لینک، دستورات زیر را اضافه می نماییم:

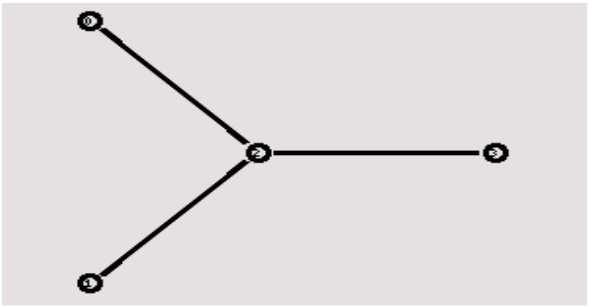
```
$NS duplex-link $n0 $n2 1Mb 10ms DropTail
$NS duplex-link $n1 $n2 1Mb 10ms DropTail
$NS duplex-link $n3 $n2 1Mb 10ms DropTail
```

بدین ترتیب تمام لینک‌ها اتصال دهنده نود های شبکه دارای سرعت 1Mb/s، تأخیر ۱۰ میلی ثانیه و بافر از نوع FIFO می‌باشند.

جهت تنظیم مناسب شکل توپولوژی شبکه در nam سه دستور زیر استفاده می‌شود:

```
$NS duplex-link-op $n0 $n2 orient right-down
$NS duplex-link-op $n1 $n2 orient right-up
$NS duplex-link-op $n2 $n3 orient right
```

بعد از اجرای دستورات فوق، توپولوژی شبکه و لینک‌ها اتصالی آن به‌صورت شکل (۱-۳) در nam نمایش داده خواهد شد.



شکل (۱-۳): خروجی حاصل از مثال ۱۵

بعد از ایجاد نودهای شبکه و لینک‌ها اتصال دهندهٔ آنها، باید منابع ترافیکی تعریف شوند و به نودهای فرستنده (نودهای n0 و n1) اتصال یابند. همچنین یک عامل گیرنده از نوع Null تعریف می‌کنیم، که به نود n3 اتصال می‌یابد و در حکم گیرنده می‌باشد.

```
set cbr0 [new Agent/CBR]
$NS attach-agent $n0 $cbr0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005

set cbr1 [new Agent/CBR]
$NS attach-agent $n1 $cbr1
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005

set null0 [new Agent/Null]
$NS attach-agent $n3 $null0
```

در دستورات فوق، دو عامل منبع ترافیکی CBR به نام‌های cbr0 و cbr1 ایجاد شده‌اند و به نودهای n0 و n1 اتصال یافته‌اند. منابع ترافیکی cbr0 و cbr1 دارای طول بسته برابر با ۵۰۰ بایت و سرعت ارسال ۲۰۰ بسته در ثانیه می‌باشند. همچنین یک عامل Null به نام null0 ایجاد شده‌است و به نود n3 اتصال یافته است.

حال جهت اتصال منابع ترافیکی cbr0 و cbr1 به عامل دریافت null0 دستورات زیر استفاده می‌شود:

```
$NS connect $cbr0 $null0
$NS connect $cbr1 $null0
```

اکنون برای تعیین زمان شروع به‌کار منابع ترافیکی و همچنین زمان خاتمه ارسال، از دستور at به‌صورت زیر استفاده می‌شود:

```
$NS at 0.5 "$cbr0 start"
$NS at 1.0 "$cbr1 start"
$NS at 4.0 "$cbr1 stop"
$NS at 4.5 "$cbr0 stop"
```

با توجه به دستورات فوق، منابع ترافیکی cbr0 و cbr1 به ترتیب در زمان‌های 0.5 و 1.0 شروع به فرستادن داده و در زمان‌های 4.0 و 4.5 متوقف می‌شوند.

با توجه به مشخصات ترافیکی منابع ترافیکی cbr0 و cbr1، مشخص می‌شود که نرخ ارسال هر یک برابر با 0.8 Mb/s می‌باشدو بنابراین مجموع نرخ ارسال منابع ترافیکی فوق برابر با 1.6 Mb/s می‌باشد، که این مقدار از ظرفیت قابل تحمل لینک n2 به n3 (1 Mb/s) بیشتر است. بنابراین انتظار داریم که بعد از مدتی، اتلاف بسته‌ها در نود n2 مشاهده شود.

برای مشاهده و تمایز جریانهای ترافیکی منابع cbr0 و cbr1 از یکدیگر، از مشخصه های جریان (fid) متفاوت به‌صورت زیر استفاده می‌کنیم:

```
$cbr0 set fid_ 1
$cbr1 set fid_ 2
```

همچنین برای آن‌که جریانهای ترافیکی cbr0 و cbr1 با رنگ‌های متفاوت در nam نشان داده شود، دستورات زیر را در اول برنامه شبیه‌سازی اخیر اضافه می‌نماییم:

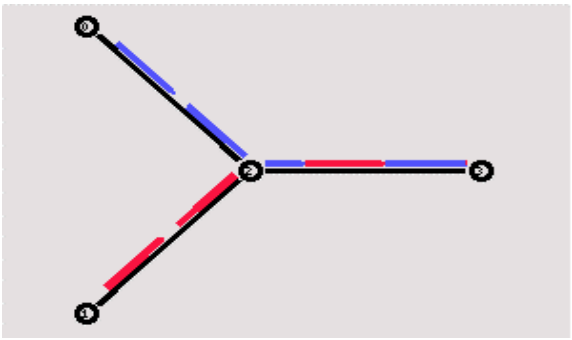
```
$NS color 1 Blue
$NS color 2 Red
```

بدین ترتیب جریان‌های ترافیکی منابع cbr0 و cbr1، با رنگ‌های آبی و قرمز از یکدیگر جدا می‌شوند. بعد از اجرای برنامه فوق، خروجی مطابق با شکل (۱-۴) را در nam مشاهده خواهیم کرد.

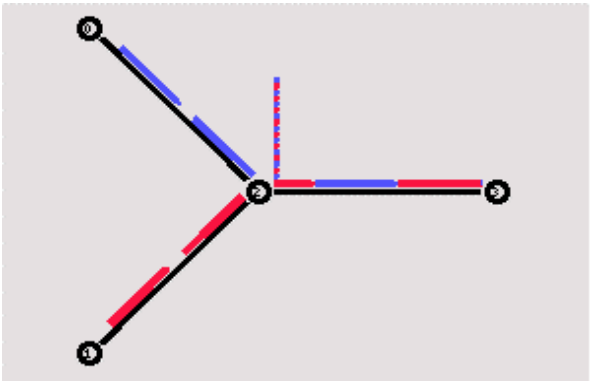
برای مونیتور نمودن صف مربوط به لینک n2 به n3 دستور زیر را اضافه می‌کنیم:

```
$NS duplex-link-op $n2 $n3 queuePos 0.5
```

بعد از اجرای دوباره مثال اخیر، خروجی شکل۱-۵ بر روی صفحه مونیتور دیده خواهد شد.



شکل (۱-۴): خروجی حاصل از مثال تکمیل یافته۱۵

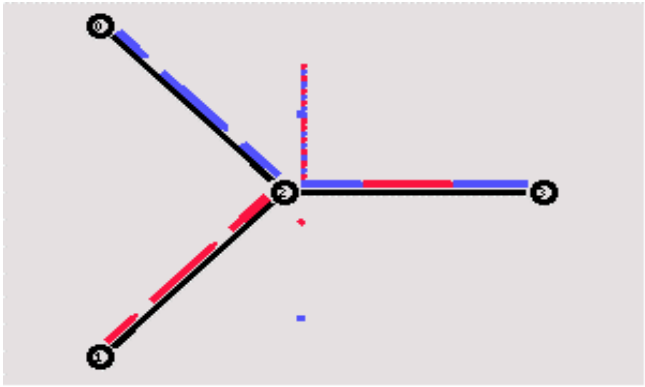


شکل (۱-۵) : خروجی حاصل از مثال تکمیل یافته ۱۵

برای حفظ عدالت در میزان اتلاف بسته‌های منابع ترافیکی cbr0 و cbr1، در بافر نود2n، تعریف لینک بین نود های n2 و n3 را به‌صورت زیر تغییر می‌دهیم :

$NS duplex-link $n3 $n2 1Mb 10ms SFQ

در این صورت از مکانیسم صف‌بندی SFQ، در صف مربوط به نود n2 به n3 استفاده می‌شود، که باعث می‌شود تا به‌طور مساوی بسته‌های ترافیکی منابع cbr0 و cbr1 در بافر نود n2 تلف گردند. خروجی مثال اخیر در این حالت در شکل (۱-۶) نشان داده شده است.



شکل (۱-۶): خروجی حاصل از مثال تکمیل یافته ۱۵

مثال۱۶ : مسیریابی دینامیکی

در این قسمت به بررسی یک مثال در مورد مسیریابی دینامیکی در NS می‌پردازیم. در مثال اخیر ابتدا یک شبکه با توپولوژی چرخان به‌وجود آمده و سپس یک خرابی موقت در یکی از لینکها آن به‌وجود می‌آید و سپس با کمک مسیریابی دینامیکی، مسیریابی تعدیل می‌یابد.

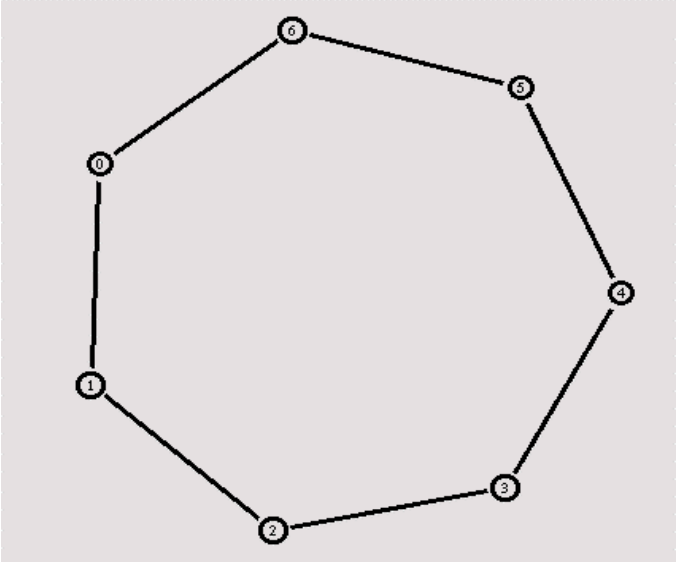
ابتدا برای ایجاد هفت نود شبکه، دستورات زیر استفاده می‌گردند:

```
for {set i 0} {$i < 7} {incr i} {
   set n($i) [$NS node]
  }
```

نودهای شبکه فوق در آرایه n() ذخیره می‌شوند. برای ایجاد لینکها اتصال دهنده نودها، دستورهای زیر به‌کارمی روند:

```
for {set i 0} {$i < 7} {incr i} {
   $NS duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail
  }
```

بدین ترتیب تمام نودهای شبکه با لینکهای با سرعت 1Mb/s، تأخیر ۱۰ میلی ثانیه و بافر از نوع FIFO به یکدیگر متصل می‌شوند و یک توپولوژی چرخان به‌صورت شکل (۱-۷) را ایجاد می‌کنند.



شکل (۱-۷): خروجی حاصل از مثال ۱۶

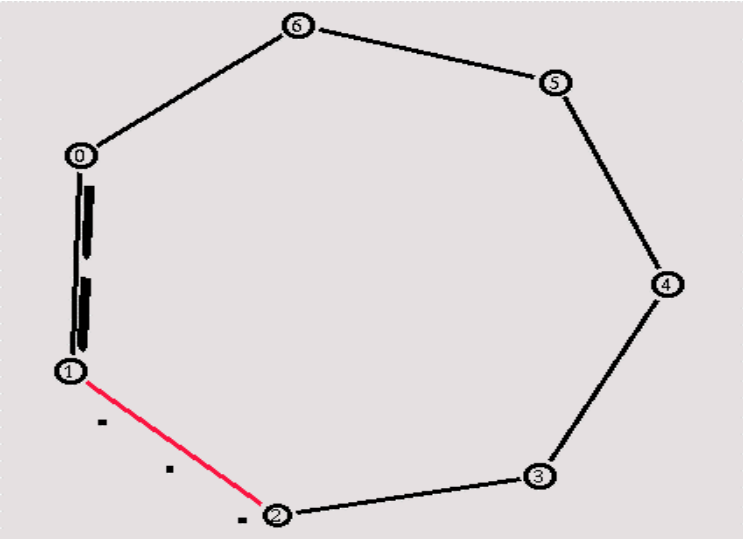
حال به کمک دستورات زیر، یک منبع ترافیکی CBR با نام cbr0 در نود n(0) ایجاد می‌گردد. همچنین عامل دریافت از نوع Null به نام null0 در نود n(3) به‌وجود می‌آید وسپس عاملهای ارسال cbr0 و دریافت null0 به یکدیگر متصل می‌گردند. منبع ترافیکی cbr0 در زمان 0.5 شروع به‌کار و در زمان 4.5 متوقف می‌گردد.

```
set cbr0 [new Agent/CBR]
$NS attach-agent $n(0) $cbr0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005

set null0 [new Agent/Null]
$NS attach-agent $n(3) $null0

$NS connect $cbr0 $null0

$NS at 0.5 “$cbr0 start”
$NS at 4.5 “$cbr0 stop”
```

حال فرض کنید که لینک واسط بین نودهای n(1) و n(2)، برای مدت یک ثانیه دچار خرابی شود. به کمک دستورات زیر می‌توان خرابی موقت به مدت یک ثانیه بین نود های n(1) و n(2) به‌وجود آورد:

```
$NS rtmodel-at 1.0 down $n(1) $n(2)
$NS rtmodel-at 2.0 up $n(1) $n(2)
```

بعد از اجرای مثال فوق، مشاهده می‌شود که بین زمان‌های یک و دو، لینک n(1) به n(2) قطع می‌گردد و تمام داده های ارسالی نود n(0) به n(3) در این فاصله ازبین می رود. در شکل (۱-۸) حالت فوق نشان داده شده است.



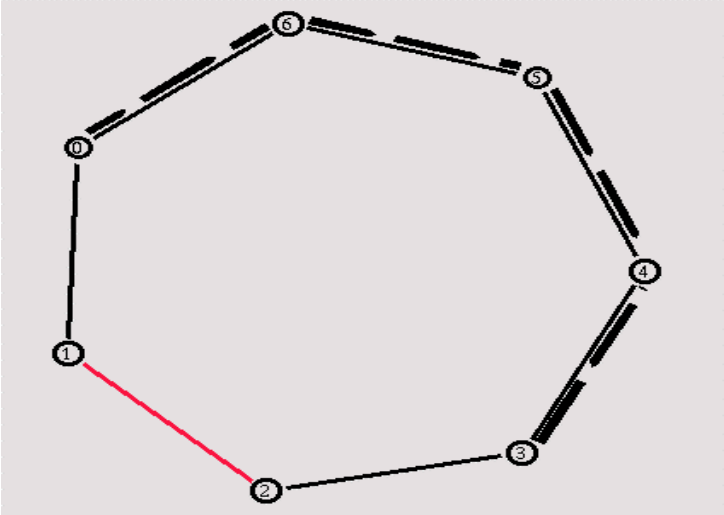
شکل (۱-۸): خروجی حاصل از مثال تکمیل یافته ۱۶

برای حل این مشکل، باید از مسیریابی دینامیکی استفاده گردد. بدین منظور در ابتدای برنامه شبیه‌ساز اخیر، دستور زیر را اضافه می‌کنیم:

```
$NS rtproto DV
```

در این مثال از الگوریتم مسیریابی دینامیکی بردار فاصله (Distance Vector, DV) استفاده شده‌است. بعد از اجرای دوبارهٔ مثال، دیده می‌شود که علاوه بر بسته‌های دادهٔ ارسالی بین نود های n(0) و n(3)، بسته‌های کوچک خاصی که در الگوریتم‌های مسیریابی دینامیکی برای مبادلهٔ اطلاعات مسیریابی بین نودهای شبکه به‌کار می‌رود، در شبکه جاری می‌شوند. وقتی که لینک بین نود n(1) و n(2) در زمان 1.0 دچار اشکال شود، جداول مسیریابی به روز آوری می‌شوند و ترافیک‌های ارسالی

نود n(0) به n(3)، از طریق مسیر n(6)،n(5)، n(4) فرستاده می‌شود. در شکل (۱-۹) خروجی حاصل از شبیه‌سازی حالت اخیر نشان داده شده است.



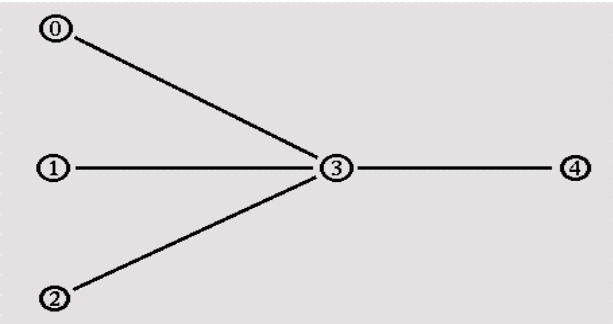
شکل (۱-۹): خروجی حاصل از مثال تکمیل یافته ۱۶

مثال ۱۷:

در این مثال به بررسی نحوهٔ ایجاد فایل خروجی برای xgraph می‌پردازیم. Xgraph یکی از اجزای تشکیل دهنده شبیه‌ساز NS می‌باشد، که برای رسم گرافیکی نتایج حاصل از شبیه‌سازی به‌کار می رود. جهت استفاده از xgraph، باید فایل خروجی مناسب برای آن ایجاد شود. در این مثال به بررسی نحوهٔ ایجاد فایل خروجی xgraph می‌پردازیم. برای ایجاد توپولوژی شبکه، از دستورات زیر استفاده می‌شود:

```
set n0 [$NS node]
set n1 [$NS node]
set n2 [$NS node]
set n3 [$NS node]
set n4 [$NS node]

$NS duplex-link $n0 $n3 1Mb 100ms DropTail
$NS duplex-link $n1 $n3 1Mb 100ms DropTail
$NS duplex-link $n2 $n3 1Mb 100ms DropTail
$NS duplex-link $n3 $n4 1Mb 100ms DropTail
```

بدین ترتیب یک شبکه با ۵ نود به نام‌های n3,n2,n1,n0 و n4 ایجاد می‌شود. نود های شبکه با کمک لینکهای با سرعت 1 Mb/s، تأخیر ۱۰۰ میلی ثانیه و بافر از نوع FIFO، به یکدیگر متصل شده اند و توپولوژی شبکه به‌صورت شکل (۱-۱۰) می‌باشد.



شکل(۱-۱۰): توپولوژی شبکه مثال ۱۷

برای اتصال منابع ترافیکی به نودهای n1،n0 و n2 روال attach-expoo-traffic به‌صورت زیر تعریف شده است:

```
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/CBR/UDP]
        $NS attach-agent $node $source
        #Create an Expoo traffic agent and set its configuration parameters
        set traffic [new Traffic/Expoo]
        $traffic set packet-size $size
        $traffic set burst-time $burst
        $traffic set idle-time $idle
        $traffic set rate $rate
        #Attach the traffic agent to the traffic source
        $source attach-traffic $traffic
        #Connect the source and the sink
        $NS connect $source $sink
        return $source
 }
```

روال فوق، دارای ۶ آرگمان ورودی است که عبارتند از : یک نود (مشخصه ورودی node)، یک گیرندۀ ترافیک که از قبل تعریف شده است (مشخصه ورودی sink)، طول بسته‌های ارسالی (مشخصه ورودی size)، میانگین طول نواحی انفجار و سکوت منبع ترافیکی نمایی (مشخصه‌های ورودی burst و idle) و نرخ ارسال منبع ترافیکی (مشخصه ورودی rate).

در روال فوق، ابتدا یک عامل ارسال از نوع UDP و با نام source ایجاد می‌گردد و به نود node متصل می‌شود. سپس یک منبع ترافیکی نمایی با نام traffic تعریف می‌شود و مشخصه‌های طول بسته، میانگین طول ناحیه burst و میانگین طول ناحیه idle آن به ترتیب برابر با size، burst و idle قرار می‌گیرد. منبع ترافیکی traffic به عامل ارسال source اتصال می‌یابد و سپس عامل ارسال source به عامل دریافت sink اتصال یافته و روال خاتمه می یابد.

برای استفاده از روال فوق، از دستورات زیر استفاده می‌شود:

```
set sink0 [new Agent/LossMonitor]
set sink1 [new Agent/LossMonitor]
set sink2 [new Agent/LossMonitor]

$NS attach-agent $n4 $sink0
$NS attach-agent $n4 $sink1
$NS attach-agent $n4 $sink2

set source0 [attach-expoo-traffic $n0 $sink0 200 2s 1s 100k]
set source1 [attach-expoo-traffic $n1 $sink1 200 2s 1s 200k]
set source2 [attach-expoo-traffic $n2 $sink2 200 2s 1s 300k]
```

بدین ترتیب سه عامل دریافت از نوع LossMonitor و با نام‌های sik0، sink1 و sink2 ایجاد شده‌اند و هرسه به نود n4 متصل گردیده‌اند. همچنین با کمک روال attach-expoo-traffic سه عامل ارسال UDP با نام‌های source0، source1 و source2 در نودهای n0،n1 و n2 ایجاد شده‌اند. به هریک از عامل‌های ارسال فوق، یک منبع ترافیکی از نوع نمایی و با مشخصه‌هایی که در بالا آورده شده است، وصل گردیده است.

همان‌طور که مشاهده می‌شود، در این مثال از عامل دریافت LossMonitor استفاده شده‌است. این عامل دریافت قادر است که میزان بایت های دریافتی را که در محاسبه پهنای باند لازم است، نگه‌داری کند.

برای ذخیره سازی داده‌های شبیه‌سازی سه فایل با نام‌های out0.tr، out1.tr وout2.tr بادستورات زیر باز می‌شوند

```
set f0 [open out0.tr w]
set f1 [open out1.tr w]
set f2 [open out2.tr w]
```

هنگام خاتمه شبیه‌سازی، سه فایل فوق باید بسته شوند و سپس برنامهٔ xgraph برای نمایش داده‌های حاصل از شبیه‌سازی اجرا گردد. بدین منظور از روال finish به‌صورت زیر استفاده می‌شود:

```
proc finish {} {
        global f0 f1 f2

        #Close the output files
        close $f0
        close $f1
        close $f2

        #Call xgraph to display the results
        exec xgraph out0.tr out1.tr out2.tr -geometry 800x400 &
        exit 0

  }
```

در این قسمت به بررسی نحوه ذخیره سازی داده‌ها در فایل‌های خروجی، می‌پردازیم. بدین منظور روال record به‌صورت زیر تعریف می‌گردد:

```
proc record {} {

        global sink0 sink1 sink2 f0 f1 f2
        #Get an iNStance of the simulator
        set NS [Simulator iNStance]
```

```
        #Set the time after which the procedure should be called again
        set time 0.5

        #How many bytes have been received by the traffic sinks?
        set bw0 [$sink0 set bytes_]
        set bw1 [$sink1 set bytes_]
        set bw2 [$sink2 set bytes_]

        #Get the current time
        set now [$NS now]

        #Calculate the bandwidth (in MBit/s) and write it to the files
        puts $f0 "$now [expr $bw0/$time*8/1000000]"
        puts $f1 "$now [expr $bw1/$time*8/1000000]"
        puts $f2 "$now [expr $bw2/$time*8/1000000]"

        #Reset the bytes_ values on the traffic sinks
        $sink0 set bytes_ 0
        $sink1 set bytes_ 0
        $sink2 set bytes_ 0

        #Re-schedule the procedure
        $NS at [expr $now+$time] "record"

    }
```

در این روال که هر نیم ثانیه، یک بار به‌طور متناوب اجرا می‌گردد، ابتدا تعداد بایتهای دریافت شده هر گیرنده محاسبه شده و در متغیرهای bw0، bw1 و bw2 قرار می‌گیرند و سپس پهنای باند هر عامل گیرنده برحسب Mb/s محاسبه شده و در فایل‌های خروجی مربوطه قرار می‌گیرد.

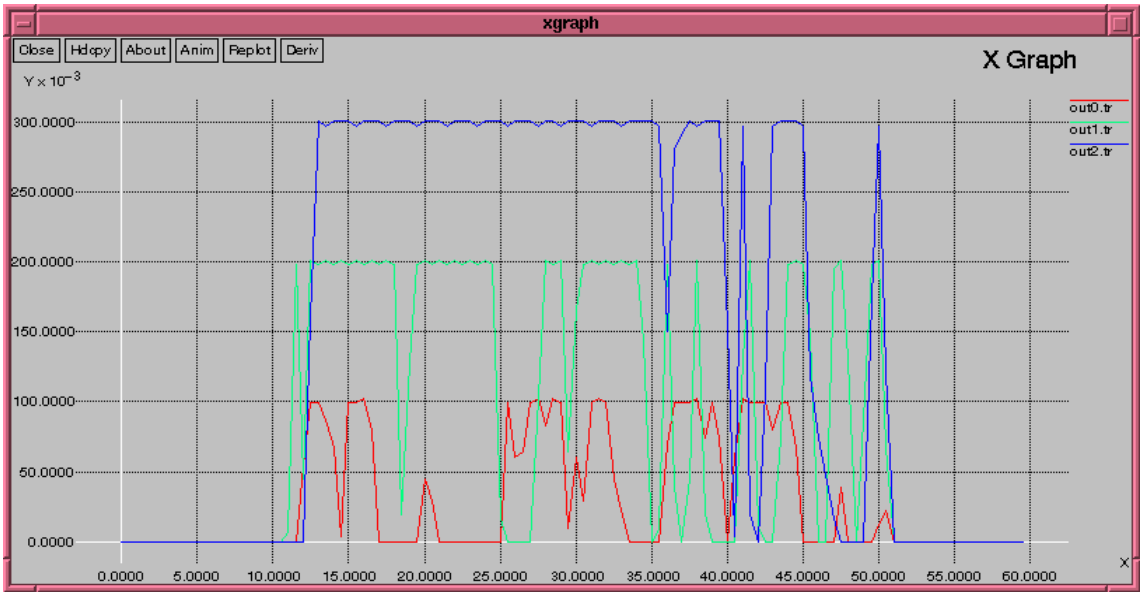
جهت زمان‌بندی شروع و پایان هر منبع ترافیکی از دستورات زیر استفاده می‌شود:

```
$NS  at 0.0 "record"
$NS  at 10.0 "$source0 start"
$NS  at 10.0 "$source1 start"
$NS  at 10.0 "$source2 start"
$NS  at 50.0 "$source0 stop"
$NS  at 50.0 "$source1 stop"
$NS  at 50.0 "$source2 stop"
$NS  at 60.0 "finish"
$NS  run
```

مشاهده می‌شودکه منابع ترافیکی source0، source1 و source2 هر سه در زمان ۱۰ شروع به ارسال و در در زمان۵۰ متوقف می‌شوند. روال record هر نیم ثانیه یک بار اجرا می‌شود و اقدام به

ذخیره سازی داده های حاصل از شبیه‌سازی می‌نماید. در زمان ۶۰ روال finish اجرا می‌شود و شبیه‌سازی خاتمه می یابد.

درشکل (۱-۱۱) خروجی برنامه فوق که نمودار گرافیکی پهنای باند می‌باشد، در محیط xgraph نشان داده شده است.



شکل(۱-۱۱): نمودار گرافیکی پهنای باند در مثال ۱۷

فصل۳

چگونگی کارکردن با فایل های trace

شبیه‌ساز NS می‌تواند مقدار زیادی اطلاعات به تفضیل برای رویدادهای رخداده روی شبکه فراهم می‌کند. اگر ما بخواهیم این اطلاعات را بررسی نماییم، ممکن است لازم باشد اطلاعات مناسب را از فایلهای trace، استخراج نماییم و آنها را دستکاری نماییم.

یکی از راهها آن است که برنامه‌ها را در هر زمان برنامه‌نویسی که می‌تواند فایل‌های اطلاعاتی را مدیریت نماید بنویسم. در عین حال چندین ابزار که به نظر می‌رسد مخصوصاً برای اهداف تطبیق داده شده تاکنون وجود دارد و تحت چندین سیستم عامل متفاوت (ویندوز،لینوکس،یونیکس) به طور رایگان عرضه شده‌اند و در همه آنها نیاز است که اسکریپت‌های کوتاهی که نوشته شده بدون نیاز به کامپایل شدن، تفسیر شده و اجرا شوند.

۳-۱ : پردازش فایل های اطلاعاتی با awk

Awk به ما اجازه می‌دهد اعمال ساده ای روی فایل‌های اطلاعاتی مانند معدل‌گیری مقادیر روی ستون داده شده، جمع کردن، یا ضرب کردن عبارات بین چندین ستون و همه اعمال قالب‌بندی مجدد اطلاعات و غیره را انجام دهیم. در ۲ مثال پایین نشان می‌دهیم چگونه از مقادیر ستون داده شده دریک فایل معدل‌گیری نماییم و سپس انحراف استاندارد را محاسبه می‌نماییم.

```
BEGIN{ FS = “\t”} {n1++ } {s=s+$4} END {print “average:” s/n1}
```

جدول ۳-۱ : اسکریپت awk برای معدل گیری مقادیر ستون ۴یک فایل

به خاطر داشته باشید ''\t'' زمانی که ستونهای جدول‌بندی شده باشند باید استفاده شوند و اگر این‌گونه نباشد، باید آن را با ″ ″ جایگزین کرد.

BEGIN {FS =''\t''} {n1 ++} {d=$4-t} {s2=+d*d}END {print ''standev :'' sqrt (s2/1n)}

جدول ۳-۲: اسکریپت awk برای بدست آوردن انحراف استاندارد ستون ۴یک فایل.

جهت استفاده اولین اسکریپت جهت محاسبه میانگین ستون چهارم یک فایل به نام ''out.ns'' در سیستم عامل unix تایپ می‌کنیم :

Awk - f Averag. awk out.ns

که ما باید به عنوان نتیجه چیزی مشابه :

Averag : 29.379

برای میانگین ستون ۴ (که اولین ستون را به نام شماره ۱ می شناسیم) را انتظار داشته باشیم.

حال برای محاسبه انحراف استاندارد آن ستون می‌نویسیم :

Awk - v t= 29.397 - f stDev : awk out.ns

آنچه که در پاسخ دریافت می‌کنیم مشابه standev:33.2003 می باشد به یاد داشته باشید که اسکریپت بالا، لازم است مقدار میانگین بدست آمده‌اند اسکریپت قبلی را در دستور محاسبه انحراف استاندارد،کپی نماییم. این مثال نشان می‌دهد که چگونه پارامترها را به یک اسکریپت awk، پاس دهیم. به یاد داشته باشید اگر در پایان اسکریپت اول (جدول ۳-۱) تقسیم به nl را انجام ندهیم، به جای بدست

آوردن معدل و میانگین، مجموع اقلام ستون۴ آن فایل را خواهیم داشت. بهترین راه برای بدست آوردن میانگین وانحراف معیار استفاده از آرایه‌هاست:

```
BEGIN {FS = "\t"} { val [n1]=$4} {n1 ++} {s=s+$4} END
  av=s/n1
   for (i  in val ) {
   d=val [i]-av
    s2=s2+d*d
}
Print "average: "av " standev"  sqrt (s2/n1)}
```

جدول ۳-۳ : اسکریپت awk میانگین و انحراف استاندارد.

مثال بعدی یک فایل با پانزده ستون (۰ تا ۱۴) را به عنوان ورودی می‌گیرد پس از آن یک خروجی ۵ستونی را که در ابتدا هیچکدام از آنها را نداشت ایجاد می‌نماید. یک ستون از فایل اولیه می-باشد که ستون شماره ۱ است و ستون ۲ تا ۵ به ترتیب مجموع ستونهای ۵-۳ ، ۸-۶ ، ۱۱-۹ و۱۴-۱۲ است

```
BEGIN {FS ="\t"} { l 1 =$3+$4+$5} {12=$6+$7+$8}{d1=$9+$10+$11}\
{d2=$12+$13+$14}{print $1"\t"l1"\t"12"\t"d1"\t"d2 }END {}
```

شکل ۴-۳ :اسکریپت awk برای برش و کنار هم قراردادن ستون‌ها

برای استفاده از این اسکریپت می‌توانید بنویسید :

```
Awk  -f suma.awk      conn4.tr  >outfile
```

که فایل اصلی conn4.tr وخروجی درون فایل به نام outfile قرار خواهد گرفت و فایل suma.awk فایلی است که کد جدول ۳-۴ درون آن واقع گردیده است.

۳-۲ : استفاده از grep :

دستور grep در unix به ما اجازه می‌دهد تایک فایل را فیلتر نماییم. ما فایل جدیدی را ایجاد می‌نماییم که تنها شامل خطوطی از فایل اولیه است که دنباله‌ای از کاراکترها را که مشخص نموده‌ایم،دارا می‌باشند. به عنوان مثال خروجی trace در ns ممکن است شامل همه بسته‌هایی که از همه لینک‌ها می‌گذرند باشند و ما ممکن است علاقه‌مند باشیم تنها بسته‌هایی از نوع TCP را که از نود شماره ۰ به نود شماره ۲ رفته‌اند را داشته باشیم. اگر آن خطوط رویدادهایی شامل رشته ’’0 2 TCP‘‘ باشند پس ما مجبوریم تایپ نماییم :

Grep “0 2 TCP “ tr1.tr > tr2.tr

۳-۳ : پردازش فایل های داده‌ای با perl :

PERL حروف ابتدایی عبارت ‘‘ Practical Extraction and Language ‘‘ می‌باشد.

Perl اجازه فیلترکردن آسان و پردازش فایل‌های اطلاعاتی ASCII در یونیکس را می‌دهد. این زبان توسط Lary wall و با ایده اصلی تسهیل وظایف سیستم‌های مدیریت ایجاد گردید. Perl بسیا رشد کرده و امروزه یک زبان همه منظوره و یکی از ابزارهای پرکاربرد برای وب و مدیریت کردن اطلاعات اینترنت می‌باشد. Perl یک زبان مفسری است که کاربران بسیاری دارد. اما اغلب جهت جستجو، استخراج وگزارش مناسب‌تر می‌باشد. برخی از مزایای استفاده از Perl عبارتست از:

- پیاده سازی آسان برنامه‌های کوچک که باید به عنوان فیلترکننده برای استخراج اطلاعات از فایل‌های متنی استفاده شوند بدون تغییر که در بسیاری از سیستم‌های عامل قابل استفاده می‌باشد.
- نگهداری و اشکال زدایی اسکریپت‌های Perl بسیار ساده‌تر از برنامه‌ها در سایر زبان‌های ویژه می‌باشد

- Perl بسیار مشهور است، به طوری‌که بسیاری از اسکریپت‌های Perl در اینترنت موجود می‌باشد.

در این بخش ما مقداری اسکریپت سودمند Perl را ارائه می‌نماییم. اولین مثال در جدول ۳-۵ به طور دینامیک گذردهی اتصال TCP را محاسبه می‌نماید برنامه از پارامتر گذردهی روی مدت‌زمانی تعریف شده توسط معیار ''granularity''، معدل‌گیری می‌نماید. به عنوان ورودی سه آرگومان دریافت می‌نماید: نام فایل trace (به عنوان مثال out.tr)، نام گرهی که ما می‌خواهیم گذردهی ترافیک و اتصال TCP را روی آن چک کنیم وgranulavity .

```
# type :perl throughput.pl<trace file > <required node > <granlarity >  > file
$infile = $ARGV[0];
$tonode =$ARGV[1];
$granularity = $ARGV[2];

#we compute how many  bytes  were transmitted during time interval specified
#by granularity parameter in seconds
$sum =0;
$clock=0;

          Open (DATA , "<$infile ")
              || die "can' t open $infile $! ";
     While ( <DATA > ) {
       @x = split ('   ' ) ;
# column  1 is time
If  ($x [1] - $clock <= $granularity )
{
# checking  if  the  event  corresponds  to a reception
If  ($x eq  $tonode )
{
#checking  if the  packet  type  is  TCP
If  ($x  [4] 'tcp' )
{
            $sum =$sum +$x{5] ;
}
}
}
}
else
{       $throughput = $sum /$granularity ;
        Print   STDOUT  " $x [1]  $ throughput  \ n" ;
```

```
        $clock = $clock +$ranularity ;
        $sum =0
}
}
       $throughput = $sum/$granularity ;
        Print   STDOUT  " $ [1] $throughput \n" ;
        $clock = $clock +granularity ;
        $sum=0 ;

       Close  DATA
exit  (0) ;
```

جدول ۳-۵ : برنامه ای به زبان Prel برای محاسبه گذردهی

۳-۴ : ترسیم نمودار با کمک ابزار gnuplot

Gnuplotیک نرم افزاری است که به طور گسترده و رایگان برای سیستم‌های عامل unix/linux به‌علاوه ویندوز، در دسترس می‌باشد. این ابزار یک دستور help دارد که برای فراگیری جزئیات کارکرد و عملکرد آن می‌تواند مورد استفاده واقع شود. ساده‌ترین راه برای استفاده از Gnuplot تایپ کردن "plot <fn >" است که در آن فایل fn که قرار است ترسیم شود دارای ۲ ستون می‌باشد که بیانگر مقادیر نقاط yوx می‌باشند نقاط می‌توانند توسط یک سبک و قلم متفاوت با دستور Plot "fn " w lines 1 ترسیم شوند.

که البته اعداد متفاوت به جای عدد ۱ می‌تواند قرار بگیرد تا خطوط با قلم‌هایی ایجاد شوند. متناوباً یک شخص می‌تواند از نوع متفاوتی از نقاط استفاده نماید و این کار با نوشتن دستور Plot "fn " W Points 9 انجام می‌شود و با قرار دادن اعداد متفاوتی پس از Points انواع متفاوتی از نقاط را به نمایش خواهد گذارد.

برخی از سایر ویژگی های gnuplot:

برای مثال دستورات زیر را در نظر بگیرید:

```
Set Size  0.6  ,0.6
Set Pont Size 3
Set Key 100,8
Set Xrang [ 90.0 : 120.0]
Plot “ fn1”  w  lines  1  , “fn2 “ w  lines 8 , “ fn3 “ with  Points 9
```

-خط شماره ۱:یک منحنی با اندازه کوچکتر از اندازه پیش فرض ایجاد می‌کند.

-خط ۲ نقاطی را که بزرگتر از اندازه پیش فرض هستند را تولید می‌نماید (البته در هر دو خط می تواند اعداد دیگری استفاده شوند).

-خط ۴ اعداد محور x به بازه ۱۲۰-۹۰ محدود می‌نماید.

-خط ۵ سه تا منحنی را در یک صفحه اضافه می‌کند که از سه تا فایل مجزای fn3, fn2,fn1 گرفته می‌شوند.

-خط ۳ می‌گوید gnuplot دقیقا کجا ”key“ را قرار می دهد. که بخشی از علامت در صفحه‌ای است که شیء ترسیم شده را توصیف می‌نماید. به طور خاص برای هر شیء ترسیم شده نوع خط و نوع نقاط را که مورد استفاده است می‌دهد. به جای یک موقعیت و مکان خاص می‌توانید از لغات کلیدی ”left“، ”right“، ”top“، ”bottom“، ”outside“، ”below“ و غیره استفاده نماید. برای مثال Set Key below (که می گوید کلید زیر نمودار قرار گیرد) یا به طور ساده ” Set nokey“ که به طور کامل کلید را غیر فعال می‌نماید. یا به یاد داشته باشید که نام پیش‌فرض هر شیء که در لغت key ظاهر می‌شود، به طور ساده نام فایل متناظر آن می‌باشد. اگر شما می‌خواهید به یک شیء ترسیمی (نمودار) یک عنوان به غیر از نام فایل متناظر آن بدهید باید نام مذکور را به طور صریح در دستور ”Plot“ قرار دهید. برای مثال :

```
Plot “fn1 “  t  “expectation”  w  lines 1  , “ fn2 “  t  “variance “ w lines  2
```

در اینجا نامهای ‘‘ expectation’’ و ’’variance’’ در کلید (key) ظاهر خواهند شد. اگر دنباله‌ای از دستورات چندین بار استفاده شوند می‌توان آنها را درون یک فایل نوشت فرض کنید نام آن فایل ’’g1.com‘‘ باشد وسپس به سادگی هرگاه که خواستید از آن استفاده نمایید آن را بارگذاری[1] نمایید:

Load “g1.com”

Gnuplot می‌تواند جهت استخراج برخی از ستون‌ها ازیک فایل چند ستونی به شکل زیر مورد استفاده واقع شود:

Plot “queue.tr “

Using 1: ($4/1000) t “kbytes “ w lines , \”queue. tr “ using 1:5 t “Packets “ w lines 2

که به مفهوم ترسیم یک منحنی با استفاده از ستون اول فایل ’’queue.tr‘‘ به عنوان محور x و ستون چهارم تقسیم بر عدد ۱۰۰۰ به عنوان محور y ها می‌باشد و روی همان صفحه ترسیم نمودار منحنی دوم با استفاده از ستون اول برای محور xها و ستون پنجم برای محور yها نیز ترسیم خواهد شد. به خاطر داشته باشید که ترتیب ’’using‘‘ و ’’t‘‘ و ’’Lines‘‘ مهم است.

۳-۵: ترسیم نمودار با کمک ابزار Xgraph

یک برنامه ترسیم سودمند است که توسط ns فراهم شده‌است(گاهی اوقات نیاز است با استفاده از configure)به طور جداگانه کامپایل شود و سپس با دستور Make در شاخه Xgraph آماده اجرا می‌شود. Xgraph اجازه می‌دهد تا با کلیک‌کردن دکمه ’’Hdcpy‘‘ فرمت‌های Postscript، Tgif و سایر فایل‌ها ایجاد شود.

ورودی دستور Xgraph به طور انتظار یکی و یا بیشتر فایل اسکی شامل هر نقطه اطلاعات x-y برای هر خط آن فایل می‌باشد.

[1] - Load

برای مثال Xgraph f2 f1 روی یک صفحه، فایل‌های f1 و f2 را چاپ می‌نماید. برخی از ویژگی-ها در Xgraph عبارتند از :

Title (عنوان): از عبارت ''title'' t– استفاده نمایید.

Size (اندازه):از مدل و عبارت geometry Xsize X ysize–

عنوان برای محورها، ''X''Xtitle'' -وعنوان برای محور y ها به صورت ytitle'' Y–مشخص می‌شود. رنگ متن و شبکه : با پرچم V– مشخص می‌شود.

به عنوان مثال دستور Xgraph ممکن است داشته باشیم:

Xgraph f1 f2 -geometry 800 X400 -t ''Loss rates '' –X''time '' –y ''Lost Packets''

۳-۶ : استخراج اطلاعات از میان یک اسکریپت tcl

می‌توان و ممکن است که دستورات unix مانند ''graph'' و ''awk'' را درون اسکریپت‌های tcl مجتمع کرد. به‌گونه‌ای که شروع پردازش اطلاعات زمانی باشد که فایل در حال نوشته‌شدن است. برای مثال راه محدودکردن فایل‌های trace آن است که از دستورات unix مرتبط به پردازش فایل، از طریق دستورات tcl که فایل مورد درخواست را بازمی‌کند استفاده شود. برای مثال ممکن است دستور

$Set file 1 [open out. tr w]

(که این دستور در شروع اسکریپت ex1.tcl در جدول ۲-۴ داشتیم) توسط دستور زیر جایگزین شود:

Set file 1 [open '' grep \ ''tcp\''>out.tr'' w]

این کد باعث می‌شود که خطوط درون فایل،در حین پردازش برای نوشته شدن، که ''file1'' دستگیره آن فایل می باشد، چک شوند و تنها آن خطوطی که عبارت و کلمه ''TCP'' را دارا می‌باشند در فایل out.tr که حاصل و نتیجه است، نوشته شود و file1 دستگیره این فایل می‌باشد.

فصل ۴ :

توصیف و شبیه‌سازی TCP/ IP

۴-۱ پروتکل(transport contral protocol) TCP:

یک پروتکل لایه انتقال است که مسئول تبادلات بیش از ۹۰٪ ترافیک اینترنت می‌باشد و درک TCP از این جهت دارای اهمیت حیاتی است که منجر به درک اینترنت می‌شود اگر چه TCP پیش از این در سطح وسیعی گسترش یافته است کماکان در حال رشد و نمو و ارتقا می‌باشد. IETF(Internet Engineering task Force) مهمترین سازمان استانداردسازی مرتبط با TCP می‌باشد که برخلاف سایر سازمان‌های استانداردسازی مانندITV و غیره همه استانداردها را به طور رایگان و لحظه به لحظه در اختیار قرار می‌دهد. در ادامه این بخش چند اسکریپت ns که TCP را به کمک شبیه‌سازی بررسی می‌نماید، ارائه می‌شود.

۴-۲ – آنالیز و بررسی مثال ex1.tcl

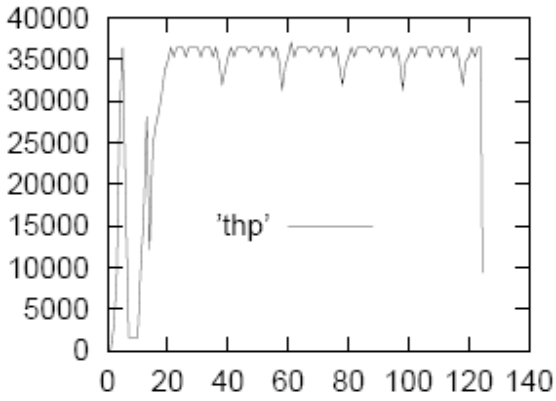
اجازه دهید برنامه (جدول۳-۵) را روی فایل out.tr که توسط اسکریپت ex1.tcl (جدول۲-۴) تولید شده است، اجرا نماییم. برای این منظور باید تایپ نماییم:

```
Perl   throughput.p   out.tr   4   1>thp
```

ما یک فایل خروجی یا متوسط گذردهی دریافتی TCP (بایت برثانیه) را به‌دست می‌آوریم که بر حسب زمان تغییرات مذکور را ثبت می‌نماید. که در این مورد ما، هر یک ثانیه یک مقدار جدید گذردهی به‌دست می‌آید. این فایل خروجی را می‌توان به کمک gnuplot با تایپ‌کردن دستورات زیر ترسیم نمود.

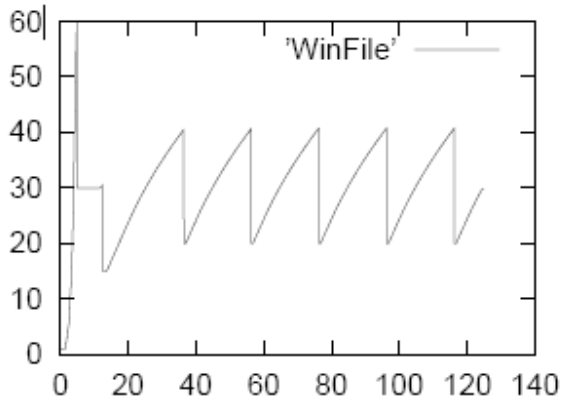
```
Gnuplot
Set size  0.4  , 0.4
Set key 60 , 1500
Plot  “thp”    w  lines 1
```

که نتیجه این دستورات را در جدول ۴-۱ ملاحظه می نمایید.



شکل ۴-۱ :گذردهی اتصال TCP

به منظور درک بهتر رفتار سیستم، ما به همین شکل نمودار اندازه پنجره ارسال TCP را ترسیم می‌نماییم. (شکل ۴-۲). این همان فایل ''Winfile'' می باشد که توسط ex1.tcl ایجادشده است.



شکل ۴-۲: اندازه پنجره ارسال اتصال TCP

از زمان ثانیه ۲۰ به بعد ملاحظه می‌نمایید که یک چرخه منظم و همگام از اتصال TCP بدست می‌آید. علت این چرخه مرتب تکرارشونده آن است که TCP همیشه از یک مکانیزم پیشگیری از ازدحام استفاده می‌نماید و اندازه پنجره ارسال خودش را تقریباً به صورت خطی افزایش می‌دهد تا زمانی که

ازدحام رخ دهد. قبل از زمان ۲۰ یک رفتار ناپایدار و زودگذر از TCP را در فاز شروع آهسته[1] مشاهده می‌نماییم.

در زمان ۴،۲ مقداری از دست دادن بسته ها را در فاز شروع داریم، در نتیجه پنجره در حالی که گذردهی نزدیک به صفر است. چگونه می‌توان این مورد را توضیح داد؟ دلیل آن است که در زمان ۴،۲یک وقفه[2] و فاصله وجود دارد. لذا اگرچه اندازه پنجره ۳۰ بسته است. اما هیچ‌گونه ارسالی وجود ندارد. در زمان ۱۱ نیز یک در زمان شروع یک رخداد از دست دادن بسته‌ها مجدداً وجود دارد.

۴-۳ – TCP روی لینکهای شلوغ[3] و تحت نظرگرفتن صف لینکها :

قبلاً همان‌گونه که ملاحظه کرده‌اید، از دست دادن بسته‌ها به علت ازدحام بود. البته رخداد از دست دادن بسته‌ها می‌تواند به علت وجود لینک‌های معیوب و یا دارای خطا[4] باشد. این موضوع مخصوصاً در مورد لینک‌های رادیویی می‌تواند درست باشد. در تلفن‌هایی که مدل چینش نودهای شبکه آن لانه زنبوری و بافت سلولی[5] می‌باشد و یا در لینک‌های ماهواره‌ای نیز این اتفاق را ممکن است شاهد باشیم. و یا مشکلاتی را از ناحیه تداخلات ضمنی و اتفاقی ممکن است متحمل شویم که می‌تواند بسته‌ها دارای خطاهایی شوند و سپس بسته‌ها حذف شوند. (به علت مواردی نظیر fading،shadowing). در این بخش قصد داریم چگونگی واردکردن یک مدل خطا را نشان دهیم: ما فرض می‌کنیم بسته‌ها روی لینک ارسال (لینک پیش‌رو) بدون هیچ وابستگی به مقادیر ثابت احتمالات در حال حذف‌شدن می‌باشد این مدل خطای لینک که وارد لینک متصل به نودهای n3,n2 (در مثال جدول ۴-۳)می‌شود به شکل زیر ایجاد می‌گردد:

---

[1] - slow - start

[2] - time –out

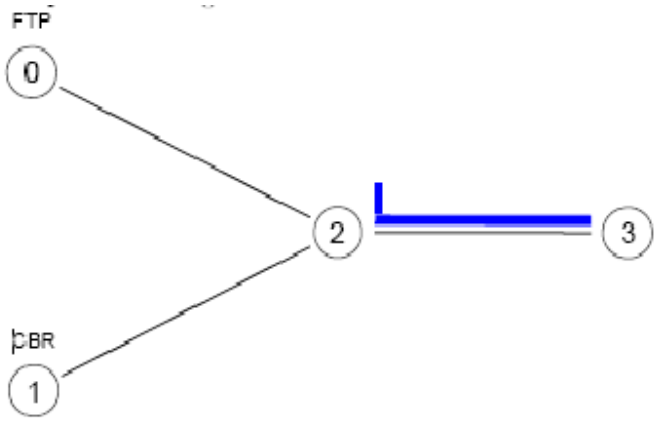
[3] - noisy

[4] - noisely

[5] - cellular

```
# Set error  model  on link n2 to n3
Set loss_module  [ new  Error  model ]
$ loss _module  Set   rate _0.2
$loss _module  ranvar  [ new Random Variable  /Uniform]
$ loss _module   drop-target  [ new  Agent /Null ]
$ ns  loss model  $loss _ module $ n2 $ n3
```

دستور ۰،۲ _rate_ Set _module loss $ یک نرخ از دست‌دادن بسته‌ها به میزان ۲۰٪ را تعیین می‌کند و آن یک مولد متغیرهای تصادفی توزیع یکنواخت را به‌کار می‌گیرد که در خط بعدی تعریف شده است. و خط آخر تعیین می‌کند که کدام لینک تحت تاثیر این تنظیمات قرار گیرد. به عنوان یک مثال از اتصالات TCP که یک لینک گلوگاه معیوب و خطا دارد را با یک اتصال UDP به اشتراک می گذارد، ما تصویر شکل ۴-۳ را در نظر می‌گیریم.



شکل ۴-۳ :مثال rdrop.tcl

- **Queue monitoring**: یک شی مهم در ns، Queue Monitor می‌باشند. این شیئ اجازه می-دهد تا اطلاعات زیاد و مفیدی را در مورد طول صف، بسته‌های ورودی و بسته‌های ارسال شده و بسته‌های از دست‌رفته را جمع‌آوری نماییم جهت پیاده‌سازی یک Queue Monitor بین نودهای n3,n2 باید تایپ نماییم:

Set qmon [$ns monitor-queue $n2 $n3 [open qm.out w] 0.1] , [$ns link $n2 $n3]
queue-sample – timeout , #[$ns link $n2 $n3] start –tracing

شی QueueMonitor دارای ۴ آرگومان است: ۲تای اولی نودهایی هستند که لینکی را که قرار است مانیتور شود، مشخص می نماید. سومی، فایل خروجی نوع trace را مشخص می‌نماید و آخرین پارامتر نوبت زمانی مانیتورکردن صف روی آن لینک را مشخص می‌نماید. در مورد مثال ما، صف روی ورودی لینک نودهای n2-n3 در هر ۰،۱ ثانیه می شود و خروجی روی فایل qm.out ریخته می‌شود فایل خروجی دارای ۱۰ستون زیر می باشد:

۱.زمان

۲. نودهای ورودی وخروجی که لینک را مشخص می‌نماید.

۳.اندازه صف به بایت که این مقدار مرتبط است با متغیر size که متعلق به شی Queue monitor می باشد.

۴. اندازه صف در بسته که مرتبط است با متغیر _pkt شیئ مزبور می باشد.

۵. تعداد بسته های وارد شده (مرتبط با متغیر _Parrivals)

۶. تعداد بسته های ارسال شده (متغیر _pdepartures)

۷. تعداد بسته های حذف شده از صف (_Pdrop)

۸. تعداد بایتهایی که وارد شده‌اند (_barrivals)

۹.تعداد بایتهایی که از لینک ارسال شده‌اند(اشاره می نماید به متغیر (_bdeppartures)

۱۰. تعداد بایتهای حذف شده _bdrops)

راه دیگر جایگزین کردن برای کارکردن مستقیم با این متغیرها در بخش۴-۵ توصیف شده است. مثال بعدی اسکریپت ۴-۱ است که یک اسکریپت کامل را جهت مدل کردن TCP با حذف بسته‌های ناشی از وضعیت خطا را نشان می‌دهد.

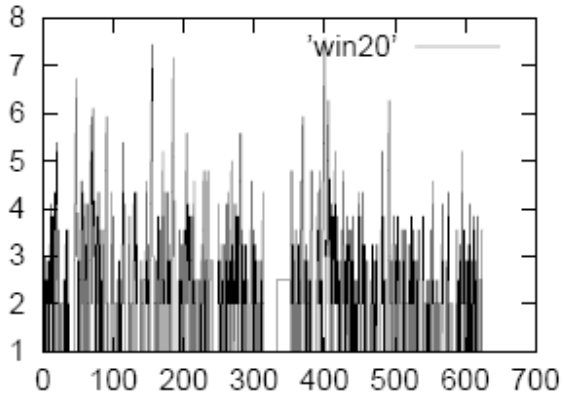
```
# create  the Simulator  instance
Set  ns [new Simulator ]
$ns  color  1 Blue
$ns color  2 Red

# 0pen  the NAM trace file
Set  nf [open out.nam  w]
$ns namtrace –all $nf

#open  the Trace  file
Set tf [open out.tr  w]
Set  wimdowVsTime2 [open WindowVsTimeNReno  w]
$ns  trace –all $tf


#Define  a 'finish'  procedure
Proc finish {}  {
        Global  ns  nf tf
        $ns  flush –trace
        Close $nf
        Close $tf
        Exec  nam out.nam &
        Exit 0
}

#create  four nodes
Set n0 [$ns node ]
Set n1 [$ns node]
Set n2 [$ns node ]
Set n3 [$ns node ]
$ns  at 0.1 “$n1  label \ “CBR\””
$ns  at 1.0 “$n0  label \”FTP\””

#create links between the nodes
$ns duplex-link $n0  $n2 2Mb  10ms DropTail
$ns duplex-link $n1 $n2 2Mb  10ms DropTail
$ns  simplex-link $n2  $n3 0.07Mb  20ms  DropTail
$ns  simplex-link $n3 $n2  0.07Mb  20ms DropTail
```

```
# set Queue Size of link (n2-n3)  to  10
$ns  queue –limit $n2 $n3  10

#Monitor  the queue for  link (n2-n3). (for NAM)
$ns simplex – link –op $n2 $n3 queuePos  0.5

#set  error model  on link n3 to n2.
Set loss_module [ new ErrorModel]
$loss_module set rate_0.2
$loss_module ranvar  [ new Randomvariable/Uniform]
$loss_module drop- target [ new Agent/Null]
$ns lossmodel $loss _module $n2 $n3
# setup a TCPconnection
Set tcp [new Agnet/TCP/newreno]
$ns attacj-agent $n0 $sink
$ns connect $tcp $sink
$tcp set fid_1

#setup a  FTP over TCPconnection
Set ftp [new application/FTP]
$ftp attach-agent $tcp
$ftp set type_FTP

# setup aUDPconnection
Set udp [new agnet/UDP]
$ns attach-agent $n1 $udp
Set null [new agent/Null]
$ns attach-agent $n3 $null
$ns connect  $udp $null
$udp set fid_2

#setup a UDP connection
Set udp [new agent/UDP]
$ns attach-agent $n1 $udp
Set null [new agent/ Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_2

#setup a CBR over UDP connection
Set cbr [new application /traffic/CBR]
#cbr attach-agent  $udp
$cbr set type_CBR
$cbr set packetsize _1000
```

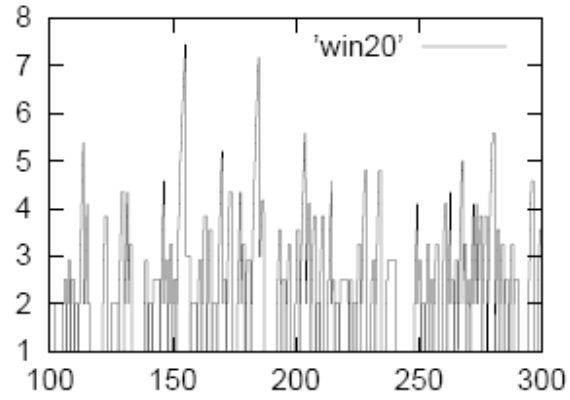
```
$cbr set rate_0.01Mb
$cbr  set random_false

#schedule events for the CBR and FTP agents
$ns at 0.1 “$cbr start “
$ns at 1.0 “$ftp start “
$ns at 624.0 “$ftp stop “
$ns at 624.5 “$cbr stop “

# printing the window size
Proc plotWindow {tcpSource file} {
global ns
set time 0.01
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file “$now $cwnd”
$ns at [expr $now +$time] “plotWindow $tcpSource $file”
$ns at 1.1 “plotWindow  $windowVsTime2”
# sample the bottleneck  queue every 0.1 sec. store the trace in qm.out
Set qmon [ $ns  monitor –queue $n2  $n3 [ open qm.out w ]  0.1] ;
[$ns link  $n2  $n3 ] queue –sample – timeout ; # [$ns link $n2 $n3 ] start –tracing


#Detach tcp  and sink agents (not really necessary)
$ns  at 624.5 “$ns detach –agent $n0  $tcp  ; detach – agent  $n3  $sink “
$ns at 625.0 “finish “
$ns run
```

جدول ۴-۱ : اسکریپت tcl.rdrop برایTCP روی یک کانال نویزدار[1]

در شکل ۴-۴ ما با استفاده از gnuplot فایل تولید شده توسط شبیه‌سازی را بازبینی می‌نماییم. یک تصویر بزرگ‌نمایی شده از این شکل را در تصویر ۴-۵ ملاحظه می‌نمایید:

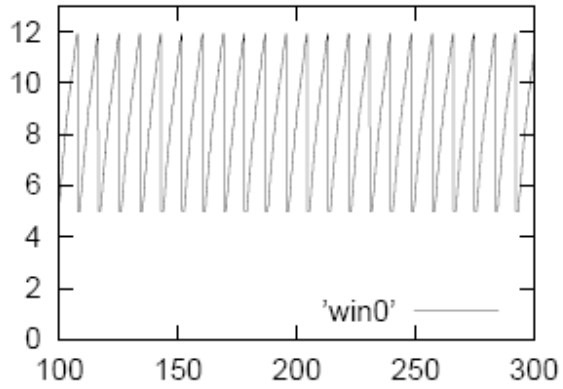


شکل ۴-۴ :اندازه پنجره TCP با ۲۰٪ احتمال بسته به صورت تصادفی



شکل ۴-۵ : اندازه پنجره ارسال TCP با ۲۰ ٪ احتمال بسته به صورت تصادفی (بزرگنمایی شده)

در چنین مورد ما می‌توانیم وقفه زمانی را حتی به صورت طولانی ملاحظه نماییم. به طور خاص در زمان ثانیه ۳۰۰ این موضوع آشکار است. برای آن‌که تاثیر عمیق از دست‌دادن بسته‌ها به طور تصادفی را، روی بازدهی TCP مشاهده نمایید، ما بار دیگر شبیه سازی را بدون هیچ‌گونه رخداد از دست دادن بسته‌ها، اجرا می‌نماییم نتیجه این شبیه‌سازی در شکل ۴-۶ به تصویر کشیده شده است:



شکل ۴-۶: اندازه پنجره ارسال TCP برای نرخ از دست‌دادن بسته‌ها به میزان ۵ ٪

یک روش مهم اندازه‌گیری بازدهی،متوسط گذردهی TCP می‌باشد. یک روش ساده جهت محاسبه آن، جستجوی زمان رسیدن بسته TCP به مقصد (نود شماره ۳) در فایل بازبینی[1] بنام out.tr می‌باشد. در این شبیه‌سازی و با توجه به خط زیر، زمان624.08754 می باشد:

R 624.82754 2 3 tcp 1000 ----------1 0.0 3.0 1562 4350

شماره ما قبل پایانی شماره یکتای بسته می‌باشد ۱۵۶۲امین بسته TCP در مقصدیافت شده است. بنابراین گذردهی TCP به سادگی برابر خواهد بود با این شماره تقسیم بر مدت زمان اتصال FTP که خواهد بود ۶۲۳ ثانیه. لذا گذردهی ۲،۵۰۷ بسته‌برثانیه یا معادل ۲،۵۰۷ کیلو بایت بر ثانیه خواهد بود چون به‌صورت پیش‌فرض بسته‌های TCP در اینجا ۱۰۰۰ بایت حجم دارند یا به‌عبارتی دیگر گذردهی 20058 bps می‌باشد.

تذکر: اگر ما نگاهی به اولین خطوط مایل out.tr بیندازیم ملاحظه می‌نماییم که بسته‌های دیگر TCP با اندازه هر بسته۴۰بایت نیز وجود دارند که بدون شمارش آنها تعداد کل ۱۵۶۲ خواهد شد. یعنی شماره سریال آنها صفر می‌باشند و ما آن بسته‌ها را شمارش نکردیم چون آنان بسته‌های سیگنالی هستند

[1] - trace

که مسئول بازکردن اتصالات TCP می‌باشند. به‌خاطر داشته باشید که ما از نوع متأخر بسته‌های تصدیق[1] مربوط به TCP استفاده کرده‌ایم:

Set sink [new agent/TCP sink /delACK]

به‌جای آنکه به‌طور ساده از دستور

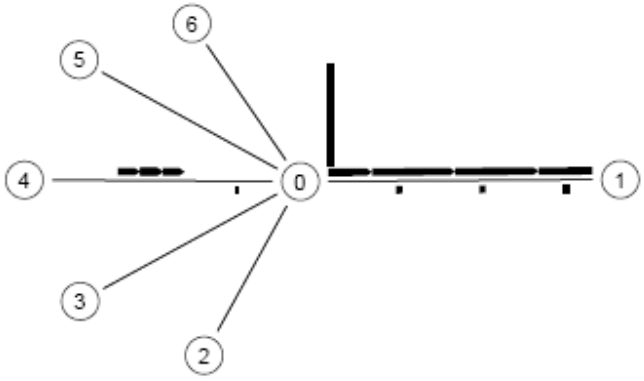
Set sink [new agent/TCP sink]

استفاه نماییم.

۴-۴- ایجاد اتصالات زیاد با خصوصیات تصادفی:

به‌منظور ایجاد بسیاری از اتصالات بهتر است به‌جای آن‌که هر یک را به‌طور جداگانه از قبیل: نود، لینک و اتصال تعریف نماییم آنها را از طریق یک حلقه تکرار و درون یک بردار به TCL تعریف نماییم. علاوه بر این جالب خواهد بود که پارامترهای اتصالات را مانند شروع یا پایان فعالیت آنها و یا تاخیر لینک‌ها و غیره را به‌روش تصادفی انجام دهیم. که از هر ۲ روش در این بخش استفاده نموده‌ایم و سپس یک مثال را ارائه می‌نماییم.

تصویر شکل ۴-۷ را در نظر بگیرید :



شکل ۴-۷ : مثال یک شبکه با چندین اتصال TCP

[1] - Delayed ACK

ما ۵ اتصال و بستر FTP را که بطور تصادفی آغاز بکار می‌نمایند را ایجاد می‌نماییم که آنها بر اساس توزیع یکنواخت زمان شروع به‌کار بین ۰ تا ۷ ثانیه را دارا می‌باشند. و کل زمان شبیه سازی ۱۰ ثانیه می‌باشد. ما لینک‌هایی که دارای تاخیرهایی به‌صورت تصادفی تنظیم شده می‌باشند را ایجاد می‌نماییم که دارای توزیع یکنواخت بین مقادیر 1ms و 5ms می‌باشند. granularity برابر ۰،۰۳ ثانیه، می‌باشد. این اتفاق در روال plotwindow رخ می‌دهد.

به‌خاطر داشته باشید که فایل ''win'' با استفاده از اشاره‌گر window Vs times آدرس‌دهی می‌شود.

و روال برای هر ۵ اتصال به‌طور بازگشتی[1] فراخوانی می‌شود. (شکل ۴-۲)

```
#create  the Simulator  instance
Set  ns [ new  Simulator ]

#Opening   the  trace files
Set nf [open out.nam w ]
$ns namtarce –all $nf

Set  [open out.tr w ]
Set  windowVsTime [open  win w]
Set param [open parameters w]
$ns trace-all $tf

#Define  a ‘finish ‘ procedure
Proc finish {} {
            Global ns nf tf
             $ns flush –trace
             Close $tf
              Exec nam out.nam&
              Exite 0
}

#create bottleneck and dest nodes  and  link between  them
```

[1] - recursive

```
Set n2 [$ns node ]
Set n3[$ns node ]
$ns  duplex –link $n2 $n3 0.7Mb 20ms DropTail


Set NumbSrc  5
Set Duration  10

#Source nodes
For {set j 1 }  {$j <=$NumbSrc } {incr j} {
Set S($j) [$ns node]
}

# create  a raandom  generator for starting the ftp and for bottleneck link delays
Set rng [new RNG]
Srng seed 0

# parameters for random variables for beginning of ftp connections
Set Rvstart [new Random Variable/Uniform]
$Rvstart set min_0
$Rvstart set max_7
$Rvstart use-rung $rng
}
#we define two random parameters for each connection
For{set i1} {$i<=$numbsrc} {incr  i} {
Set startT($i) [expr [$Rvstart  value]]
Set dly ($i) [expr [$Rvdly  value]]
Puts $param “dly($i) $dly ($i) ms”
Puts $param “startT ($i) $startT($i) sec”}
#links between source and bottleneck
For {set j 1} {$j <=$numbsrc} {incr j} {
$ns duplex-link $s($j) $n2 10Mb $dly ($j) ms Droptail
$ns queue-limit $s($j) $n2 100
}
#Monitor the queue for link (n2-n3).(for NAM)
$ns duplex-link-op $n2 $n3 queuepos 0.5
#setQueue sizw of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#TCP sources
For {set j 1} {$j<=$Numbsrc} {incr j} {
Set tcp_src($j) [new agent/TCP/Reno]
}
#TCP Destination
For{set j 1} {$j <=$Numbsrc}  {incr  j} {
```

```
$ns attach-agent $s($j) $tcp_src ($j)
$ns attach-agent $n3 $tcp_snk($j)
$ns connect $tcp_src($j) $tcp_snk($j)
}
#FTPsources
For {set j 1} {$j<=$Numbsrc} {incr j} {
Set ftp ($j ) [$tcp_src($j)  attach-source FTP]
}
#parametrisation of TCP sources
For{set j 1} {$j<=$NumbSrc} {incr j} {
$tcp_src ($j) set packetsize_552
}

#Schedule events for  the FTP agents:
For{set i1} {$i<=$Numbsrc} {incr i} {
$ns at $startT($i) "$ftp($i) start"
$ns at $Duration "$ftp ($i) stop"
}
Proc plotwindow {tcpSource file k} {
Global ns

Set time 0.03
Set now [$ns now]
Set cwnd [$tcpsource set cwnd_]
Puts $file "$now $cwnd"
$ns at [expr $now+ $time]  "plotwindow $tcpSource $file $k"}

#the procedure will now be called for all tcp source
For {set j 1} {$j<=$Numbsrc}  {incr j} {
$ns at 0.1 "plotwindow $tcp_src($j) $window VsTIME $j"
}
$ns at [expr $Duration] "finish"
$ns run
```

جدول ۴-۲: اسکریپت TCL بنام ex3.tcl برای رقابت چندین اتصال TCP

۴-۵- اتصالات TCP از نوع کوتاه:

بخش اعظم ترافیک اینترنت را انتقال فایل‌ها تشکیل می‌دهند. متوسط حجم فایل منتقل شده ۱۰ کیلو بایت می‌باشد.

این بدان مفهوم است که متوسط یک فایل کمتر از ۱۰ بسته TCP نخواهد بود در جائی‌که عموماً اندازه بسته‌های TCP یک کیلوبایت باشد. این بدان مفهوم است که اغلب انتقالات فایل‌ها در فاز شروع آهسته انجام می‌شود. این فایل‌ها را اصطلاحاً ''mice'' می‌نامند.

اما اکثر انتقالات فایل حجم‌های بزرگ را اصطلاحاً ''elephant'' می‌نامند.

یک توزیع عمومی که اندازه فایل منتقل شده را توصیف نماید توزیعی از نوع pareto با پارامتر shape بین ۱و۲ می‌باشد و متوسط اندازه فایل ۲،۵ کیلوبایت می‌باشد.

به یاد داشته باشید که توزیع pareto با میانگین ۱۰ کیلو بایت و اندازه متوسط ۲،۵ کیلو بایت یک توزیع pareto را با پارامترهای shape،$\beta = 1.16$ بایک اندازه حداقل ۱،۳۷ کیلوبایت را تعریف می‌نماید.

توزیع زمان‌های ورود اتصال جدید به طور مکرر به شکل نمایی می‌باشد. در این بخش قصد داریم روش هایی را برای شبیه‌سازی sessionهای کوتاه ارائه نماییم که علاوه بر آن توزیع زمان انتقال مربوط به تعداد اتصالات فعال و گذردهی را نیز اندازه‌گیری می‌نماییم. ما قصد داریم یک توپولوژی شبکه را آن‌چنان‌که در جدول ۴-۳ ملاحظه نمودید در نظر می‌گیریم.

```
Set  ns [new Simulator]

#There are several  sources of TCP sharing a bottleneck link
# and a single destination . Their number is given by the parameter NodeNb

#   S(1)                    ----
#    .                          |
#    .                     ----     ---- N -------- D(1) . . . D(NodeNb)
#    .                          |
#    S (NodeNb)          ----
#Next  file will contain the transfer  time of different  connections
Set Out [open Out .ns w]
# Next file will contain the number of connections
Set conn [ open conn.tr  w]
#Open the Trace file
Set tf [open  out .tr  w]
$ns trace –all $tf

#we  define three files that will be used to trace the queue size ,
# the bandwidth and losses at the bottleneck .
Set qsize  [open queue size .tr  w ]
Set qbw   [ open queuebw .tr   w]
Set qbw   [ open queuelost.tr  w]

# defining the topology
Set  N  [ $ns  node ]
Set  D [$ns node ]
$ns duplex –link  $N  $D 2Mb  1ms DropTail
$ns queue –limit  $N  $D 3000

# Number  of source  node
Set NumberFilows  530

#Nodes  and  links
For  { set j  1 } {$j<=$NodeNb} { incr  j   }  {
Set   S($j)  [ $ns  node]
$ns   duplex – link $S($j) $N 100Mb 1ms DropTail
$ns  queue- limit  $S($j)   $N 1000
}

#TCP  Sources  and Destinations
For  {set   i 1  } { $i <=$NodeNb} {  incr   i  }  {
For  { set  j  1   } { $j <=$NumberFlows} { incr  j } {
Set  tcpsrc ($i,$j)  [ new  Agent /TCP/Newreno]
Set tcp _snk ($i.$j ) [ new Agent /TCPSink ]
```

```
$tcpsrc ($i,$j)  set  window _2000
  }

#connections
For {set i  1 }  { $i<=$NodeNd}  { incr  i }  {
For  {set  j  1}  { $j <=$NumberFlows}  {incr  j }   {
$ns  attach- agent  $S($i)   $tcpsrc($i.$j)
$ns  connect  $tcpsrc($i , $j ) $tcp _snk($i.$j)
   }
 }
#FTP  sources
For { set  i 1}{ $i<=$NodeNb}{   incr  i }  {
For {set  j 1 } {$j<=$N umberFlows} { incr} {
Set ftp ($i.$j)  [ $tcpsrc($i.$j) attach-source FTP]
  }
}

# Generators  for random  size  of files.
Set  rng1 [new  RNG]
$rng1 seed 0
Set  rng2 [ new  RNG]
$rng2 seed  0

#Random  interarrival  time  of  TCP transfers  at  each  source  i
Set  RV [new  Random Variable/Exponential ]
$RV  set  avg_0.045
$RV  use-rng $rng1
#Random Size of files to transmit
Set RVSize   [new RandomVariable/Pareto]
$RVSize   set  avg_ 10000
$RVSize   set  shape_ 1.5
$RVSize   use-rng  $rng2
# We now define the beginning times of transfers and the transfer sizes
# Arrivals of sessions follow a poison process.
#
For { set i 1} {$i <= $NodeNb} { incr i } {
       Set   t [$ns now]
       For {set j 1} {$j <=$NumberFlows} { incr j} {
  # set  the beginning time  of  next  transfer  from source  i
  Set  t [expr  $t +  [ $RV  value ] ]
Set  Conct ($i, $j )  $t

 #set  the  size  of  next  transfer   from  source   i
 Set  Size ($i, $j)  [ expr  [ $RVSize  value ] ]
 $ns  at  $Conct ($i,$j) " countFlows  $i 1"
```

```
} }

#Next is a recursize procedure that checks for each session whether
# it has ended . The procedure calls itself each 0.1 sec (this is
# set in the variable “time”) .
#If a connection has ended then we print in the file $Out
#    * the connection identifiers i and j ,
#    * the start and end time of the connection,
#  * the throughput of the session ,
#  * the size of the transfer in bytes
# and we further define another beginnig of transfer after a random time .
Proc Test { } {
Globar Conct tcpsrc Size NodeNb NumberFlows ns RV ftp Out tcp_snk RVSize set time
0.1
For { set i 1} {$i<=$NodeNb} {incr i } {
For {set j 1 } {$j<=$NumberFlows} {incr j } {

# We now check if the transfer is over
If { [$tcpsrc($i,$j) set ack_] = = [$tcpsrc($i.$j) set maxseq_] } {
        If {[$tcpsrc($i,$j) set ack_] >=0} {
                # if the transfer is over , we print relevant information in $Out
        Puts $Out “$i,$j\t$Conct( $i,$j)\t[expr [ $ns now ] ] \t\
 [ expr ($Size ($i.$j))/ (1000*([expr [$ns now ] ] - $Conct ($i.$j)))]\t$Size ($i,$j)”
   countFlows $i 0
   $tcpsrc($i,$j) reset
   $tcp_snk($i,$j) reset
} } } }
$ns at [expr [ $ns now] +$time “Test”
}

For {set j 1 } {$j <=$NodeNb} { incr j } {
Set Cnts ($j) 0
#The follwing recursive procedure updates the number of coneections
#az a function of time. Each 0.2 it prints them into $conn. This
#is done by calling the procedure with the “sign” parameter equa]
#3 (in with case the “ind” parameter does not play a role) . the
3 proceddure is also called by the test procedure whenever a connection
# from source I ends by assigning the “sign” parameter 0, or when
# it begins , by assigning it 1( I is passed through the “ind” variable) .
Proc countFlows {ind sign } {
Global cnts conn NodeNb
Set ns [Simulator instance]
      If {$sign = = 0 } { set Cnts ($ind) [ expr $Cnts ($ind) – 1]
}elseif {$sign = = 1} { set Cnts ($ind) [expr $Cnts($ind) + 1 ]
} else {
```

```
    Puts – nonewline $Conn "$ns now] \t"
    Set  sum  0
    For {set  j 1 } { $j<= $NodeNb} { incr   j}  {
        Puts – nonewline  $Conn  "$Cnts($j) \t"
        Set  sum  [expr  $sum + $Cnts($j)]
   }
  Puts  $Conn "$sum"
   $ns  at [ expr [ $ns  now  ] + 0.2 ] "countflows  1   3 "
}}

# define a "finish" proceddure
Proc  finish  {}  {
         Global  na  tf  qsize  qbw  qlost
$ns  flush – trace
Close $qsize
Close $qlost
#  Execute Xgraph  to display  the queue  size , queue  bandwidth and  loss rate exec xgraph
queuesize . tr –geometry 800x400 – t "Queue size "-x"secs"-y"# packets" & exec  xgraph
queuesize .tr –geometry 800x400- t "bandwidth" –x "secs " –y "kbps"-fg  white &
Exec  xgraph  queuelost.tr – geometry 800x400 –t "#Packets lost " –x "secs" –y "packets"
&       exit 0
}

#QUEUE MONITORING
Set qfile [ $ns monitor-queue $N $D [open queue .tr w]  0.05]
[$ns link $n $D] queue-sample-timeout;

#the following procedure records queue size, bandwidth and loss rathe
Proc record {} {
Global ns qfile qsize qbw qlost N D
Set time 0.05
Set now [$ns now]

#print the current queue size in $qsize, the current used
# bandwidth in $qbw, and the loss rate in $qloss
$qfile instvar parrivals_ pdepartures_ bdrops_ bdepartures_ pdrops_
Puts $qsize "$now [expr $parrivals_ -$pdepartures_ -$pdrops_]"
Puts 4qbw   "4now [expr $bdepartures_*8/1024/$time]"
Set bdepartures_0
Puts $qlost "$now [expr $pdrops_/$time]"
$ns at [expr $now+$time] "record"
}
$ns at 0.0 "record"
$ns at 0.01 "test"
$ns at 0.5 "countflows 1 3"
```

```
$ns at 20 “finish”
$ns run
```

جدول۴-۳:اسکریپت ShortTcl.tcl برای اتصالات کوتاه TCP

چندین منبع مولد ترافیک یک لینک گلوگاه را به اشتراک می‌گذارند تا به یک مقصد مشترک برسند. تعداد منابع ترافیکی توسط پارامتر’’Nodenb’’ تعیین می‌شود که در این مثال ما ۶ می‌باشد. مبدأ ترافیکی TCP توسط ۲ پارامتر مشخص می‌شوند: گره مبدأ و تعدادsessionهایی که از آن گره نشأت می‌گیرند. برای هر TCP Agent ما یک FTP Application جدید تعریف می‌کنیم. اتصالات TCP جدید وارد شده مطابق بر یک poisson process می‌باشد. بنابراین اتصالات جدید TCP را با استفاده از متغیرهای تصادفی توزیعی نمایی تولید می‌نماییم. لینک گلوگاه دارای پهنای‌باند 2Mbps و تاخیر1msمی‌باشند. ما از یک اندازه پنجره ۲۰۰۰تایی و یک توزیع reno استفاده می‌نماییم. متوسط زمان بین ورود یک TCP session جدید به هر گره در مثال ما، ۴۵ میلی ثانیه می‌باشد.

این بدان مفهوم است که با متوسط۲۲،۲۲ ، session جدید وارد شده به هر گره ضرب در Node Nb (تعداد گرهها) داریم ۱۳۳،۳۳ session بر ثانیه.

ما sessionها را با متوسط اندازه ۱۰کیلو بایت به کمک توزیع pareto با پارامتر shape برابر با ۱،۵ تولید می‌کنیم. لذا نرخ عمومی تولید بیت‌ها خواهد بود:

$133.33 * 10^4 * 8 = 10.67$ Mbps

که همان‌گونه که ملاحظه می‌نمایید از ظرفیت لینک گلوگاه بسیار بیشتر است. بنابراین انتظار داریم که یک اثر طبیعی از ازدحام ظاهر شود. اگرچه TCP ظرفیت و گنجایش اجتناب از ازدحام را در شبکه و به طور خاص روی صف لینک گلوگاه دارا می‌باشد. لذا ازدحام به فرم‌های دیگری ظاهر خواهد شد.

نظارت کردن بر تعداد نشست‌ها sessions :

در زمینه sessionهای کوتاه از نوع TCP ما نه تنها علاقه‌مند به آمارهای بسته‌ها می‌باشیم بلکه آمارهای sessionها را نیز علاقه‌مند به جمع‌آوری هستیم. قصد داریم در برنامه NS یک روال بازگشتی را به نام ''Test'' تعریف نماییم که هر session را چک می‌کند که آیا پایان پذیرفته است یا خیر ؟ این روال خودش را هر یک‌دهم ثانیه فراخوانی می‌نماید که این مقدار از متغیر ''time'' ذخیره گردیده است. اگر این اتصال پایان یافته‌باشد موارد ذیل را در یک فایل خروجی می‌ریزد:

- شناسه‌های I , g را بر هر اتصال که حاکی از g امین اتصال از نود I ام می‌باشد
- زمان شروع و خاتمه اتصال
- گذردهی آن اتصال
- اندازه انتقال فایل آن اتصال به بایت

آنگاه روال شروع انتقال بعدی را پس از یک زمان تصادفی تعریف می‌نماید. فایل خروجی در این اسکریپت Out.ns خواهد بود. برای این‌که چک کنیم که یک session پایان یافته است یا خیر از دستور زیر استفاده می نماییم

If { [$tcpsrc ($i,$j) set ack_]==[$tcpsrc($i,$j) set maxseq_]} {

برنامه دیگر بازگشتی به نام ''CountFlows''جهت به‌روز کردن تعداد اتصالات فعال از هر نود استفاده می‌شود (که درون آرایه ''cnts'' ذخیره شده و عضو j ام تعداد اتصالات در جریان از نود j ام را نشان می‌دهد) این زیرروال ۲ پارامتر دارد ''ind'' و ''sign''.

''ind'' مشخص می‌نماید که زیر روال چه کاری را باید انجام دهد: اگر ۰ باشد خاتمه را برای اتصال فراخوانی می‌نماید. این پارامترها در زمان آغاز به کار و یا خاتمه یک اتصال وقتی زیربرنامه

فراخوانی می‌شوند، استفاده می‌شوند این زیرروال خودش را مکرراً در هر ۰،۲ ثانیه فراخوانی می‌نماید تا تعداد فراخوانی‌های فعال را درون یک فایل به نام con.tr ثبت نماید. برای این‌که این کار صورت پذیرد باید پارامتر ''sign'' که ارسال می‌شود نه مقدار ۱ باشد و نه ۰ (که ما در این‌جا آن را به مقدار ۳ تنظیم کرده‌ایم)

مانیتور کردن صف:

در برنامه tcl که در ادامه خواهد آمد یک راه جایگزین جهت مانیتورکردن صف که مقداری پیچیده‌تر از آنچه در بخش ۴-۳ دیدیم می‌باشد، ارائه شده است ما مجدداً از دستورهای

```
Set qfile [$ns monitor-queue $N $D [open queue.tr w] 0.05]
[$ns link $N $D] queue-sample-timeout;
```

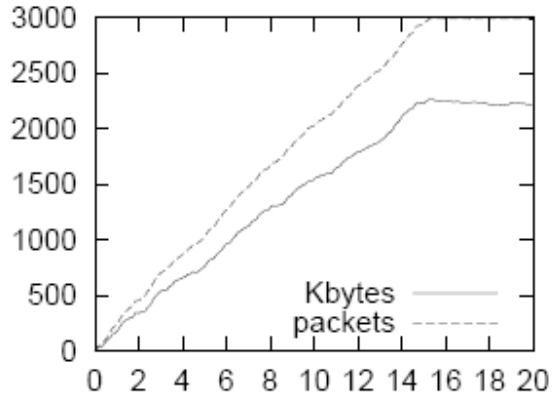
استفاده می‌کنیم.

البته می‌توانیم خط دوم را حذف نماییم و مستقیماً با صفات ''queue-monitor'' که در بخش ۴-۳ توصیف شد، کار خواهیم کرد.

این اتفاق دریک روال به نام ''record'' انجام می‌پذیرد که خودش را در هر ۰،۰۵ ثانیه به طور بازگشتی فراخوانی می‌نماید. به طور مثال پهنای‌باند مورد استفاده صف را به کیلوبایت بر ثانیه درون یک فایل با تقسیم کردن حجم ارسالی از صف به زمان این رخداد ثبت می‌نماییم.

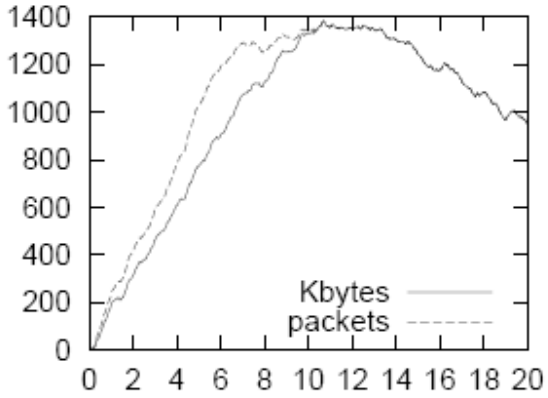
به یاد داشته باشید که Queue Monitor میزان کل تعداد بایت‌های وارد شده را در متغیر _ bdepartures نگهداری می‌نماید برای نگه‌داشتن حساب تعداد بایت‌های ارسال شده دریک زمان خاص باید مقدار متغیر_ bdepartures رادر انتهای محاسبه پهنای‌باند به مقدار صف تنظیم نماییم.

مقدارsessionهای تولید شده (به ازای هر مبدأ ترافیکی 530 تا) ضمانت می‌کند تا ورودی‌ها از همه نودها تا پایان شبیه‌سازی ادامه یابد.



شکل ۸-۴ : اندازه صف برای مثال short-Tcp.tcl

وقتی که این اسکریپت را اجرا می‌نماییم اندازه صف را به کیلو بایت و بسته اطلاعاتی آن‌چنان‌که در تصویر ۴-۸ همچنین با یک تعداد کمتر از sessionها (۱۳۰ تا به ازای هر نود) شبیه‌سازی را اجرا نموده و اندازه صف را به کیلوبایت و به بسته در تصویر ۴-۹ ملاحظه می‌نمایید.



شکل ۹-۴ :اندازه صف برای مثال short-Tcp.tcl جایی که ما تعداد session ها را محدود نموده‌ایم

اندازه صف برای مثال short-Tcp.td برخی مشاهدات را در اینجا باید در نظر گرفت:

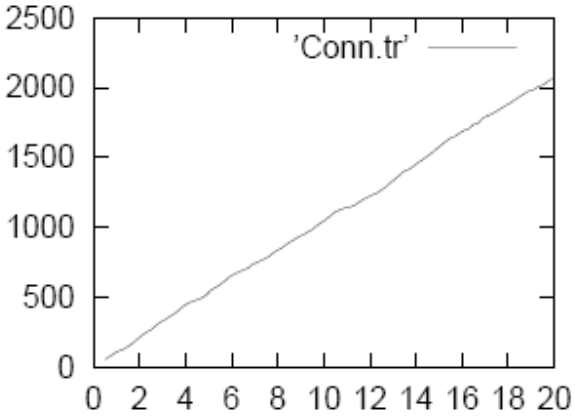
۱. در هر دو نمودارها تعداد بسته‌های درون صف بیشتر از تعداد و حجم صف‌بندی شده می‌باشد. این ممکن است عجیب به نظر آید چرا که یک بسته TCP اندازه یک کیلوبایتی دارد. دلیل آن است

که تعداد زیادی از sessionها خیلی کوچک هستند (۳بسته یا کمتر) بنابراین تعداد بالاسری بسته‌ها که ۴۰بایتی می‌باشند قابل توجه خواهد بود این بسته‌ها در ابتدای ایجاد یک اتصال TCP فرستاده می‌شوند. همه این بسته‌ها را باید در نظر گرفت و همه تعداد بیشتر بسته‌های بالای کیلوبایت‌ها را نیز باید محاسبه نمود.

۲. در شکل ۴-۸ مشاهده می‌شود که اندازه صف در۳۰۰۰ثابت می‌ماند این اندازه صف ماکزیمم است که بدان رسیده است. از این لحظه به بعد رخداد اتلاف و حذف بسته‌ها در صف را خواهیم داشت

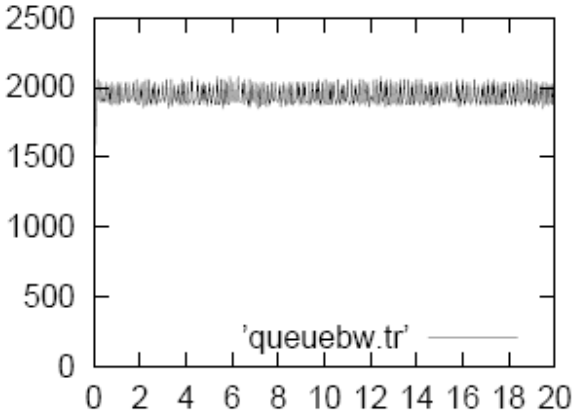
۳. نظر به اینکه همیشه تعداد بسته‌ها بیشتر از تعداد کیلوبایت‌ها می‌باشد که در صف قرار دارند،(شکل ۴-۸) ما ملاحظه می‌نماییم در شکل ۴-۹ پس از گذشت مدتی تعداد بسته‌ها موافق با تعداد کیلوبایت‌ها می‌شود و ۲ نمودار به هم متمایل می‌شوند. در این لحظه تمام بسته‌های درون صف همگی بسته‌های اطلاعاتی TCP می‌باشند و هیچ بسته ۴۰ بایتی که متناظر به ابتدا و شروع session باشند، وجود ندارد. این ناشی از این حقیقت است که ما تعداد sessionها را برای هر نود به ۱۳۰ تا محدود کردیم.

۴. اگر ما نرخ خروجی لینک گلوگاه را از نرخ تولید اطلاعات کم کنیم، ما بیشتر از مقدار اطلاعات صف‌بندی شده در صف گلوگاه را بدست می‌آوریم. علت آن است که اطلاعاتی وجود دارد که در بافر فرستنده ذخیره گردیده و در حین شبیه‌سازی ارسال شده است. در ادمه در تصویر ۴-۱۰ نمودار افزایشی تعداد اتصالات در جریان را روی سیستم مشاهده می‌نمایید:



شکل ۴-۱۰ :تعداد اتصالات

و پهنای‌باند مصرفی گلوگاه را نیز در شکل ۴-۱۱ خواهید دید.



شکل ۴-۱۱ : پهنای‌باند مصرف‌شده در لینک گلوگاه

۴-۶-ابزارهای پیشرفته مانیتورینگ :

در بخش ۴-۵ به طور متناوب ما خاتمه یافتن هر session از نوع TCP را با مقایسه شماره توالی تصدیق در جریانی با شماره توالی حداکثر اتصال، چک کردیم. این روش بررسی بسیار هزینه‌بر است. ما ۲ روش جایگزین مانیتورکردن را معرفی می‌نماییم:

۱. اولین روش تعریف اعمالی است که در صورت خاتمه، انجام پذیرند که این اعمال درون یک زیرروال به نام ''done'' که به طور خودکار وقتی یک اتصال خاتمه می‌یابد، فراخوانی می‌شود. شناسه اتصالی که پایان می‌یابد به علاوه سایر خصوصیات اتصال مانند زمان شروع و غیره که بتوانند وضعیت اتصال را تعریف نمایند، می‌توانند مورد استفاده قرار بگیرند. این روش درون اسکریپت short TCP.tcl در جدول ۴-۴ ارائه شده است.

```
Set ns [new simulator]

# there are severalsources each generating many TCP sessions sharing a bottleneck
#link and a sungle destination. Their number is given by the parameter nodeNb
#         S(1)              -----
#          .                      |
#          .                  ----  ----- N  ----------- D(1) … D(NbdeNb)
#          .                      |
#         S(nodeNb)      -----

# next file will contain the transfer time of different connections
Set out [open out.ns w]
#next file will contain the number of connections
Set conn [open conn.tr w]
#open the trace file
Set tf [open out.tr w]
$ns trace-all $tf
#defining the topology
Set N [$ns node]
Set D [$ns node]
$ns duplex-link $N $D 2 mb 1 ms droptail
$ns queue-limit $N $D 3000

#number  of  flows  per  source  node
Set number flows 530
$nodes and links
For {set j 1} {$i<= $nodenb} {  incr j } {
Set S($j) [$ns node]
$ns duplex-link $S($j) $n 100mb 1ms  droptail
$ns queue-limit $ S($j) $N 1000  }

#TCP sources, destinations, connections
For {set  I  1} {$i <=$NodeNb} {incr I } {
For {set j   1} {4j<=$numberflows} {incr j } {
Set tcpsrc ($i ,$j) [new agent/TCP/newreno]
Set tcp_snk(4i,$j) [new agent/tcpSink]
$tcpsrc ($i,$j) set window_2000
$ns attach-agent $S($i) $tcp_sink($i,$j)
$ns attach-agent $d $tcp_snk($i,$j)
$ns connect $tcpsrc ($i,$j) $tcp_snk($i,$j)
Set ftp($i,$j)  [$tcpsrc ($tcpsrc($i,$j) attach-source FTP]
} }

 # Generators  for random  size  of  files
```

```
Set  rng 1 [new  RNG]
$rng 1 seed  0
Set rng 2 [new RNG ]
$rng 2 seed 0

#Random  inter –arrival time  of TCP transfer  at each source  i
Set RV [new RandomVariable / Exponential ]
$RV set avg  _0.045
$RV use – r ng $ r ng 1

#Random  size  of files  to  transmit
Set  RVSize [new RandomVariable/Pareto]
$RVSize set avg _ 10000
$RVSize set shape _ 1.5
$RVSize use –r ng $rng2

# We now  define  the beginning times of transfers  and  the transfer  sizes
# Arrivals  of  session follow  a Poisson .
#
For  { set i  1 } { $j<=$NumberFlows} { incr } {
       Set  t [ $ns  now ]
      For {set  j  1  } { $j<= NumberFlows } { incr  j } {
# set  the  beginning  time  of  next  transfer  from  from source  and  attributes
Set  t [expr  $t + {$RV  vslue ] ]
$tcpsrc($i,$j )  set  starts  $t
$tcpsrc  ($i,$j )  set  sess  $j
$tcpsrc  ($i,$j )  set  node  $i
           $tcpsrc  ($i,$j )  set  size  [ expr  [ $RVSize value ]]
 $ns  at  [ $tcpsrc ($i,$j)  set  starts ] "$ftp ($i,$j) send {$tcpsrc ($i, $j ) set  size ] "

#  update  the number  of  flows
  $ns  at  [$tcpsrc ($i,$j)  set starts ] " countFlows  $i  1 "
}}
For { set  j  1} {$j <= $NodeNb} { incr  j } {
Set Cnts( $j ) 0
}
# The following   procedure  is  called  whenever a connection  ends
Agent/ TCP  instproc  done  {}  {
Global tcpsrc  NodeNb NumberFlows  ns  RV  ftp Out tcp _ snk RVSize
#  print  in  $Out : node . session ,  start  time , end  time , duration ,
# trans- pkts, transm-bytes , retrans-butes , throughput
Set duration [ expr  [$ns  now ] -  [$self  set  starts ] ]
Puts  $Out " [$self  set node ] \t  [$self  set sess ] \t  [$self  set starts ] \ t \
           [$ns now] \t $duration \t  [$self  set  ndatapack _] \ t \
           [ $self  set ndatabutes _]\t [$self set  nrexmitbytes_ ] \ t \
```

```
        [ expr  [ $self set ndatabytes_ ] /$duration “
        countFlows [$self  set node ] 0
}

# The following  recursive procedure updates  the number  of connections
# as a function  of time . Each 0.2 sec  it  them into $ns Conn . This
# is done by calling  the procedure  with “sign “ parameter equal
# 3 ( in which case the “ind “ parameter does  not play  a role ) . The
# procedure  is also called by the “done” procedure whenever  a connection
# from source i  ends  by assigning  the “sign” parameter 0 , or when
# it begins  , by assigning  it 1 ( i  is passed through  the “ind “ variable ) .
#
Proc  countFlows  {ind  sign } {
Global  Cnts  Conn  NodeNb
Set ns  [Simulator  instance ]
        If { $sign = = 0 }  { set  Cnts ($ind ) [ expr $Cnts { $ind ) – 1 ]
 } elseif { $sign = = 1 } { set Cnts ($ind) [expr  $Cnts ($ind) + 1  ]
 } else  {
    Puts – nonewline  $Conn “[$ns  now ] \t”
     Set  sum  0
 For {set  j  1}  { $j <=$NodeNb } { incr j } {
  Puts – nonewline  $Conn  “ [$ns  now ] \t”
  Set  sum  0
 For  {set j 1 } { $j<= $NodeNb }{ incr j } {
       Puts – nonewline  $Conn  “$Cnts{$j) \t”
       Set  sum [expr $sum  + $Cnts ($j) ]
}
Puts  $Conn “$sum “
$ns at  [ expr [ $ns  now ]  + 0.2 ] “ countFlows  1   3”
} }

Define  a “ finish “ procedure
Proc finish { } {
          Global  ns  tf
Close  $t f
        $ns  flush –trace
        Exit  0
}

$ns  at 0.5  “countFlows  1  3”
$ns at 20 “finish”

$ns run
```

جدول ۴-۴ : اسکریپت *short TCP.tcl* برای اتصالات کوتاه *TCP*

۲. راه دیگر می‌تواند استفاده ازمانیتور به ازای هر جریان باشد (Per-flow). که می تواند آمارهایی را که روی هر جریان رابا اطلاعاتی نظیر: مقدار بسته های ارسال شده و بایت ها و بسته‌های از دست داده شده وغیره به ما ارائه دهد.

تعاریف وضعیت اتصالات TCP در اسکریپت مشابه همان روشی که جهت تعریف کردن اندازه ماکزیمم پنجره ارسال TCP، حدآستانه آغاز فاز slow-start شروع آهسته وغیره، استفاده کردیم می‌باشد.

در اسکریپت ما زمان شروع session، شناسه نود و session و اندازه انتقال را به صورت زیر تعریف نموده ایم:

```
$tcpsrc ($i,$j) set starts $t
$tcpsrc ($i,$j) set sess    $j
$tcpsrc ($i,$j) set node  $i
$tcpsrc ($i,$j) set size [expr [ $RVSize  value ]]
```

زیر روال "Done" به صورت زیر تعریف می‌شود و جایگزین زیر روال Test در روش گذشته در اسکریپت shory TCP.tcl (جدول ۴-۳) می‌شود:

```
Agent/TCP instproc done {}{
Global tcpsrc nodenb numberFlows ns RV ftp out tcp_snk RVsize
#print in $out :node,session, start time, endtime,duration,
#trance-pkts,transm-bytes , retrans-bytes, throughput
Set duration [expr [$ns now] –[$self set stars] ]
Puts $out "[$self set node ]  \ t [$self set sess]  \t  [$self set start]  \t\
        [$ns now] \ t $duration \ t[$self set nrexmitbytes_]  \t\
        [$self set ndatabytes_]    \t [$self set nrexmitbytes_]   \ t\
         [expr [$self set ndatabytes_] /$duration ]"
            Countflows  [$self set node ] 0
}
```

به خاطر داشته باشید که ما سایر وضعیت اتصالات TCP را استفاده می‌نماییم:

Ndatapack: تعداد بسته‌های منتقل شده بوسیله اتصال است (اگر بسته‌ای چندین بار مجدداً ارسال شده باشد،یک‌بار شمرده می‌شود)

Ndata bytes : تعداد بایت‌های اطلاعات ارسال شده توسط اتصال است.

- Ndatabyte _is the number of data bytes transmited by the connection.
- N rexmitpackets _ is the number of packets retransmited by the connection.
- N rexmitbytes _is the number of bytes retransmited by the connection.

_ Nrexmitpackets :تعداد بسته‌ها مجدداً ارسال شده توسط اتصال است.

_ Nrexmitbytes :تعداد بایت‌های مجدداً ارسال شده توسط اتصال است.