شبیه سازی شبکه

به کمک

NS2

تهیه و تنظیم:

محمد (فرزاد) یزدی

mfyazdi@gmail.com

ویرایش اول- بهار ۱۳۸۹

mfyazdi@gmail.com

فهرست مطالب

مقدمه

شبیه ساز شبکه (نسخه ۲) که با نام NS2 شناخته شده است، بطور ساده یک ابزار شبیه سازی رخداد–گسسته است که در مطالعه طبیعت دینامیکی شبکه های ارتباطی، مورد استفاده قرار میگیرد. به کمک NS2 شبیه سازی پروتکلها و عملکردها (مثل TCP و UDP و الگوریتمهای مسیریابی و ...)ی شبکه های سیمی و شبکه های بی سیم بخوبی قابل انجام هستند. همچنین امکان تغییر و تعریف پروتکلهای شبکه ای و رفتار متناظر آنها توسط کاربران فراهم شده است.

انعطاف پذیری و طبیعت ماژولار این شبیه ساز، سبب محبوبیت و استفاده همیشگی آن در مجامع تحقیقاتی شبکه (از زمان ایجاد آن در سال ۱۹۸۹،) شده است. از آن موقع این ابزار دستخوش تغییرات و تجدید نظرهای سازنده ای شده است که دست اندرکاران این زمینه سهم قابل توجهی در آن داشته اند. از جمله این دست اندرکاران میتوان به موارد زیر اشاره کرد:

- دانشگاه کالیفرنیا [۱] و دانشگاه کورنل[۲] که شبیه ساز شبکه **واقعی**[۳] را ایجاد کردند، و این شبیه ساز ، سنگ بنای NS محسوب میشود.
- آژانس DARPA [۴]، که از سال ۱۹۹۵، و از طریق پروژه VINT [۵] ، توسعه NS را پشتیبانی کرد.
- بنیاد ملی علوم [۶] (ایالات متحده) ، که اخیرا به جمع توسعه دهندگان NS پیوسته است.
- ISI[۷]، که در حال حاضر توسعه NS2، توسط این انجمن علمی صورت میگیرد.
- و در آخر (و نه به لحاظ درجه اهمیت)، محققان و توسعه دهندگانی که در مجامع مختلف با فعالیتهای مداوم خود، NS2 را پر قدرت و فراگیر نموده اند.

همانطور که گفته شد، درحال حاضر توسعه NS2 بوسیله ی ISI انجام می شود و توسط DARPA و NSF پشتیبانی می شود.

NS2 اساسا برای کار در محیط لینوکس ساخته شده است، اما میتوان به کمک برنامه های کمکی مثل cygwin [۸](که یک محیط مجازی از سیسم عامل لینوکس ایجاد میکند) آن را روی سیستم عامل windows نیز نصب نمود. جزئیات نصب در ویندوز، بصورت یک فایل ویدیویی تهیه شده است.

۱- University of California

۲- Cornell University

۳- REAL Network Simulator: در اصل به عنوان ابزاری برای مطالعه رفتار دینامیکی جریان و طرحهای کنترل تراکم (congestion) در شبکه های داده ای سوئیچ–بسته (packet-switched data networks)، پیاده سازی شده بود.

۴- Defense Advanced Research Projects Agency

۵- Virtual InterNetwork Testbed : توسط آژانس DARPA و با هدف ایجاد یک شبیه ساز شبکه که بتواند مطالعاتی در زمینه پروتکلهای مختلف و شبکه های ارتباطی انجام دهد، ایجاد شد.

۶- NSF: National Science Foundation

۷- Information Science Institute

۸- یک محیط شبه لینوکسی (Linux-like) برای ویندوز است و شامل دو قسمت است، یک قسمت، DLL که مثل API لینوکس عمل میکند و قسمت دیگر مجموعه ای از ابزار برای فراهم کردن ظاهر و کارایی لینوکس. این نرم افزار محصول شرکت red hat و با شعار cygwin=Cygnus+GNU+windows میباشد. [مرجع: http://cygwin.com]

معماری و ساختار NS2

NS2 بر پایه دو زبان بنا شده است: یکی شبیه ساز شی گرا که به زبان C++ نوشته شده و دیگری مفسر [۱] OTcl [۲] که برای اجرا کردن دستورات برنامه های کاربران استفاده می شود. C++، مکانیسم داخلی اشیاء شبیه سازی را تعریف میکند (در پشت صحنه!) و OTCL بوسیله اسمبل و پیکربندی اشیاء، شبیه سازی را برپا میسازد (روی صحنه!).

تذکر: قطعات NS2 عبارتند از خود شبیه ساز NS و NAM [۳] که انیماتور شبکه بوده و باعث بصری سازی [۴] خروجی ns می شود.

حال ممکن است این سوال مطرح شود که چرا NS از دو زبان برای شبیه سازی استفاده میکند؟ در پاسخ به این سوال باید گفت:

از یک طرف، در شبیه سازی هایی که در آنها پروتکلها با جزئیات زیاد پیاده سازی میشوند، زبان برنامه نویسی ای لازم است که بتواند بایتها، بسته ها و سرآیندها را بطور مطلوب دستکاری کند و الگوریتمهایی را پیاده سازی کند که قرار است روی حجم زیادی از داده ها اجرا شوند. در این حالت سرعت زمان اجرا اهمیت بیشتری نسبت به زمان پروسه تغییر و اجرای مجدد (که شامل اجرای شبیه سازی، یافتن خطاها، رفع خطاها، کامپایل مجدد و اجرای مجدد است،) دارد.

از طرف دیگر، حجم عمده ای از تحقیقات شبکه ای، شمار اندکی از پارامترها یا پیکربندیها را در بر دارند و به سرعت چند سناریو را کاوش میکنند. در این حالت زمان تکرار مجدد (یعنی تغییر مدل و اجرای دوباره آن) بسیار مهمتر است. از آنجایی که در این حالت، پیکربندی اولیه فقط یکبار (و در زمان آغاز شبیه سازی) اجرا میشود، زمان اجرا اهمیت کمتری خواهد داشت.

NS هر دوی این نیازها را به کمک دو زبان برنامه نویسی برآورده میکند. C++ زمان اجرای بالایی دارد حال آنکه در اعمال تغییرات کند است؛ بنابراین برای پیاده سازی پروتکلهای با جزئیات زیاد مناسب است. در مقابل، OTCL بسیار کندتر اجرا میشود اما در عوض در اعمال تغییرات بسیار سریع است و بصورت تعاملی عمل میکند؛ بنابراین برای پیکربندی شبیه سازی کارآیی ایده آلی دارد.

همانطور که در شکل ۱-۱ ملاحظه میشود، کد شبیه سازی که به زبان TCL نوشته میشود به کمک دستور ns (دستور قابل اجرای NS2) اجرا میشود. خروجی، فایل Trace (با فرمت tr.) است که در بیشتر موارد از آن برای رسم نمودار یا متحرک سازی (انیمیشن) شبیه سازی استفاده میشود. در مورد تمامی این موارد در قسمتهای بعدی توضیحات کاملتری ارائه خواهد شد.

C++ و OTCL با استفاده از TclCL به هم پیوند میخورند. یعنی در واقع TclCL (TCL به همراه کلاسهایش[۵]) واسط میان C++ و OTCL است؛ به عبارت دیگر، امکان استفاده اشیاء و متغیرها را در هر دو زبان فراهم میکند. به این ترتیب، OTCL میتواند از اشیاء کامپایل شده در C++ استفاده کند و اشیاء OTCL با هر شیء متناظرشان در C++ مطابقت می یابند.

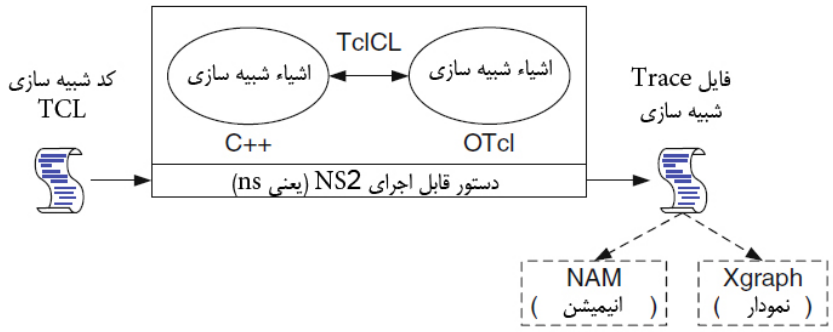
۱- Interpreter

۲- Object Oriented Tool Command Language: تعمیم یافته شی گرای زبان TCL که در سال ۱۹۹۷ توسط David Wetherall از دانشگاه MIT ایجاد شده است

۳- Network Animator

۴- Visualization

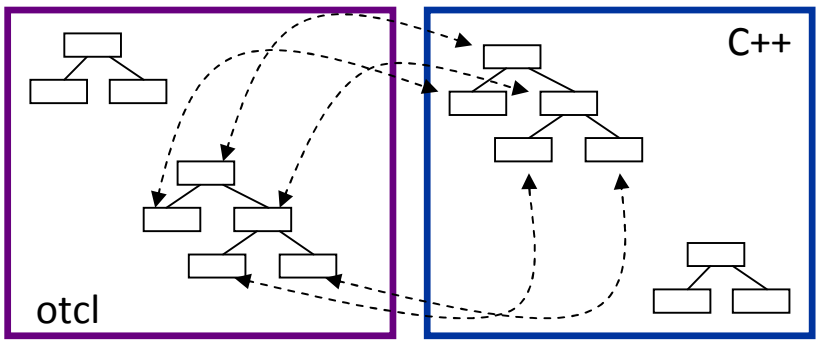
۵- TCL with classes: یک لایه چسبی از C++ روی OTCL ایجاد میکند



شکل ۱-۱: معماری کلی NS2

در اینجا دو کلاس زنجیره ای (سلسله مراتبی) وجود دارد: یکی زنجیره C++ کامپایل شده و دیگری زنجیره Otcl تفسیر (ترجمه) شده که یک تناظر یک به یک میان این دو برقرار است. (شکل ۲-۱)

طراحی شبیه ساز در واقع با جداسازی « داده[۱] » و « کنترل[۲] » صورت می گیرد. یعنی C++ برای داده (شامل پردازش در-بسته[۳]، هسته[۴] NS2، تسریع در اجرا، حاوی جزئیات مفصل ، کنترل کامل) و OTCL برای کنترل (شامل پیکربندی[۵] سناریوی شبیه سازی ، عمل متناوب یا عمل فعال شده[۶]، دستکاری اشیاء موجود C++، تسریع در تغییر و نوشتن).



شکل۲-۱: اشیای مجزای C++ و Tcl

زنجیره C++ کامپایل شده، امکان بازدهی بهتر و اجرای سریعتر شبیه سازی را فراهم میکند. این امر، به ویژه برای سنجش عملکرد پروتکلها و از طریق کاهش بسته ها[۷](ی داده) و کاهش زمان پردازش رخداد[۸]، موثر است.

در کد OTcl میتوان یک توپولوژی شبکه ای[۹] مشخص را پیاده سازی نمود که در آن رفتار انواع پروتکلها[۱۰]، کاربردها[۱۱]، گره[۱] ها و... قبلا در کلاسهای C++ تعریف شده اند.

۱- Data
۲- Control
۳- Per Packet Processing
۴- Core of ns
۵- configuration
۶- triggered or Periodic action
۷- Packet
۸- event processing time
۹- Network Topology: شکل و نحوه استقرار شبکه که شامل وسایل شبکه، محل قرار گرفتن آنها و تکنولوژی انتقال می باشد.
۱۰- Protocol
۱۱- Application

در NS2، به عنوان یک شبیه ساز رخداد–گسسته [۲]، پیشرفت زمان بستگی به تنظیم رخداد[۳]هایی دارد که به وسیله زمانبند [۴] کنترل میشوند. رخداد، یک شیء در C++ است که یک شناسه (ID) منحصر بفرد دارد و به وسیله زمانبند و یک اشاره گر شیء، کنترل میشود.

کارکردهای NS2

بطور کلی کارکردهای **NS2** عباتند از:

شبیه سازی شبکه های باسیم یا بی سیم یا ماهواره ای، مسیر یابی[۵] ، انتقال (از طریق TCP ، UDP، Multicast، Unicast)، منابع ترافیکی (Telnet, FTP, Web)، ضوابط صف بندی، کیفیت خدمات[۶] (Diffserv, Intserv)، شبکه های حسگر [۷]، ردیابی[۸] اطلاعات شبیه سازی شبکه، تولید آماری، تولید اعداد تصادفی و نمونه سازی برای سیستم های باسیم.

NS2 برای شبکه های بی سیم، امکان مسیریابی ویژه (غیر عمومی [۹]) و تعریف IP سیار [۱۰](یا گره سیار [۱۱]) را فراهم می کند.

مزایا و معایب NS2

مهمترین مزایای NS2 عبارتند از:

- قابلیت پیکر بندی آسان به دلیل استفاده از دو زبان برنامه نویسی مختلف (OTCL, C++)
- بسیاری از پروتکل ها قبلاً در آن پیاده سازی شده اند
- مدل های در دسترس بی شماری دارد
- مدل های سیار واقع گرایانه ای ایجاد میکند
- بهترین گزینه برای کسانی است که علاقمند به سطح دقت بالا در لایه های فیزیکی (PHY) هستند
- در شبیه سازی های بزرگ مقیاس (که معمولاً در این موارد از شبیه سازی مرحله ای استفاده می شود) کاربرد دارد
- قابلیت استفاده در شبیه سازیهای موازی [۱۲]
- بسیار شناخته شده است

۱- Node

۲- Discrete- Event Simulator

۳- event

۴- Scheduler

۵ - Routing

۶- QOS: Quality Of Services

۷- sensor network

۸- trace

۹- Ad hoc routing

۱۰- Mobile IP

۱۱- Mobile Node

۱۲- در این نوع شبیه سازیها، دو (یا چند) شبیه ساز مختلف با هم ترکیب شوند و بطور همزمان دو نمونه از برنامه شبیه سازی سری (Serial) معمولی را روی پردازشگر های مختلف اجرا می کنند. این نمونه ها مستقل از یکدیگر اجرا می شوند و بصورت پیوسته، پارامترهای مدل شبیه سازی که مشاهده می کنند بر می گردانند یعنی در واقع چند پاسخ بصورت موازی (MRIP: Multiple Replication in Parallel) برگردانده میشود؛ و این همان چیزی است که پردازش کنترلی مرکزی (Central Controlling Process) نیاز دارد. در حال حاضر پروژه هایی مثل پروژه Akaroa2 (که شبیه سازی موازی را با NS2 یا OMNET++ انجام میدهد،) وجود دارند که از این قابلیت شبیه سازها استفاده میکنند؛ با اینکار، افزایش سرعت و قابلیت اطمینان را برای شبیه سازیها به ارمغان می آورند.

- گروه های کاربران متعددی از آن استفاده میکنند
- Open source بوده و رایگان است

مهمترین معایب NS2 عبارتند از:

- مدت زمان طولانی برای آشنایی با آن
- سورس کد و دستور العملهای آن بد مستند سازی شده اند؛ مستندات آن برای افراد تازه کار مناسب نیست و تنها برای کسانی که با این شبیه ساز کار کرده اند کارآ است
- دشواری در ارزیابی سریع یک ایده کوچک (شما باید تمام ساختار های شبیه سازی را بدانید حتی اگر بخواهید قسمتی از پشته های پروتکل را شبیه سازی کنید.)

برای شروع کار با شبیه ساز NS2 لازم است تا آشنایی مختصری درباره زبان TCL حاصل شود. از این رو در ادامه به مقدمات زبان TCL پرداخته میشود.

زبان TCL

الف– آشنایی با TCL/TK

TCL [۱]، (با تلفظ تیکل Tickle)، به عنوان یک زبان چسبی (که مثل چسب به دیگر برنامه ها می چسبد!) ، کاربران را قادر می سازد تا برنامه ها و برنامه های کمکی[۲] را کنترل کنند. TCL و مکمل گرافیکی آن یعنی TK [۳] (با تلفظ تی کِی tee kay) در سال ۱۹۸۰ توسط دکتر "جان آستِرهاوت"[۴] در دانشگاه برکلی کالیفرنیا ایجاد شدند. ایجاد برنامه TCL توسط دکتر آسترهاوت، دو هدف عمده داشت:

۱- استفاده به عنوان زبان اسکریپتی[۵] ، که برنامه ها را قادر می سازد تا با احضار دستورات TCLبا یکدیگر ارتباط داشته باشند.
۲- استفاده به عنوان مفسر (مترجم) قابل جاسازی[۶] در برنامه های دیگر، که کاربران را قادر می سازد تا به کمک زبان اسکریپتی TCL، برنامه ها(یی که به زبانهای دیگر نوشته شده اند،) را به دلخواه خود تغییر دهند. چرا که کار با برنامه TCL، به مراتب ساده تر از کار با برنامه های دیگر است.

TCL، تقریبا مشابه "ویژوال بیسیک برای برنامه های کاربردی[۷] " (VBA) است. همانطور که به کمک VBA میتوان از کارکرد برنامه های World، Excel و PowerPoint در یک برنامه کاربردی دیگر استفاده کرد، به وسیله TCL (و TK) نیز میتوان از مجموعه متنوعی از برنامه ها در یک برنامه دیگر بهره برد.

۱- Tool Command Language: زبان دستور– ابزار
۲- Utility
۳- (graphical) Toolkit
۴- Dr. John Ousterhout
۵- scripting language: یک زبان برنامه نویسی ساده که برای انجام کارهای خاص یا محدود طراحی میشود و اغلب با یک برنامه کاربردی خاص مرتبط است.
۶- embeddable interpreter
۷- Visual Basic for Applications (VBA): زبان ماکرونویسی که نگارشی از ویژوال بیسیک بوده و محصول شرکت مایکروسافت است.

همانطور که گفته شد، هدف اصلی TCL به عنوان یک زبان سطح بالا، فراهم کردن امکان تعامل میان برنامه های مختلف بود. اما از مدتها پیش، مثل هر برنامه یا زبان برنامه نویسی موفق دیگر، TCL/TK نیز از مرز هدف اصلی خود تجاوز کرده و گاها برنامه هایی به کمک آن نوشته میشوند که صدها و هزاران خط کد را در بر میگیرند ؛ قابلیتهای زبان هسته TCL تا جایی رشد کرده که کاربران این زبان قادرند برنامه های شبکه ای توانمندی را خلق کنند که میتوانند با پایگاه داده ها تعامل داشته باشند، وب را جستجو و در آن گشت و گذار کنند، به صفحات وب خدمت رسانی کنند و ابزار MIDI [۱]را کنترل کنند.

قابل ذکر است که TCL یک برنامه چند–محیطی[۲] است. به این معنی که کدهای TCL که مثلا در محیط لینوکس نوشته می شوند در هر محیط دیگری (مثلا ویندوز)، که مفسر (مترجم) TCL در آن وجود دارد، بدون هیچ گونه تغییری اجرا میشود. بنابراین جاوا اولین زبانی نیست که ادعا میکند "یک بار بنویس، همه جا اجرا کن![۳] ". قابلیت چند محیطی بودن TCL، باعث می شود که، به عنوان مثال، کاربران مجبور به یادگیری نحوه تعامل با فایلهای لینوکس یا پشته پروتکلی TCP/IP نباشند.

TK در ساده ترین تعریف، امتداد زبان TCL (یا به عبارت دقیقتر، یک کتابخانه TCL) است و در واقع یک ابزار کمکی بمنظور خلق و استفاده از واسطهای گرافیکی کاربر [۴]می باشد. TK شامل دستوراتی برای ایجاد دکمه ها، جعبه های متنی [۵](و دیگر واسطهای گرافیکی) و همچنین کنترل رنگ و فونت است. با توجه به اینکه استفاده از این ابزار کمکی برای کار با شبیه ساز NS2 ضروری نیست، بیش از این به تشریح این ابزار پرداخته نمیشود. بنابراین در قسمت بعدی مستقیما به کار با برنامه TCL و دستورات مقدماتی آن تمرکز میشود.

با توجه به توضیحات گفته شده، ویژگیهای TCL از این قرارند:

- امکان توسعه سریع را فراهم میکند
- از واسط گرافیکی بهره میبرد
- با بسیاری از محیطها سازگار است
- انعطاف پذیر است؛ بنحوی که میتوان آن را به سادگی در برنامه های دیگر استفاده کرد
- استفاده از آن ساده است
- رایگان است

ب– دستورات مقدماتی TCL

۱- **درج توضیحات:** برای درج توضیحات[۶] در زبان TCL از کاراکتر # (number sign یا pound sign یا sharp) استفاده می شود. بنابراین عباراتی که در جلوی این کاراکتر نوشته میشوند توسط مترجم TCL تفسیر نمیشوند.

```
#so we are going to start programming with TCL
```

۲- **اعلان متغیر:** جهت تعریف متغیرها در این زبان از کلمه کلیدی set استفاده می شود. به عنوان مثال اگر بخواهیم مقدار اولیه ۱۲ را در متغیر a ذخیره کنیم از دستور

```
set a 12
```

استفاده میکنیم. بنابر این این عبارت معادل عبارت a=12 (در زبان C) است.

۱- Musical Instrument Digital Interface: رابط دیجیتالی ادوات موسیقی، فایل های MIDI جهت برقراری ویدئو کنفرانس ها و پخش فیلم در اینترنت به کار می روند

۲- cross-platform: اصطلاحی برای یک نرم افزار کاربردی (یا سخت افزار) که در بیش از یک محیط قابل اجرا (استفاده) است.

۳- Write once, run anywhere

۴- graphical user interface

۵- text box

۶- comment

تذکر: TCL یک زبان رشته–محور [1] است؛ به این معنی که تمام متغیرها در این زبان از نوع رشته (string) هستند. حتی اعداد! در قسمتهای بعدی متوجه خواهیم شد که چگونه مترجم TCL عملیات ریاضی روی اعداد را (به عنوان عبارت ریاضی و نه به عنوان رشته،) تشخیص خواهد داد.

مثال: اگر بخواهیم رشته hello را در متغیر b ذخیره کنیم مینویسیم:

```
set b hello
```

در صورتی که رشته مفروض *بیش از یک کلمه* باشد باید از علامت کوتیشن مضاعف [2] یعنی " در ابتدا و انتهای رشته استفاده کنیم؛

مثلا اگر بخواهیم رشته hello world را در متغیر c ذخیره کنیم مینویسیم:

```
set c "hello world"
```

تذکر ۱: در مورد رشته های تک کلمه ای هم میتوان از کوتیشن مضاعف استفاده کرد:

```
set b "hello"
```

تذکر ۲: اگر یک رشته طولانی داشته باشیم که میان کلمات آن فاصله (space) وجود نداشته باشد، الزامی در استفاده از کوتیشن مضاعف نیست:

```
set c helloWorld
```

تذکر ۳: برای حذف یک متغیر از دستور unset استفاده می شود:

```
unset c
```

در این حالت، و پس از اجرای این فرمان، متغیر c حذف می شود و دیگر متغیری به نام c وجود نخواهد داشت.

۳- **نمایش مقدار متغیرها**: در مثالهای قبل دیدیم که چگونه مقدار ۱۲ در متغیر a ذخیره میشود؛ حال میخواهیم ببینیم چگونه میتوان به مقدار ذخیره شده در متغیر a دسترسی یافت. برای دسترسی به مقدار یک متغیر از علامت دلار یعنی $ استفاده میشود. مثلا اگر بخواهیم **مقدار** متغیر a را در متغیر d ذخیره کنیم، مینویسیم:

```
set d $a
```

یعنی **مقدار** a را در d قرار بده! این عبارت معادل عبارت d=a (در زبان C) است. در این حالت مقدار متغیر d برابر ۱۲ میشود.

به عبارت کلی هر جا بخواهیم از **مقدار** یک متغیر استفاده کنیم (چه در عبارات ریاضی، چه در مقدار دهی متغیرهای دیگر، چه در هنگام چاپ خروجی و ...)، از علامت $ استفاده میکنیم.

۴- **چاپ نتایج در خروجی**: برای چاپ نتایج در خروجی از کلمه کلیدی puts استفاده میشود. مثلا اگر بخواهیم مقدار متغیر a را در خروجی چاپ کنیم، می نویسیم:

```
puts $a
```

در این حالت مقدار ۱۲ در خروجی چاپ می شود. و اگر داشته باشیم:

```
puts "a is $a"
```

در این حالت عبارت a is 12 در خروجی چاپ میشود.

تذکر: به جای علامت " میتوان از کروشه برای مجموعه داده ها استفاده کرد:

```
set c {hello world}
```

که معادل عبارت set c "hello world" است.

تفاوت استفاده از کروشه و استفاده از کوتیشن مضاعف با مثال زیر مشخص می شود:

۱- string-based

۲- double quotation

حالت ۱ : puts “c is $c”

خروجی : c is hello world

حالت ۲ : puts {c is $c}

خروجی : c is $c

همانطور که مشاهده می شود، در حالت اول که از علامت “ برای مجموعه ای از رشته ها استفاده کرده ایم، عبارت $c به عنوان **مقدار متغیر** c قلمداد می شود در حالی که در حالت دوم که از علامت } برای مجموعه ای از رشته ها استفاده کرده ایم، عبارت $c صرفا به عنوان یک رشته تفسیر شده است.

۵- **عبارات ریاضی:** همانطور که پیشتر گفته شد، تمام متغیرها در زبان TCL از نوع رشته هستند. اما چگونه میتواند روی اعداد (که آنها هم به عنوان رشته در نظر گرفته میشوند،) عملیات ریاضی انجام داد؟ این کار با دستور expr که از واژه "expression" به معنی "عبارت" (و در اینجا بمعنی عبارت ریاضی) گرفته شده، انجام می شود. به عنوان مثال برای چاپ خروجی جمع دو عدد ۵ و ۱۰ به این ترتیب عمل میکنیم:

```
puts [expr 5+10]
```

خروجی: ۱۵

دقت کنید که عبارت ریاضی به همراه کلمه کلیدی expr حتما باید درون علامت براکت [] نوشته شوند.

مثال: برنامه ای که عدد ۳ را در متغیر a و عدد ۷ را در متغیر b قرار داده، این دو متغیر را در هم ضرب کرده و حاصل را در متغیر c قرار داده و نهایتا مقدار c را در خروجی چاپ میکند:

```
#define a=3 and b=7
   set a 3
   set b 7
#now define c=a*b
   set c [expr {$a*$b}]
    puts “c is $c”
```

خروجی: c is 21

در کد فوق میتوان به جای عبارت expr{$a*$b} از expr($a*$b) یا expr $a*$b استفاده کرد اما بهتر است بمنظور یکدستی برنامه (چنانچه در ادامه نیز خواهید دید)، مجموعه داده ها با { و } مشخص شوند.

تذکر: در زبان TCL نوع اعداد (اعشاری یا صحیح) با توجه به مقادیر متغیرها تعریف میشوند. به مثال زیر توجه کنید:

```
set a [expr 1/5]
```

در این حالت مقدار متغیر a برابر 0 (صفر) میشود. اگر بجای عبارت فوق بنویسیم:

```
set a [expr 1.0/5.0]
```

در این حالت مقدار متغیر a برابر 0.2 میشود.

۶- **دریافت مقادیر از ورودی:** برای دریافت یک مقدار از ورودی از دستور gets stdin استفاده میشود. به عنوان مثال پس از اجرای کد زیر:

```
gets stdin c
```

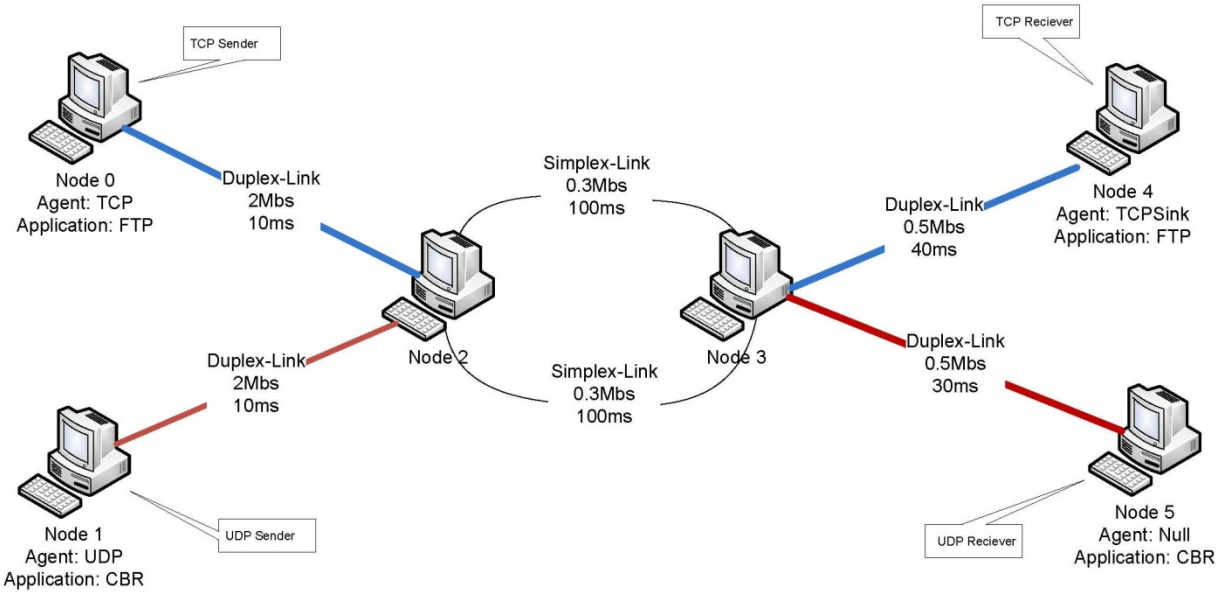
TCL منتظر میماند تا مقداری وارد شود و کلید enter فشرده شود. در این حالت مقدار وارد شده در متغیر c ذخیره میشود.
مثال: برنامه ای که با چاپ پیغام مناسب، یک اسم را از کاربر دریافت کرده و اسم وارد شده را به همراه تعداد کاراکترهای آن چاپ میکند:

```
puts -nonewline "Please enter name: "
flush stdout
set count [gets stdin playerName]
puts "Player name is $playerName."
puts "It has $count characters."
```

تذکر ۱: عبارت nonewline– جهت گرفتن نام کاربر در همان سطر نمایش پیغام "Please enter name: " استفاده شده است.
تذکر ۲: دستور flush stdout برای خالی کردن قسمتی از حافظه (بافر [۱]) استفاده شده است. هنگامی که در یک سطر هم پیغام چاپ میکنیم و هم ورودی میگیریم باید از این دستور استفاده کنیم؛ تا فضای حافظه مورد نیاز در اختیارمان قرار بگیرد.
تذکر ۳: پس از وارد کردن هر رشته بصورت ورودی، تعداد کاراکترهای آن رشته نیز بصورت خودکار وارد چاپ میشود؛ بنابراین نیازی به نوشتن دستور مجزا برای محاسبه تعداد کاراکترهای یک رشته نمیباشد.

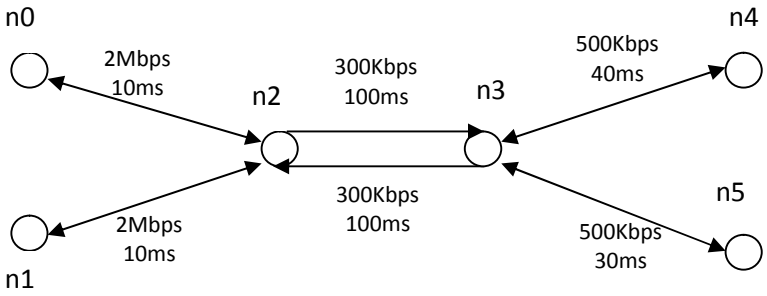
۱- buffer

بررسی سناریو



توپولوژی سناریو

در این توپولوژی، شش گره در نظر میگیریم. گره های n0 و n2 با یک لینک دوطرفه (که مشخصات آن عبارت است از پهنای باند ۲ مگابایت در ثانیه و تاخیر انتشار ۱۰ میلی ثانیه) به هم متصل شده اند. گره های n1 و n2 نیز با لینکی مشابه لینک n0n1 مرتبط هستند. میان گره های n2 و n3 دو لینک یکطرفه مشابه (پهنای باند ۳۰۰ کیلوبایت در ثانیه و تاخیر انتشار ۱۰۰ میلی ثانیه)، میان گره های n3 و n4 یک لینک دوطرفه (پهنای باند ۵۰۰ کیلوبایت در ثانیه و تاخیر انتشار ۴۰ میلی ثانیه) و نهایتا میان گره های n3 و n5 لینکی دوطرفه (با پهنای باند ۵۰۰ کیلوبایت در ثانیه و تاخیر ۳۰میلی ثانیه) وجود دارد. این توپولوژی را در شکل زیر ملاحظه مینمایید:



در این سناریو میخواهیم از n0 به n4 از طریق پروتکل TCP و کاربرد FTP و همچنین از n1 به n5 از طریق پروتکل UDP و کاربرد CBR بسته ارسال کنیم.

در ادامه به نحوه پیاده سازی این توپولوژی و آشنایی با مفاهیم اولیه پرداخته میشود.

شروع و پایان شبیه سازی

الف) شروع شبیه سازی

شبیه سازی ns با دستور زیر شروع می شود:

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

این دستور، اولین خط برنامه در TCL است. با این دستور یک متغیر مثل ns را به کمک دستور set تعریف کرده ایم. این متغیر، در واقع یک شی جدید (new) از کلاس شبیه ساز (Simulator) است. از کلمه کلیدی new برای ایجاد یک شی جدید از هر کلاسی استفاده می شود. با تعریف یک شیء (ns) از کلاس Simulator، قادر خواهیم بود از تمامی دستورات و تابع و خواص (از پیش تعریف شده) کلاس Simulator استفاده کنیم.

تذکر۱: همانطورکه گفته شد، ns یک متغیر و یک شی از کلاس Simulator است؛ نام این متغیر می تواند هر چیز دیگری (مطابق میل شما) باشد. نامی که برای آن انتخاب می کنیم، در تمام طول برنامه، معرف کلاس شبیه ساز است.

تذکر۲: به کوچک و بزرگ بودن حروف دقت کنید. مثلا در کلاس Simulator ، حرف S بزرگ بوده و بقیه حروف کوچک هستند. معمولا حرف شروع نام کلاسها به صورت بزرگ تعریف شده اند که در ادامه بحث با آنها بیشتر آشنا خواهید شد. (مثل Application,Agent و ...)

تا اینجا یک شی از کلاس شبیه ساز (Simulator) ایجاد کرده ایم که تمامی خواص و کارکردهای این کلاس را به ارث می برد.

ب) ایجاد فایل برای ذخیره سازی رخدادها

در قسمت ایجاد فایل ها، با نحوه باز کردن و بستن فایل ها آشنا شدیم. در برنامه های شبیه سازی هم برای نگه داری اطلاعات خروجی حاصل از اجرای شبیه سازی ، باید فایلی ایجاد کنیم.

فایلی که حاوی اطلاعات خروجی شبیه سازی است، فایل ردیابی (Trace) نام دارد. فایلهای ردیابی، یا به صورت متنی هستند و یا به صورت گرافیکی بصری (که با برنامه Nam ظاهر می شوند).

قطعه کد زیر را در نظر بگیرید:

```
1.  #open the Trace File
2.  set tracefile1 [open out.tr w]
3.  $ns trace-all $tracefile1
```

خط اول، کد مربوط به توضیح برنامه است (به علامت # توجه کنید)

در خط دوم، می خواهیم یک فایل Trace قابل نوشتن (w:writable) ایجاد کنیم که نام out.tr را برای آن انتخاب کرده ایم.

tracefile1 در واقع مشخصه[۱] یا اشاره گر فایل out.tr است که در طول برنامه از آن به جای صدا زدن فایل اصلی (یعنی out.tr) استفاده می کنیم.

در خط سوم از دستور trace-all استفاده کرده ایم. این دستور، از دستورات تعریف شده شبیه ساز است؛ بنابراین آن را با $ns معرفی کرده ایم تا مترجم OTCL متوجه شود که این دستور مربوط به ردیابی رخدادهای شبیه سازی است. در جلوی دستورtrace-all ، مقدار فایل Tracefile1 (یعنی $tracefile1) را نوشته ایم که در واقع همان فایل out.tr است. در مجموع این خط بیان می کند که تمام

۱- ID: Identifier

رخدادهایی که در شبیه سازی ($ns) اتفاق می افتند، ردیابی کن (trace-all) و آن را در فایلی بنویس که مشخصه آن tracefile1 است (یعنی out.tr).

بطور مشابه، اگر بخواهیم فایل ردیابی را در برنامه گرافیکی یعنی nam اجرا کنیم، به ترتیب زیر عمل می کنیم:

```
1. # open the Nam  Trace File
2. Set namfile [open out.nam w]
3. $ns namtrace-all $namFile
```

خط اول مربوط به توضیحات کد است.

در خط دوم یک فایل out.nam قابل نوشتن (w) ایجاد کرده ایم و جهت صدا زدن این فایل در طول برنامه، شناسه namfile را به آن اختصاص داده ایم.

در خط سوم، با استفاده از دستور namtrace-all، همه رخدادها (all-) را در فایل namfile (که مشخصه فایل out.nam است،) ذخیره می کنیم تا بتوانیم به کمک nam و بصورت گرافیکی، این رخدادهای اتفاق افتاده در طول شبیه سازی را در فایل مورد نظر ثبت کنیم. اما در برخی موارد حجم رخدادها در یک سناریوی شبیه سازی به قدری زیاد می شود که عملا با هزاران رخداد در فایل Trace مواجه می شویم. در حالی که ممکن است تنها رفتار قسمتی از سیستم، مطلوب ما باشد. در قسمت های بعدی خواهیم دید که چگونه می توان فایل ردیابی (Trace) ای ایجاد کرد که تنها روی قسمتی از سیستم متمرکز باشد.

ج) پایان شبیه سازی

رویه[1] پایان برنامه: جهت اتمام برنامه، تمامی فایل های Trace که ایجاد کرده ایم باید ببندیم. به رویه زیر توجه کنید:

```
1.  # define a 'finish' procedure
2.  Proc finish{}{
3.  global ns trancefile1 namfile
4.  $ns flush-trace
5.  close $trancefile1
6.  close $namfile
7.  exec nam out.nam &
8.  exit 0
9.  }
```

در قسمت های قبلی با نحوه تعریف رویه ها در Tcl آشنا شدیم. رویه finish فوق، چنان که مشاهده می شود، آرگومان ورودی ندارد.

کلمه کلیدی global بیان می دارد که متغیرهایی که از آنها استفاده می کنیم (یعنی ns، tracefile1 و namfile) متغیرهایی هستند که در خارج از رویه، آنها را تعریف کرده ایم و در واقع متغیرهای عمومی هستند.

دستور flush-trace که از دستورات شبیه ساز ns است (و قبلا در کلاس Simulator تعریف شده است) trace های برنامه را در فایل های مطلوب (یعنی out.tr و out.nam که با اشاره گرهای tracefile و namfile مشخص شده اند،) نسخه برداری و ثبت می کند. در واقع محلی از حافظه را جهت دریافت trace ها و کپی آنها در فایل های مقصد، رزرو می کند.

۱- procedure

دستور close برای بستن فایل هایی که قبلا باز کرده ایم (و اطلاعات را روی آنها نوشته ایم) به کار می رود. بنابراین با این دستور، فایل ها از حالت قابل نوشتن خارج شده و ذخیره شده و در نهایت بسته می شوند.

دستور exec جهت اجرا کردن برنامه ها به کار می رود. از آنجایی که این دستور روی فایل های واقعی (و نه اشاره گرهای آنها) عمل می کند، از این رو اسم فایل اصلی و حقیقی یعنی out.nam نوشته شده است.

دستور nam هم برای اجرا کردن برنامه nam به کار می رود. قابل ذکر است که حتی در محیط xwin اگر دستور nam out.nam را بنویسید باز هم می توانید فایل Trace برنامه را به صورت گرافیکی ببینید.

تذکر: در دستور nam، نام فایل اصلی باید نوشته شود و نه فایل اشاره‌گر آن.

در نهایت دستور exit 0 باعث پایان یافتن برنامه می‌شود. عدد 0 در جلوی exit، بیانگر خروج موفق است یعنی اگر برنامه بصورت صحیح اجرا شد، پایان یابد. این مقدار، مقدار پیش فرض دستور exit است. مقادیر دیگری به جای 0 می‌تواند وجود داشته باشد که بیانگر این مطلب هستند که برنامه خاتمه یافت اما بطور صحیح اجرا نشد (خروج ناموفق)!

بعد از پایان هر رویه‌ای، باید آن را در مکان مناسب صدا بزنیم تا قابلیت اجرایی پیدا کند؛ رویهٔ finish هم از این قاعده مستثنی نیست. بنابراین باید تعیین کنیم که چه وقت برنامه پایان یابد. برای این کار یک زمان (بر حسب ثانیه) مشخص می‌کنیم و اعلام می کنیم که شبیه‌ساز پس از این مدت زمان، پایان یابد (و رویهٔ finish صدا زده شود)؛ مثلاً دستور زیر را در نظر بگیرید:

```
$ns at 100 "finish"
```

این دستور گویای این مطلب است که باید پس از ۱۰۰ ثانیه، شبیه سازی پایان یابد.

حال باید شبیه‌سازی را اجرایی کنیم. برای این منظور از دستور run، به این ترتیب استفاده می‌کنیم:

```
$ns run
```

گره

گره[۱]، به هر وسیله‌ای اطلاق می‌شود که قابلیت اتصال به شبکه را داشته باشد. با این تعریف، کامپیوترها (PC، Laptop و ...)، موبایل‌ها، پرینترها، روترها و ... همگی تحت عنوان گره در برنامه تعریف می‌شوند.

در OTCl یا به طور دقیق‌تر در NS2، گره ها به این صورت تعریف می‌شود؛

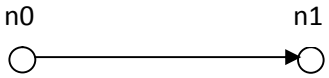
```
set node0 [$ns node]
```

با این دستور یک متغیر با اسم (دلخواه) node0 تعریف کرده‌ایم که تمام ویژگی‌های یک node (با حروف کوچک) در شبیه ساز ns را به ارث می‌برند. معمولاً در شبکه‌هایی که شامل تعداد زیادی گره هستند از متغیرهای n0 , n1 و ... استفاده می‌شود. دقت کنید در هر جای برنامه که نیاز به استفاده از یک گره قبلاً تعریف شده، مثل node0 داریم، باید از کاراکتر $ هم استفاده کنیم. در ادامه بیشتر با این مطلب آشنا خواهید شد.

۱- Node

لینک

هنگامی که چند گره تعریف می‌کنیم، نیاز به تعریف شیء ارتباطی آنها نیز خواهیم داشت. لینک‌ها[1] در واقع اشیاء ارتباطی میان چند گره هستند. مثلاً در شکل زیر میان دو گره n1 و n0 یک لینک وجود دارد.



شکل ۳-۱ لینک یکطرفه میان گره n0 و گره n1

این لینک به این صورت تعریف می‌شود.

```
$ns simplex–link $n0 $n1 2Mb 5ms DropTail
```

با توجه به این که لینک‌ها اشیائی از شبیه ساز (ns) هستند، از این رو از عبارت $ns استفاده شده است.

simplex–link بیانگر یک لینک یک طرفه[2] است. در این نوع از لینک‌ها ارتباط بین دو گره، بصورت یک طرفه است؛ یعنی یک گره همواره فرستنده است (n0) و دیگری همواره گیرنده (n1)؛ در مقابل این نوع لینک، لینک‌های دو طرفه[3] وجود دارند یعنی duplex–link که هر دو گره می‌توانند فرستنده یا گیرنده باشند.

اولین گره که در تعریف link نوشته می‌شود، بیانگر گره ارسال کننده (در اینجا n0) و دومین گره، دریافت کننده (در اینجا n1) است.

پهنای باند[4] یا ظرفیت لینکی که تعریف کرده‌ایم ۲ مگابایت (2Mb) در ثانیه است.

زمان تاخیر انتشار[5]، ۵ میلی‌ثانیه است.

نظام صف بندی[6] از نوع DropTail (یا FIFO[7]) است.

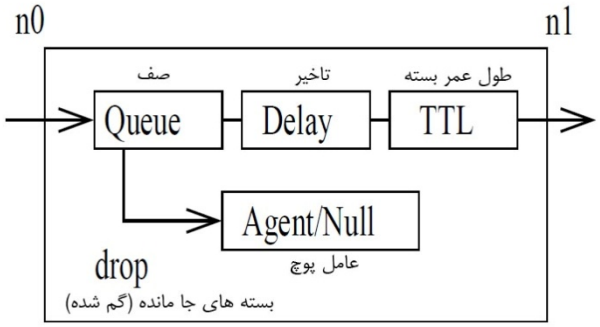
تذکر: راجع به این مفاهیم مفصلاً بحث خواهد شد.

با توجه به تعریفهای فوق، نحوه پیاده سازی لینک را بصورت زیر میتوان تشریح نمود:

$ns	simplex-link	$n0	$n1	2Mb	5ms	DropTail
	نوع لینک (یکطرفه/دوطرفه)	گره منبع	گره مقصد	پهنای باند	تاخیر انتشار	نظام صف

همانطور که ملاحظه می‌کنید، در ns، صف‌ها به عنوان قسمتی از لینک‌ها پیاده‌سازی شده‌اند؛ بنابراین با تعریف لینک‌ها و نظام‌های صف در آنها، نحوه رویارویی با بسته‌های اطلاعاتی سرریز (Over flow) نیز تعریف می‌شوند. در اینجا صفی با نظام DropTail تعریف شده است؛ ساختار این صف به گونه‌ای است که اگر ظرفیت صف (با بسته‌های ارسال شده) تکمیل شود، در این صورت آخرین بسته‌ای که در صف جایی ندارد، گم (drop) می‌شود. شکل زیر، ساختار یک لینک یک طرفه را نشان می‌دهد.

۱- Link
۲- unidirectional
۳- bidirectional
۴- band width
۵- propagation delay
۶- Queuing Discipline
۷- First In First Out



شکل ۳-۱ : ساختار لینک یکطرفه

مطابق شکل، بسته‌های سرریزی که جا می‌مانند، به یک عامل پوچ (Agent/Null) ارسال می‌شوند.

هر بسته، طول عمری دارد (TTL [۱])؛ این طول عمر در واقع مدت زمان مجاز هر بسته تا گم شدن است؛ یعنی پس از انقضای این زمان، بسته گم می‌شود (جا می ماند).

تذکر: لینک‌های دو طرفه از دو لینک یک طرفه موازی و غیر هم جهت تشکیل شده است. بنابراین ساختار آن شامل دو ساختار مشابه شکل فوق و با جهت‌های مخالف است.

هر صف، ظرفیت مشخصی دارد؛ این ظرفیت به صورت پیش فرض در ns، مقدار ۵۰ دارد. با این حال این ظرفیت با دستور زیر قابل تغییر است:

```
$ns queue-limit $n0 $n1 20
```

با این دستور، ظرفیت صف لینکی که n1 و n0 را به هم وصل می‌کند، برابر ۲۰ تعریف می‌شود؛ queue-limit، متغیر از پیش تعریف شده برای ظرفیت صف‌ها در ns است و سه آرگومان دارد (دو آرگومان مربوط به گره های تشکیل دهنده لینک و یک آرگومان مربوط به ظرفیت صف). این مقدار را می‌توانید در فایل ns-defult.tcl که در آدرس ns-allinone-2.xx/ns-2.xx/tcl/lib قرار دارد، و با دستور Queue set limit_ 50 مشخص شده است، مشاهده کنید.

تذکر: برخی از نظام‌های صف موجود در ns عبارتند از: SFQ (Stochastic Fair Queuing)، CBQ (priority and a round-robin schedular)، DRR (Deficit Round-Robin)، FQ (Fair Queuing) و RED (Random Early Discard) .

عامل‌ها و کاربردها

تاکنون توپولوژی ای ایجاد شده که شامل اجزاء فیزیکی شبکه (گره ها و لینک‌ها) میباشد، اکنون باید جریان ترافیکی [۲] میان آنها را ایجاد کنیم تا توپولوژی کامل شود. برای این منظور به تعریف مسیریابی (در منبع‌ها و مقصدهای مشخص)، عامل‌ها [۳] (پروتکل‌ها) و کاربردها [۴] هایی که استفاده می‌کنند، نیازمندیم.

در این سناریو مایلیم که یک کاربرد FTP [۵] بین گره های n0 و n4 و یک کاربرد CBR [۶] ، بین گره های n1 و n5 ایجاد کنیم. کاربرد FTP

۱- Time To Live
۲- Traffic flow
۳- Agent
۴- Application
۵- File Transfer Protocol
۶- Constant Bit Rate

مربوط به پروتکل TCP/IP [۱] و کاربرد CBR [۲] مربوط به پروتکل UDP [۳] هستند.

پروتکلTCP و کاربرد آن FTP

TCP، یک پروتکل دینامیکی مطمئنِ کنترلِ ازدحام [۴] است.TCP، از بسته های تصدیقی[۵] ای بهره میبرد که توسط مقصد تولید میشوند و بیان میدارند که آیا بسته ها به درستی دریافت شده اند یا خیر. بسته های گمشده به عنوان سیگنال های ازدحام [۶] تفسیر میگردند. در TCP به لینکهای دو طرفه نیاز داریم تا (علاوه بر ارسال بسته های داده توسط منبع،) بسته های تصدیق نیز بتوانند از مقصد (به منبع) ارسال شوند.
تذکر: انواع مختلفی از پروتکل TCP وجود دارند. مثل Tahoe (که به عنوان پیش فرض در ns استفاده میشود)، Reno،NewReno، Vegas و SACK.
برای شروع، باید یک شیء از کلاس TCP که در کلاس ns تعریف شده ایجاد کنیم:

```
set tcp [new Agent/TCP]
```

نام این شی را tcp (با حروف کوچک) در نظر گرفته ایم که حاوی تمام خصوصیات و عملکردهای کلاسTCP (با حروف بزرگ) به عنوان مبدا (یا ارسال کننده) است. اگر بخواهیم از انواع دیگر عاملهای فرستنده مثل Reno استفاده کنیم، مینویسیم:

```
set tcp [new Agent/TCP/Reno]
```

حال این پروتکل ارسالی را باید به گره ارسال کننده یعنی n0 ضمیمه کنیم:

```
$ns attach-agent $n0 $tcp
```

دقت کنید نام عامل (agent)های فرستنده و گیرنده در پروتکلهای TCP و UDP متفاوت هستند. در بالا n0 را به عنوان عامل فرستنده (Agent/TCP) تعریف کرده ایم.
اگر بخواهیم یک عامل گیرنده تعریف کنیم مینویسیم:

```
set sink [new Agent/TCPSink]
```

کلمه کلیدی TCPSink را برای عامل گیرنده پروتکل TCP تعریف کرده ایم. sink در واقع شیء ای از این عامل گیرنده است که در طول برنامه به جای صدا زدن مستقیم Agent/TCPSink از آن استفاده میکنیم. این عامل گیرنده را به گره گیرنده (یعنی n4) ضمیمه میکنیم:

```
$ns attach-agent $n4 $sink
```

پس از تعریف عاملهای گیرنده و فرستنده باید آنها را به هم وصل کنیم یعنی اتصال TCP را میان گره منبع و مقصد برقرار میکنیم:

```
$ns connect $tcp $sink
```

در TCP، پارامترهای زیادی وجود دارند که مقدار اولیه آنها به طور پیش فرض تعریف شده اند به عنوان مثال اندازه بسته های اطلاعاتی به طور پیش فرض ۱۰۰۰ بایت در نظر گرفته شده است (در UDP هم به همین صورت است)؛ اگر بخواهیم این مقدار را تغییر دهیم و مثلا آن را

۱- TCP(transfer control protocol) / IP (internet protocol)
۲- Constant Bit Rate
۳- User Datagram Protocol
۴- Congestion Control
۵- Acknowledgement
۶- congestion signals

۵۵۲ بایت کنیم از دستور زیر استفاده میکنیم:

```
$tcp set packetSize_ 552
```

متغیر _packetSize یک متغیر از پیش تعریف شده است.

وقتی در یک برنامه شبیه سازی، چند جریان (مثلا در اینجا هم جریانTCP و هم جریان UDP) داریم، برای متمایز کردن این جریانها، یک شناسه جریان (Flow ID= fid) برای هر یک از جریانها اختصاص میدهیم. این کار سبب می شود که مثلا با تعیین رنگ برای هر یک از جریانها، آنها را در برنامه گرافیکی nam متمایز کنیم. دستور زیر نحوه اختصاص یک شناسه جریان برای عامل فرستنده tcp را نمایش میدهد:

```
$tcp  set  fid_1
```

برای آبی کردن جریانهای TCP از دستور

```
$ns color 1 Blue
```

استفاده میشود؛ در این حالت جریانهایی با شناسه ۱، (یعنی جریانهای TCP) با رنگ آبی نشان داده میشوند. در ادامه خواهیم دید که با دستورهای مشابه $udp0 set fid_2 و $ns color 2 red، رنگ جریانهای UDP قرمز خواهند شد.

تذکر: انواع مختلفی از رنگها را میتوان استفاده نمود از جمله: Cyan، Magenta، Yellow، Black، Red، Green، Blue و...

با تعریف TCPحال نوبت به تعریف کاربرد آن یعنیFTP می رسد:

```
1.  #setup  a  FTP over TCP  connection
2.  set ftp [new Application/FTP]
3.  $ftp  attach-agent  $tcp
```

درخط دوم یک کاربرد ftp (با حروف کوچک) تعریف کرده ایم که تمام خصوصیات مربوط به کلاس FTP (با حروف بزرگ) را به ارث میبرد و در خط سوم این کاربرد را به تولید کننده این ترافیک، یعنی tcp ضمیمه کرده ایم.

پروتکل UDP وکاربرد آن CBR

نحوه تعریف پروتکل UDP و کاربرد آن (CBR) مشابه تعریف TCP و کاربرد آن (FTP) میباشد.

تعریف اتصال UDP:

```
1.  #setup  a  UDP  connection
2.  set  udp [new  Agent/UDP]
3.  $ns  attach-agent  $n1  $udp
4.  set  null  [new  Agent/Null]
5.  $ns  attach-agent  $n5 $null
6.  $ns connect  $udp  $null
7.  $udp  set  fid_2
```

همانطور که دیده میشود عامل فرستنده پروتکل UDP را با Agent/UDP مشخص می کنیم و یک شی udp که حاوی تمام خصوصیات و کارکردهای UDP فرستنده است (خط ۲) تعریف کرده ایم.

عامل گیرنده در پروتکل UDP، Agent/Null است. متغیری با نام null که تمام خصوصیات و کارکردهای عامل گیرنده را (در UDP) به ارث میبرد، تعریف کرده ایم.

خاصیت فرستندگی را روی گره n1 (خط ۳) و خاصیت دریافت کنندگی را روی گره n5 (خط ۵) ضمیمه کرده ایم و در نهایت این دو عامل را

(در خط ۶) به هم متصل کرده ایم. خط ۷ هم بیانگر شناسه این جریان[۱] (جریان udp) است که برای مشخص کردن رنگ آن، باید دستور $ns color 2 Red را وارد کنیم. در این حالت پس از اجرای برنامه nam، جریان های UDP با رنگ قرمز مشخص می شوند.

نحوه تعریف کاربرد CBR:

```
1. #setup  a  CBR over UDP connection
2. set cbr  [new Application/Traffic/CBR]
3. $cbr  attach-agent  $udp
4. $cbr  set  packetSize_1000
5. $cbr  set  rate_0.01 Mb
6. $cbr  set  random_false
```

کاربرد CBR همانطور که از اسم آن بر می آید (ارسال با نرخ ثابت: constant bit rate), بیت ها را با نرخ ثابت ارسال می کند, از این رو باید مقدار این ثابت را تعریف کنیم (در خط ۵). در اینجا این مقدار ثابت را 0.01 مگابایت در نظر گرفته ایم.
در خط ۶ از دستور _random استفاده شده و مقدار آن false است، این دستور بیان میدارد که آیا noise به صورت تصادفی در زمانهای انتقال، ایجاد شود یا خیر؟ به طور پیش فرض این مقدار برابر false است، اما در صورتی که بخواهیم در این جریان noise ایجاد کنیم باید از random_1 استفاده کنیم.
اندازه بسته (خط ۴) هم به طور پیش فرض مقدار ۱۰۰۰ بایت دارد که می توان آن را نیز تغییر داد. (_packetSize با S بزرگ)

منابع ترافیکی پروتکل UDP

در ns2 میتوانیم منابع ترافیکی[۲] دیگری به جزء CBR را در پروتکل UDP استفاده کنیم؛ در این حالت با استفاده از توزیع های مختلف برای ایجاد نرخ انتقال بصورت تصادفی (و غیر ثابت)، ترافیک ایجاد می شود؛ از جمله آنها :

- منبع ترافیکی exponential (نمایی) که به صورت on/off می باشد.
- منبع ترافیکی pareto که به صورت on/off میباشد .
- منبع ترافیکی trace (مبتنی بر رد یابی[۳])

هستند.
هر یک از این منابع ترافیکی، پارامترهای مشخصی دارند مثلاً برای تعریف یک منبع ترافیکی که از توزیع نمایی exponential یا توزیع پرتو pareto استفاده می کند، به این ترتیب عمل میکنیم؛

```
set source [ new Application/Traffic/Exponential]
```

یا

```
set source [ new Application/Traffic/Pareto]
```

این منابع ترافیکی ، پارامترهایی از این قبیل دارند :

- _packetSize که اندازه بسته را بر حسب بایت (byte) مشخص می کند.

۱- Flow ID
۲- Traffic Resources
۳- Trace-driven

- _brust_time که متوسط زمان on بودن را مشخص می کند.
- _idle_time که متوسط زمان off بودن را مشخص می کند.
- _rate که نرخ انتقال[1] در دوره های on بودن را مشخص می کند .

به علاوه در منبع Pareto on /off یک پارامتر به نام شکل به صورت _shape نیز تعریف می کنیم .
مثلاً برای تعریف یک منبع ترافیکی از نوع توزیع Pareto و تولید اعداد تصادفی از آن داریم :

```
set source [new Application/Traffic/Pareto]
$source set packetSize_500
$source set brust_time_200ms
$source set idle_time_400ms
$source set rate_100k
$source set shape_1.5
```

در مورد منبع ترافیکی Trace باید یک فایل باینری ایجاد کنیم که شامل زمانهای ورود بسته (inter-packet time) با واحد میلی ثانیه (msec) و اندازه بسته (packet size) با واحد بایت (byte) می باشد .
و به این ترتیب عمل می کنیم :

```
1. set tracefile [new Tracefile]
2. $tracefile filename <file>
3. set src [new Application/Traffic/Trace]
4. $src attach-tracefile $tracefile
```

که Tracefile (با حرف T بزرگ،) یک کلاس از پیش تعریف شده است و بیان میکند قرار است ترافیک از روی یک فایل trace ایجاد شود. یک شی از این کلاس با نام tracefile (با حرف t کوچک) ایجاد کرده ایم و در طول برنامه این نام مشخص کننده کلاس Tracefile از پیش تعریف شده است.

زمانبندی رخداد ها[2]

NS یک شبیه سازی مبتنی بر رخداد و حالت–گسسته است. اسکریپت های TCL، زمانی که یک رخداد باید اتفاق بیفتد، تعریف می شوند. دستور مقدار دهی اولیه [new Simulation] set ns ، یک زمانبند رخداد ایجاد می کند و سپس رخدادها به شکل دستور زیر زمانبندی میشوند:

$ns at زمان رخداد

زمانبند ، زمانی شروع به کار می کند که ns اجراء شود یعنی پس از اجرای دستور

```
$ns run
```

در مثال سناریو باید زمان شروع و پایان کاربردهای FTP و CBR را مشخص کنیم .این کار با دستورات زیر صورت می گیرد :

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

۱- Transmission rate
۲- scheduling events

بنابراین FTP در بازه زمانی 1.0 تا 124.0 ثانیه فعال می شود (یعنی در لحظه ۱/۰ ثانیه ترافیک FTP شروع شده و در لحظه ۱۲۴/۰ پایان می یابد) و CBR در بازه زمانی 0.1 تا 124.0 ثانیه فعال میشود (یعنی در لحظه ۰/۱ ثانیه ترافیک CBR شروع شده و در لحظه ۱۲۴/۰ پایان می یابد).

حال نوبت به اجرای شبیه سازی می رسد؛ اگر این فایل را با نام scenario.tcl ذخیره کرده باشیم، برای اجرای آن در محیط xwin، تایپ می کنیم :

```
ns  scenario.tcl
```

و اگر مایل باشیم آنرا مستقیماً با nam مشاهده کنیم، تایپ می کنیم :

```
nam scenario.tcl
```

بصری سازی با استفاده از برنامه nam

برای ثابت کردن محل قرار گرفتن هر گره در برنامه TCL، از دستور orinet استفاده می کنیم .

به عنوان مثال :

```
$ns  duplex-link-op  $n0  $n2  orinet  right-down
```

تذکر : اگر موقعیت node ها به طور ثابت تعریف نشوند، در هر بار اجرای برنامه، موقعیت آنها به طور تصادفی انتخاب می شوند و این کار مانع بررسی دقیق شبیه سازی می شود .

در زیر برخی از دستورات که در nam کاربرد دارد ذکر شده اند :

۱- **رنگ آمیزی گره ها** : برای مثال، اگر بخواهیم گره n0 با رنگ قرمز ظاهر شود می نویسیم :

```
$n0 color  red
```

۲- **شکل گره ها** : به طور پیش فرض، گره ها دایره ای هستند ولی می توان آنها را به صورت مربعی (box) یا شش گوش (hexagon) هم نمایش داد :

```
$n1 shape box
```

۳- **رنگ آمیزی لینک ها** : مثلاً می خواهیم رنگ لینک میان n0 و n2 را سبز کنیم :

```
$ns duplex-link-op  $no  $n2 color "green"
```

۴- **مارک کردن گره ها** : عمل مارک کردن با دستور add-mark و عمل حذف مارک با دستور delete-mark انجام میشود. مثلا میخواهیم گره n3 را در زمانی که شروع به انتقال فایل می کند و تا زمانی که انتقال فایل ادامه دارد، مارک کنیم و یک مربع آبی دور آن بکشیم :

```
$ns at 2.0 “$n3 add-mark mark3 blue box”
$ns at 30.0 “$n3 delete-mark mark3”
```

که نام این عمل مارک کردن را mark3 گذاشته ایم.

۵- **اضافه کردن برچسب (Label) روی یک گره** : این کار با دستور label انجام میشود. مثلا می خواهیم زمانی که گره n3 شروع به ارسال بسته می کند زیر آن بنویسیم active node :

```
$ns at 1.2   "$n3 label \ "active node\""
```

ویا روی لینک n0 و n2 بنویسیم TCP:

```
$ns duplex-link-op $n0 $n2 label "TCP link"
```

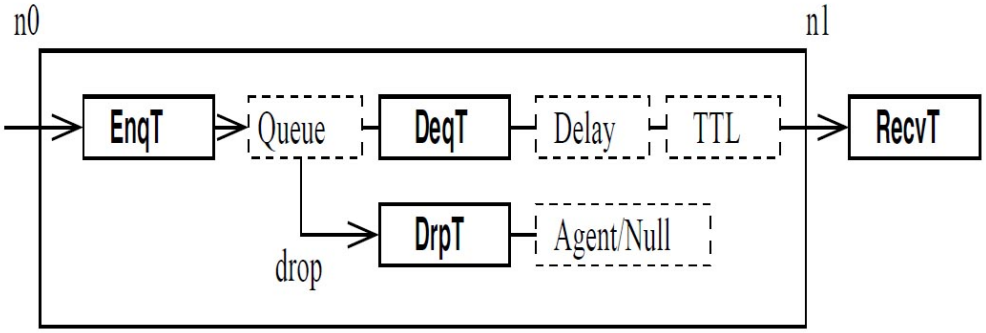
ردیابی[1]

الف) اشیاء ردیابی[2]

شبیه ساز ns همانطورکه می تواند شبیه سازی را به کمک برنامه nam و به صورت بصری و گرافیکی ایجاد کند، به همان خوبی می تواند یک فایل اسکی (ascii) که حاوی اطلاعات ردیابی شبیه سازی است، تولید نماید.

وقتی از ردیابی استفاده می کنیم، چهار شیء را در لینک قرار می دهد:

Enqt و Deqt و Recvt و Drpt که در شکل ملاحظه می شوند:



شکل ۴-۱ : اشیاء ردیابی در لینک یکطرفه

Enqt، اطلاعات بسته هایی را ثبت می کند که وارد لینک شده و در صف ورودی لینک به صف می شود. اگر بسته ها سرریز شوند، اطلاعات این بسته ها (که گم شده اند) توسط Drpt کنترل می شود. Deqt حاوی اطلاعات بسته هایی است که از صف خارج می شوند و در نهایت Recvt اطلاعاتی راجع به مقصد رسیده اند به ما می دهد.

ns ، امکان کسب اطلاعات بیشتر از طریق ردیابی را فراهم می آورد. یکی از این راهها، نظارت صف [3] است که بعدا به توضیح آن خواهیم پرداخت.

ب) ساختار فایل های ردیابی

وقتی عملیات ردیابی روی یک فایل ascii ذخیره شود، به این معنی است که رخدادهای صورت گرفته طی فرایند شبیه سازی، به همراه زمانهای اتفاق افتادن آنها، در این فایل ثبت میشوند؛ این فایل شامل۱۲ فیلد به شرح زیر خواهد بود:

۱- Tracing

۲- Tracing object

۳- Queue monitoring

Event	time	From node	To node	Pkt type	Pkt size	Flags	Fid	Src addr	Dst addr	Seq num	Pkt id
۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲

۱- اولین فیلد مربوط به رخداد است. این فیلد، برابر یکی از چهار نماد ممکنِ +، -، r یا d خواهد بود.

r : «رسید» (recieved) زمانی که بسته داده، به قسمت خروجی لینک میرسد

+ : وارد شدن بسته به صف (enqueued)

- : خارج شدن بسته از صف (dequeued)

d : گم شدن –جا ماندن از صف(dropped)

۲- فیلد دوم، زمان اتفاق افتادن هر رخداد است

۳- فیلد سوم، گره ورودی لینک که رخداد در آن اتفاق می افتد.

۴- فیلد چهارم، گره خروجی لینک که رخداد در آن اتفاق می افتد.

۵- نوع بسته (مثل TCP یا CBR و....). این مقدار همان متغیری است که برای هر پروتکل یا کاربرد تعریف کرده ایم مثل cbr و....

۶- اندازه ی بسته

۷- مقادیر پرچم ها (flags) که بعدا خواهیم دید چگونه تعیین می شوند.

۸- شناسه جریان(flow ID) که مثلا برای tcp آن را ۱ و برای cbr آن را ۲ در نظر گرفته ایم.

۹- آدرس مبدا که به شکل node. port نمایش داده می شود.

۱۰- آدرس مقصد که به شکل node. port نمایش داده می شود.

۱۱- شماره ترتیب بسته در پروتکل لایه شبکه است. قابل ذکر است که هرچند در پیاده سازی UDP در شبکه های واقعی از شماره ترتیب استفاده نمیشود، با این وجود ns شماره ترتیب بسته ها را در UDP هم ثبت میکند؛ این کار بدلیل اهداف آنالیزی صورت میگیرد.

۱۲- شناسه منحصر بفرد برای هر بسته

در زیر، نمونه ای از فایل Trace را مشاهده می کنید.

1	2	3	4	5	6	7	8	9	10	11	12
r	2.176155	0	2	tcp	592	-------	1	0.0	4.0	19	35
+	2.176155	2	3	tcp	592	-------	1	0.0	4.0	19	35
r	2.178523	0	2	tcp	592	-------	1	0.0	4.0	20	36
+	2.178523	2	3	tcp	592	-------	1	0.0	4.0	20	36
d	2.178523	2	3	tcp	592	-------	1	0.0	4.0	20	36
+	2.191941	2	3	tcp	592	-------	1	0.0	4.0	21	37
r	2.194309	0	2	tcp	592	-------	1	0.0	4.0	22	38
+	2.194309	2	3	tcp	592	-------	1	0.0	4.0	22	38
d	2.194309	2	3	tcp	592	-------	1	0.0	4.0	22	38
r	10.034005	2	0	ack	40	-------	1	4.0	0.0	169	379
+	10.034005	0	2	tcp	592	---A---	1	0.0	4.0	170	385
-	10.034005	0	2	tcp	592	---A---	1	0.0	4.0	170	385

```
r 10.039632 3 2 ack 40 ------- 1 4.0 0.0 169 380
+ 10.039632 2 0 ack 40 ------- 1 4.0 0.0 169 380
- 10.039632 2 0 ack 40 ------- 1 4.0 0.0 169 380
```

تذکر ۱ : این فایلِ ردیابی، دستکاری شده است؛ به این معنی که در این فایل روال منطقی میان دریافتها و ارسالها و ... وجود ندارد و صرفا بمنظور آشنایی با شکل فایلهای ردیابی نوشته شده اند.

تذکر ۲ : مقادیر پرچمها روی خط چینها مشخص میشوند (در اینجا مقدار A رخ داده است).

ج)ردیابی زیر مجموعه ای از رخدادها

در قسمت های قبل نحوه ردیابی کل رخدادهای شبیه سازی (trace-all) را دیدیم. حالا می خواهیم نحوه ردیابی تنها زیر مجموعه ای از رخدادها را بررسی کنیم.

یکی از روشها، جایگزین کردن دستور $ns trace-queue به جای دستور $ns trace-all <filename> است؛ مثلا در دستور زیر :

```
$ns trace-queue $n2 $n3 $file1
```

فایل ردیابی ای ایجاد می شود که تنها شامل ردیابی رخدادهایی است که روی لینک متصل کننده n2 و n3 اتفاق می افتند؛ دستور مشابه آن در برنامه nam بصورت namtrace-queue (به جای trace-queue) میباشد.

تذکر: دستور trace-queue باید دقیقا پس از تعریف لینک ها نوشته شود.

راه دیگر برای ردیابی قسمتی از رخدادها، استفاده از دستورات Unix در کد Tcl جهت فیلتر کردن رخدادهاست که در قسمت های بعدی به آنها پرداخته می شود.

پردازش فایل های داده ای به کمک awk

AWK یک زبان اسکریپتی کمکی است (که سرنام ایجاد کنندگان آن Kernighan و Weinberger و Aho میباشد) و برای دستکاری داده ها و ایجاد گزارشات مطلوب کاربرد دارد. همانطور که گفته شد فایل متنی trace (مثل out.tr) شامل ۱۲ ستون است؛ اما همیشه تمام این ستونها مطلوب ما نیست. بعنوان مثال فقط میخواهیم میزان بسته هایی که در لحظات مختلف زمان به گره n5 میرسند بررسی کنیم. در این حالت تنها دو ستون *بسته های رسیده* و *زمان رسیدن بسته ها* را نیاز داریم؛ استخراج این اطلاعات (دو ستون) به کمک برنامه های کمکی از جمله AWK صورت میگیرد.

به عنوان مثال با کمک کدهای AWK، می توان براحتی میانگین یا مجموع یک ستون خاص از یک فایل داده ای را محاسبه نمود.

مثال زیر نحوه محاسبه میانگین از اعداد ستون ششم ($6) یک فایل را نشان می دهد :

```
{BEGIN { FS = " "} { nl++ } { s=s+$6} END {print "average:" s/nl
```

nl بیانگر سطر جدید (new line) است.

S بیانگر حاصل جمع است که هر بار با ستون ششم سطر بعدی جمع میشود.

تذکر: اگر ستونها با tab جدا شده باشند، از "\t" استفاده می شود. اگر از فاصله (space) برای جدا سازی ستونها استفاده شده باشد،" " را جایگزین عبارت "\t" می کنیم.

این برنامه AWK را با نام دلخواه مثل Average.awk ذخیره می کنیم؛ بنابراین برای محاسبه ی میانگین ستون ششم داده های یک

فایل مثل out.tr به کمک برنامه Average.awk، دستور زیر را در محیط Unix (یا cygwin) مینویسم:

```
awk  -f average.awk out.tr
```

تذکر: awk (با حروف کوچک) دستوری در سیستم عاملهای یونیکس برای اجرای فایلهای AWK است.

در این حالت، عبارتی مشابه Average:30.8 (که 30.8 میانگین ستون ششم فایل out .tr است) مواجه می شویم. این خط از دستور به بیان ساده اعلام میدارد که برنامه average.awk را روی فایل موجود out.tr اعمال کن!

تذکر ۱: اگر بخواهیم این مقدار را در یک فایل ذخیره کنیم می نویسیم:

```
awk  -f average.awk out.tr > file
```

تذکر ۲: فایلی که در اختیار داریم چند ستون دارد که شماره گذاری این ستونها از یک شروع می شود.

در محاسبه عملکرد [۱] شبیه سازی، مثل مقدار بسته های دریافت شده توسط هر گره مقصد (در اینجا گره ۴ برای tcp و گره ۵ برای udp) در واحد زمان، از یک کد نوشته شده به زبان AWK استفاده می کنیم. این کد به صورت زیر است:

```
#throughput for tcp (between n0 and n4) & throughput for udp (between n1 and n5)
BEGIN {
last = 0
f1 = 0
f2 = 0
total = 0
}
{
if($5=="tcp" && $1=="r" && $4=="4 "){
f1 +=$6
}
if($5=="cbr" && $1=="r" && $4=="5 "){
f2 +=$6
}
total +=$6
}
#every  second
if ($2 - 1> last) {
last = $2
print $2 , (f1*8/1000000) , (f2*8/1000000) , (total*8/1000000)
f1 = 0
f2 = 0
total = 0
 }
}
END {
print $2 , (f1*8/1000000) , (f2*8/1000000) , (total*8/1000000)
}
```

کد نوشته شده به زبان AWK برای محاسبه نحوه عملکرد پروتکلهای tcp و udp در ارسال بسته ها

پس یک فایل Trace به کمک فایل throughput.awk ایجاد می کنیم که شامل حجم بسته ها و زمان دریافت آنها می باشد:

۱- throughput: عملکرد، توان عملیاتی

```
awk  -f throughput.awk out.tr>ThrData.tr
```

فایل out.tr، فایل ردیابی کل سیستم است؛ به کمک برنامه throughput، تنها رخدادهای TCP و UDP مربوط به گره مقصد را ثبت کرده و آن را در فایل ThrData.tr ذخیره می کنیم.

تذکر: فایل ThrData.tr چهار ستون دارد. اولین ستون مربوط به زمان رسیدن بسته ها به مقصدهایشان، دومین ستون مربوط به حجم بسته های رسیده به مقصد (یعنی گره n4) توسط TCP ، سومین ستون مربوط به حجم بسته های رسیده به مقصد (یعنی گره n5) توسط CBR و چهارمین ستون مربوط به حجم کل بسته های به مقصد رسیده (چه از طریق tcp و چه از طریق cbr) میباشند.

با استفاده از فایل ThrData.tr و برنامه gnuplot می توان *نمودار* کارایی TCP و UDP را نیز رسم نمود. به کمک قطعه کد زیر، که به زبان perl نوشته شده است، فایل ThrData.tr را به عنوان ورودی (برای رسم نمودارهای آن) تعریف میکنیم؛ و کل فایل را با نام plot ذخیره می کنیم.

```
set term postscript eps enhanced color
set title "Throughput"
set xlabel "Time (s) "
set ylabel "Throughput (Mbps ) "
set output "UdpTcptp.eps"
plot "ThrData.tr" using 1:2 title "TCP" with linespoints ,\
"ThrData.tr" using 1:3 title "CBR" with linespoints
```

حال با دستور gnuplot plot فایلی ایجاد می شود که پسوند آن eps[۱]. است، نام و فرمت این فایل را در کد plot می بینید. بنابراین UdpTcptp.eps (خط پنجم کد را ببینید!) حاوی فایل گرافیکی و نمودار بازدهی (کارایی) است. این فایل را میتوان با برنامه هایی نظیر Adobe Illustrator مشاهده نمود؛ همچنین میتوان به کمک نرم افزارهای کمکی، آن را به فرمتهای دیگر عکسی مثل bmp، jpeg و ... تبدیل نمود.

۱- Encapsulated PostScript

منابع و مآخذ

1- The *ns* Manual (formerly *ns* Notes and Documentation), Kevin Fall and Kannan Varadhan, January 6, 2007
2- Introduction to Network Simulator NS2, Teerawat Issariyakul and Ekram Hossain, Springer, 2009
3- NS simulator for beginners, Eitan Altman and Tania Jimenez, 2003
4- http://www.isi.edu/nsnam/ns/
5- Tcl and ns2 Tutorial, Neal Charbonneau, nealc.com, 2010
6- Tcl/Tk Programming for the Absolute Beginner, KURT WALL, THOMSON pub