دانشگاه علم و صنعت ایران

مرکز آموزش‌های الکترونیکی

گروه مهندسی فناوری اطلاعات و ارتباطات

شبیه‌سازی شبکه‌های VANET با استفاده از شبیه‌ساز NS2

پروژه درس شبکه‌های کامپیوتری پیشرفته

اساتید درس: **دکتر فتحی**
مهنس غفاریان

تهیه کنندگان: **داریوش اکبری تولون**
آزیتا پلوان
علیرضا قاضی سعیدی

مرداد ۸۸

چکیده:

هدف از این گزارش ارائه نمونه‌های شبیه‌سازی شده شبکه‌های VANET با استفاده از شبیه‌ساز NS2 بوده است. شبکه‌های VANET نوعی از شبکه‌های Ad_hoc هستند که ارتباط بین وسایل نقلیه مجاور، همچنین بین وسایل نقلیه و تجهیزات ثابتی که معمولا کنار جاده‌ها نصب می‌شوند را فراهم می‌آورند، هدف اصلی در شبکه‌های VANET ایجاد راحتی و امنیت بیشتر برای مسافران می‌باشد که از طریق هشدار تصادف، اعلان شرایط ترافیکی و علائم رانندگی و در جاده‌ها و سایر موارد مشابه انجام می‌شود. شبکه‌های VANET موضوعات تحقیقاتی جدید و رو به گسترشی را ایجاد کرده‌اند، از آنجایی‌که ایجاد محیط واقعی برای بررسی موضوعات مختلف مربوط به این نوع شبکه‌ها با مشکلات و پیچیدگی‌های بسیاری مواجه خواهد بود و نیز هزینه زیادی را می‌طلبد استفاده از شبیه‌سازهای مطرح از جمله NS2 روش مناسبی برای حل این نوع مشکلات می‌باشد.

در این گزارش ضمن ارائه توضیحات جامعی در خصوص شبیه‌ساز NS2، قابلیت‌ها، نحوه نصب و شبیه‌سازی با آن، شبکه‌های VANET مورد بررسی قرار گرفته و چند نمونه شبیه‌سازی شده برای این شبکه‌ها ارائه شده است.

فهرست مطالب

عنوان **شماره صفحه**

فصل ۱- معرفی شبیه ساز NS2 ..... ۶
۱-۱- مقدمه ۷
۱-۲- نصب شبیه‌ساز ۷
1-2-1- محیط مورد نیاز و نحوه نصب N.S. ۷
1-2-2- دانلود کردن N.S. ۷
1-2-3- نصب N.S. ۷
۱-۲-۴- نصب NS2 در محیط ویندوز ۱۰
1-3- شروع کار با N.S. ۲۶
1-4- شبیه‌سازی N.S. ۲۷
1-4-1- OTCL زبان مورد استفاده کاربر ۲۹
1-4-2- یک مثال ساده از شبیه‌سازی ۳۱
1-4-3- زمان‌بند رخداد (Event Scheduler) ۳۷
1-4-4- اجزای شبکه Network components ۳۹
1-4-5- گره و مسیریابی ۴۰
1-4-6- ردیابی (Tracing) ۴۱
1-4-7- بسته ۴۳
1-4-8- مثال تجزیه و تحلیل ردیابی ۴۴
1-4-9- مثالی از نمایشگر صف RED ۴۷
1-4-10- چه چیزی را کجا پیدا کنیم؟ ۵۰
1-4-11- پیوند OTCL ۵۲
1-4-12- صادر کردن کلاس از C++ به OTCL ۵۲
1-4-13- صادر کردن متغیر از C++ به OTCL ۵۳
1-4-14- اضافه کردن یک کاربرد و یک عامل (Agent) جدید ۵۷
1-4-15- اضافه‌کردن یک صف جدید ۶۲
1-4-16- مثال LAN ۶۵
1-4-17- مثال چندپخشی (Multicasting) ۶۷
1-4-18- مثال وب سرور ۶۹
1-5- معرفی برنامه NAM ۷۲

۱-۶- دستورات اولیه درNS2 ..... ۷۶
فصل ۲- معرفی شبکه‌های VANET ۸۲
۲-۱- مقدمه ۸۳
۲-۲- بررسی شبکه‌هایVANET ۸۳
2-3- تکنولوژی‌ها و پروتکل‌های شبکه‌های VANET ۸۵
فصل ۳- شبیه‌سازی شبکه‌های VANET درNS2 ..... ۸۷
۳-۱- مقدمه ۸۸
۳-۲- نمونه شبیه‌سازی شده ۱ ..... ۸۹
۳-۲-۱- طرز اضافه کردن یک پروتکل جدید به NS-2 ..... ۸۹
۳-۲-۲- برنامه tcl با چهار گره موبایل ۹۶
3-3- نمونه شبیه‌سازی شده ۲ ..... ۱۰۷
۳-۴- نمونه شبیه‌سازی شده ۳ ..... ۱۱۲
۳-۴-۱- پیکربندی ۱۱۹
۳-۴-۲- مراحل شبیه‌سازی ۱۲۰
۳-۴-۳- پیاده‌سازی یک مثال ۱۲۲
3-4-4- تولید نقشه ۱۲۶
۳-۴-۵- تولید حرکت خودروها ۱۳۴
۳-۴-۶- تعریف جریان ۱۳۴
۳-۴-۷- تعیین دور ۱۳۶
۳-۴-۸- حرکت خودکار خودروها ۱۳۸
۳-۴-۹- تنظیمات شبیه‌سازی ۱۳۹
منابع و مراجع ۱۴۵

فهرست اشکال

شکل ۱: : اجزای کلی ns ۲۸
شکل ۳: توپولوژی یک شبکه ساده ۳۴
شکل ۴: زمان‌بند رخداد گستر ۳۷
شکل ۵: بخشی از سلسله مراتب کلاس ۳۹
شکل ۶ :گره چندپخشی و تک پخشی ۴۰
شکل ۷ :لینک ساده ۴۱
شکل ۸: واردکردن شیئی‌های ردیابی ۴۲
شکل ۹: مشاهده صف ۴۲
شکل ۱۰: مثالی از حرکت بسته ۴۳
شکل ۱۱: فرمت بسته در NS ۴۴
شکل ۱۲: مثالی از فرمت ردیابی ۴۵
شکل ۱۳: لزرش CBR در گره n3 ..... ۴۷
شکل ۱۴: تنظیمات مربوط به مثال نمایشگر صف RED ۴۸
شکل ۱۵: گراف ردیابی صف Red ۵۰
شکل ۱۶: ساختار شاخه‌ها در NS ۵۱
شکل ۱۷: مثال C++ از جزء شبکه و شیئ پیوند ۵۳
شکل ۱۸: مثالی از تخصیص متغیرها ۵۳
شکل ۱۹: مثالی از مفسر دستور OTCL ۵۴
شکل ۲۰: اجرای دستور OTCL از شیئ C++ ۵۴
شکل ۲۱:ex.linkage.tcl ۵۶
شکل ۲۲: نتیجه مربوط به برنامه ex.linkage.tcl ۵۶
شکل ۲۳: ساختار سرآمد MM و کلاس آن ۵۸
شکل ۲۴: اضافه کردن به فایل packet که از نوع C++ است. ۵۹
شکل ۲۵: اضافه کردن به فایل ns-packet.tcl که از نوع OTCL است. ۵۹
شکل ۲۶: ارسال اجرای تایمر ۶۰
شکل ۲۷: افزودن دو تابع محلی به کلاس Agent ۶۱
شکل ۲۸: افزودن دو تابع محلی به کلاس Application ۶۱
شکل ۲۹: تنظیم مقادیر اولیه پارامترها ۶۱
شکل ۳۰: توپولوژی و سناریوی شبیه‌سازی برنامه Test ۶۲
شکل ۳۱: پیاده‌سازی کلاس DtRr Queue ۶۵
شکل ۳۲: برنامه آزمایش Dr Rr Queue ۶۵
شکل ۳۳: توپولوژی شبکهٔ LAN و شبیه‌سازی آن ۶۷
شکل ۳۴: صفحهٔ NAM مربوط به شبیه‌سازی ارسال چندپخشی ۶۹
شکل ۳۵: توپولوژی شبکه web server و شبیه‌سازی آن ۷۱

شکل ۳۶: معرفی محیط ابزار Network Animator ۷۳
شکل ۳۷: خروجی Tcl Code در Nam . بسته ها بصورت پیکانهایی نمایش داده میشوند ۷۵
شکل ۳۸: مونیتور کردن یک لینک ۸۱
شکل ۳۹: محیط نرم افزار xgraph ۱۰۰
شکل ۴۰: باز کردن فایل Trace ۱۰۱
شکل ۴۱: صفحه دانلود نرم‌افزار sumo ۱۱۳
شکل ۴۲: محیط نرم‌افزار sumo ۱۱۴
شکل ۴۳: ارتباط نرم‌افزار SUMO و NS2 و MOVE ۱۲۲
شکل ۴۴: انتخاب سیستم عامل سیستم ۱۲۵
شکل ۴۵: مشخص کردن مسیر نرم‌افزار sumo در محیط نرم افزار move ۱۲۶
شکل ۴۶: ایجاد نقشه به صورت دستی ۱۲۶
شکل ۴۷: وارد کردن گرهها به صورت دستی ۱۲۷
شکل ۴۸: انتخاب منوی Edge از منوی اصلی ۱۲۸
شکل ۴۹: پنجره ظاهر شده بعد از انتخاب edg ۱۲۸
شکل ۵۰: تولید نقشه ۱۳۳
شکل ۵۱: پنجره ظاهر شده برای تعریف جریان ۱۳۴
شکل ۵۲: پنجره ظاهر شده پس از کلیک کردن بر روی flow ۱۳۵
شکل ۵۳: تعیین دور در منو اصلی ۱۳۶
شکل ۵۴: پنجره ظاهر شده پس از کلیک بر روی Turn ۱۳۷
شکل ۵۵: حرکت خودکار خودروها ۱۳۸
شکل ۵۶: پنجره ظاهر شده پس از کلیک بر روی create vehicle ۱۳۸
شکل ۵۷: کلیک بر روی گزینه configuration ۱۳۹
شکل ۵۸: پنجره ظاهر شده پس از کلیک بر روی configuration ۱۳۹
شکل ۵۹: پنجره ظاهر شده پس از کلیک بر روی Visualization ۱۴۰
شکل ۶۰: انتخاب برای مدل ترافیک ۱۴۱
شکل ۶۱: مدل ترافیک نرم افزار ns2 ..... ۱۴۱
شکل ۶۲: خروجی برای ns2 تولید کردن ۱۴۲
شکل ۶۳: وارد کردن اطلاعات مربوط به شبیه‌ساز ns2 ..... ۱۴۳
شکل ۶۴: وارد کردن اطلاعات در NS2 ..... ۱۴۳
شکل ۶۵: خروجی در شبیه‌ساز NAM ۱۴۴

معرفی شبیه ساز NS2

۱-۱- مقدمه

در این فصل از گزارش به معرفی شبیه ساز NS2، طریقه نصب آن در محیط لینوکس و ویندوز پرداخته می‌شود، همچنین در رابطه با قابلیت‌های شبیه ساز، دستورات ساده آن، برنامه نویسی TCL و سایر موراد توضیحات جامعی در قالب مثال‌های مختلف ارائه شده است.

۱-۲- نصب شبیه‌ساز

در این فصل کلیاتی در مورد چگونگی نصب این نرم‌افزار خواهیم داشت.

در طراحی این شبیه‌ساز از زبان برنامه‌نویسی C++ استفاده شده است و همچنین از زبان otcl نیز به عنوان واسط و مترجم فرامین استفاده می‌شود. به علت سرعت بالای C++، از آن برای پیاده‌سازی پروتکل‌ها و پردازش بسته‌های اطلاعات ورودی استفاده می‌شود. اما برای شبیه‌سازی ساختار و توپولوژی شبکه از زبان otcl استفاده می‌شود.

۱-۲-۱- محیط مورد نیاز و نحوه نصب N.S.

شبیه‌ساز N.S. بر روی سیستم‌های عامل مختلف UNIX نظیر Sunos و Linux قابل نصب می‌باشد. البته می‌توان از سیستم عامل windows نیز استفاده نمود، ولی N.S. ذاتاً برای محیط‌های UNIX طراحی شده است.

۱-۲-۲- دانلود کردن N.S.

به دو طریق می‌توان از برنامه N.S. استفاده کرد. روش اول استفاده از بسته‌های نرم‌افزار مختلف (مانند TCL/TK و otcl و ...) و روش دوم، استفاده از بسته نرم‌افزاری all-in-one می‌باشد. پیشنهاد می‌کنیم برای شروع کار با این برنامه از بسته نرم‌افزاری all-in-one استفاده کنید. تنها عیب آن این است که ممکن است بعضی از اجزاء بسته all-in-one مورد استفاده شما نباشد و فضایی از دیسک را اشغال کنند. البته توجه به این نکته ضروری است که بسته نرم‌افزاری all-in-one فقط در سیستمهای بر مبنای Unix کار می‌کنند.

این نرم‌افزار را می‌توان از آدرس اینترنتی http://www.isi.edu.nsnam/dist/ دانلود کرد.

۱-۲-۳- نصب N.S.

به روشهای مختلفی می‌توانید بسته نرم‌افزاری all-in-one را در اختیار داشته باشید. اگر بسته نرم‌افزاری N.S. را با استفاده از سیستم عامل windows، download کرده‌اید به طوریکه سورس این نرم‌افزار روی پارتیشن‌های windows می‌باشد، می‌توانید با استفاده از دستور mount به پارتیشن‌های ویندوز دسترسی پیدا کرده و مراحل نصب را انجام دهید. اگر بخواهید سورس بسته نرم‌افزاری N.S. را روی پارتیشن‌های پیدا

کرده و مراحل نصب را انجام دهید. اگر بخواهید سورس بسته نرم‌افزاری .N.S را روی پارتیشن‌های Linux داشته باشید، باید تنظیمات Internet Explorer محیط Linux را انجام داده و آن را مستقیماً در سیستم عامل Linux، دانلود کنید. همچنین روش سومی نیز وجود دارد که سورس این نرم‌افزار را روی cd داشته باشید که در این حالت باید CDROM را mount کرده و از سورس نرم‌افزار برای نصب آن استفاده کنید (در واقع در این حالت باید سورس مربوط را روی پارتیشن‌های Linux کپی کنید). حال با توجه به اینکه یکی از سه روش بالا به این نرم‌افزار دسترسی پیدا کرده‌اید، مراحل نصب به صورت زیر خواهد بود:

ابتدا فایل ns-allinone-2.1b6.tar (یا نسخه‌های بالاتر آن) که به صورت فشرده شده است را باید از حالت فشرده خارج کنید. اصطلاحاً باید آن را untar کنید، که در سیستم عامل Linux این کار با دستور زیر انجام می‌شود:

```
Tar-xzvfns-allinone-2.1 b6.tar
```

فایلهای مزبور از حالت فشرده خارج می‌شوند. دستورات زیر را اجرا کنید تا برنامه نصب شروع به کار کرده و این نرم‌افزار را روی سیستم نصب کند.

```
Cd ns-allinone-2.1b6
./install
```

بعد از اتمام نصب، در فایل bash-profile، طبق دستوراتی که در انتهای نصب بر روی صفحه مانیتور ظاهر می‌شود، کلیه مسیرها را تنظیم کنید. فایلهای اجرای برنامه .N.S و Nam در شاخه bin وجود دارند. در نهایت باید عبارت مقابل را به دستور path در فایل فوق اضافه کنید تا مراحل نصب پایان یابد.

```
Root/ns-allinone-2.1 b6/bin
```

توجه به این نکته ضروری است که جهت آنکه شبیه‌ساز .N.S به طور صحیح اجرا گردد، باید از TCL نسخه v7.5 یا بالاتر استفاده شود. همچنین چنانچه از gcc به عنوان کامپایلر C++ استفاده می‌شود، باید از نسخه 2.7.2 یا بالاتر استفاده نمود. بعد از نصب .N.S برای اطمینان از صحت عملیات نصب و تست سیستم، می‌توان در بالاترین شاخه، دستور validate/. را اجرا نمود. این دستور یکسری تست‌های استاندارد از قبل تهیه شده برای .N.S را اجرا می‌کند و گزارشی از خروجی هر تست بر روی صفحه مانیتور اعلام می‌شود. برنامه تست validate پروتکل‌های زیر را تست می‌نماید:

- در سطح لایه کاربردی
  - پروتکل HTTP جهت اتصال به web.
  - پروتکل‌های telnet و FTP.
  - منابع ترافیکی از نوع نرخ بیت ثابت (CBR).
  - منابع ترافیکی از نوع ON/OFF.

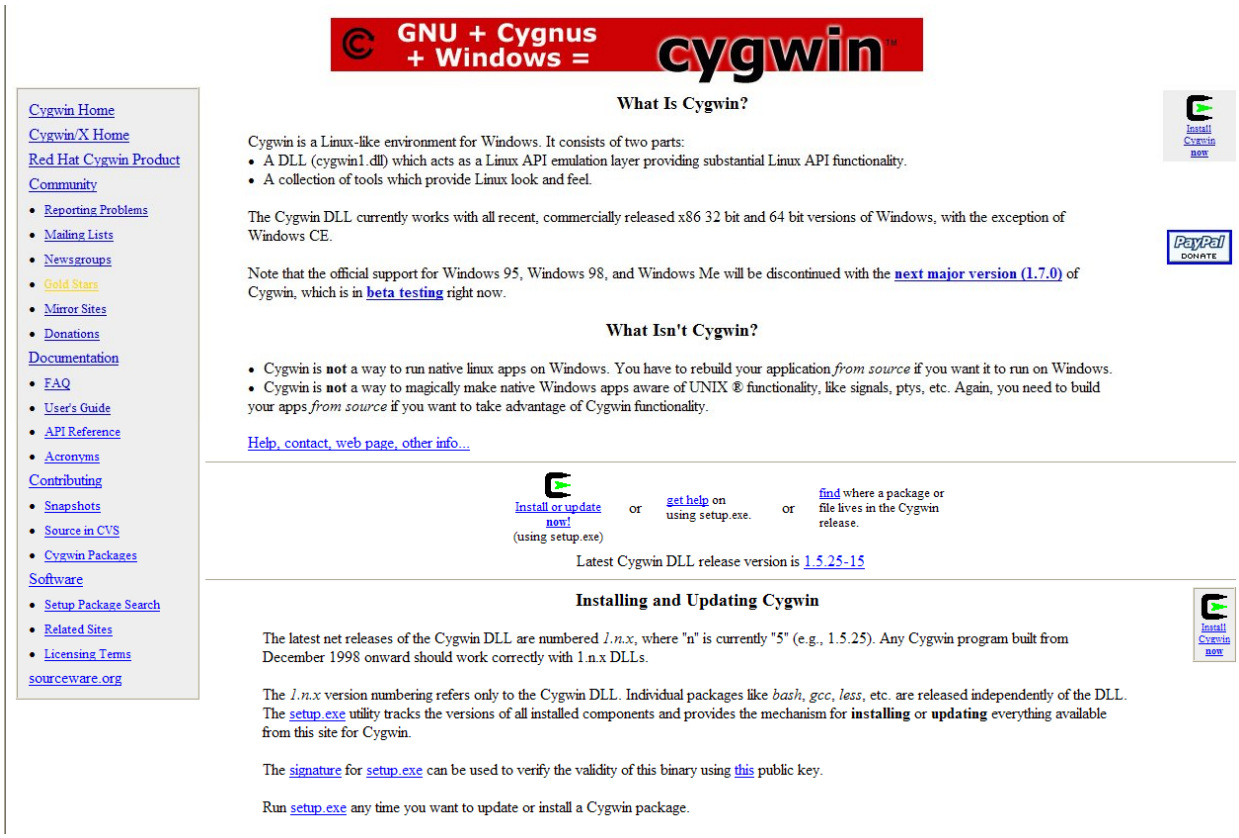
- در سطح لایه ارسال
  - پروتکل TCP پایه.
  - انواع پروتکل‌های TCP شامل: Full، FACK ، SACK، Neew-Reno، Reno و Tahoe
  - پروتکل‌های SRM و RTP.
- پروتکل‌های مسیریابی
  - مسیریابی الگوریتمی
  - مسیریابی سلسله مراتبی
  - مسیریابی شبکه‌های محلی و پراکنشی (Broadcast)
  - مسیریابی Manual
  - مسیریابی Multi cast
  - مسیریابی دینامیکی
  - شبیه‌سازی در سطوح لایه جلسه (Session Level Simulation)
- مکانیسم‌های مسیریابی
  - چندین الگوریتم زمان‌بندی صف شامل الگوریتم‌های:
    - FQ– (Fair Queuing)
    - DRR– (Defict Round Robin)
    - SFQ– (Stochastic Fair Queuing)
    - FIFO– (First In First out)
    - CBQ– (Class Based Queue)
  - مدیریت صف از نوع RED
  - الگوریتم‌های کنترل دسترسی شامل : MS, HB, ACTP, ACTD
- مکانیزمهای لایه پیوند داده
  - شبکه‌های محلی LAN با پروتکل CSMA/CD

در شبیه‌ساز .N.S علاوه بر پروتکل‌های فوق که ذکر گردید، پروتکل‌های زیر نیز موجوداند که در برنامه تست Validate در نظر گرفته نشده‌اند:

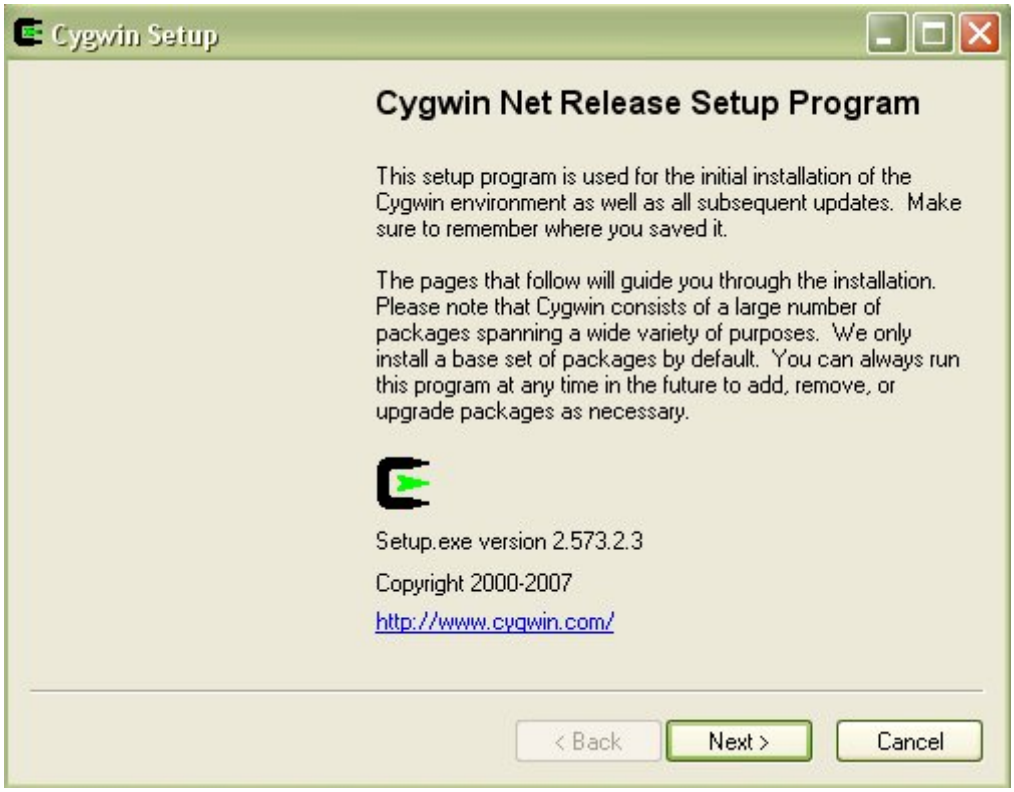
- پروتکل ASYMTCP
- پروتکل‌های RLM, SNOOP, RTP, RTCP
- شبکه‌های محلی با پروتکل‌های CSMA/CA
- فیلترهای Token Bucket
- منابع Trace-generated
- گیرنده‌های Delay-adaptive
- ماژول‌های تأخیر
- IVS

۱-۲-۴- نصب NS2 در محیط ویندوز

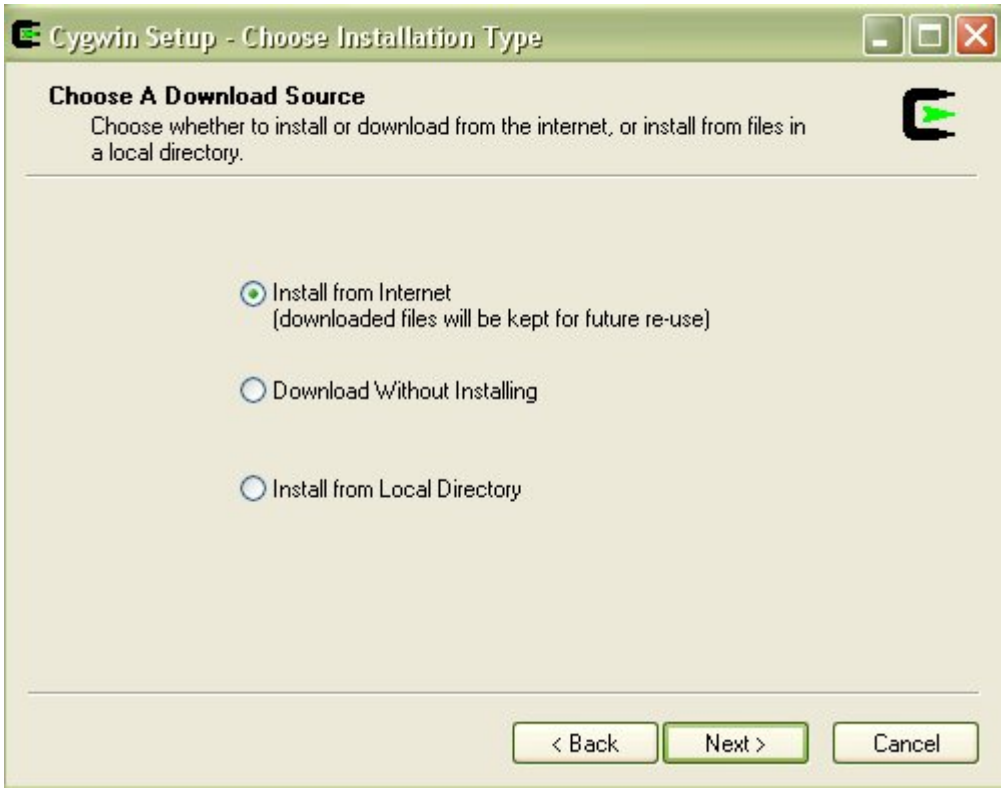
برای نصب ns درویندوز نیاز است ابتدا برنامه ای به نام cygwin نصب گردد، تا محیط لینوکس را در ویندوز شبیه سازی کنیم. بدین منظور ابتدا برنامه cygwin را نصب می کنیم. صفحه اینترنتی http://www.cygwin.com/ را باز می کنیم.



روی لینک install cygwin now که در گوشه سمت بالای سمت راست صفحه قرار دارد کلیک نموده، در این مرحله پنجره زیر ظاهر می شود.

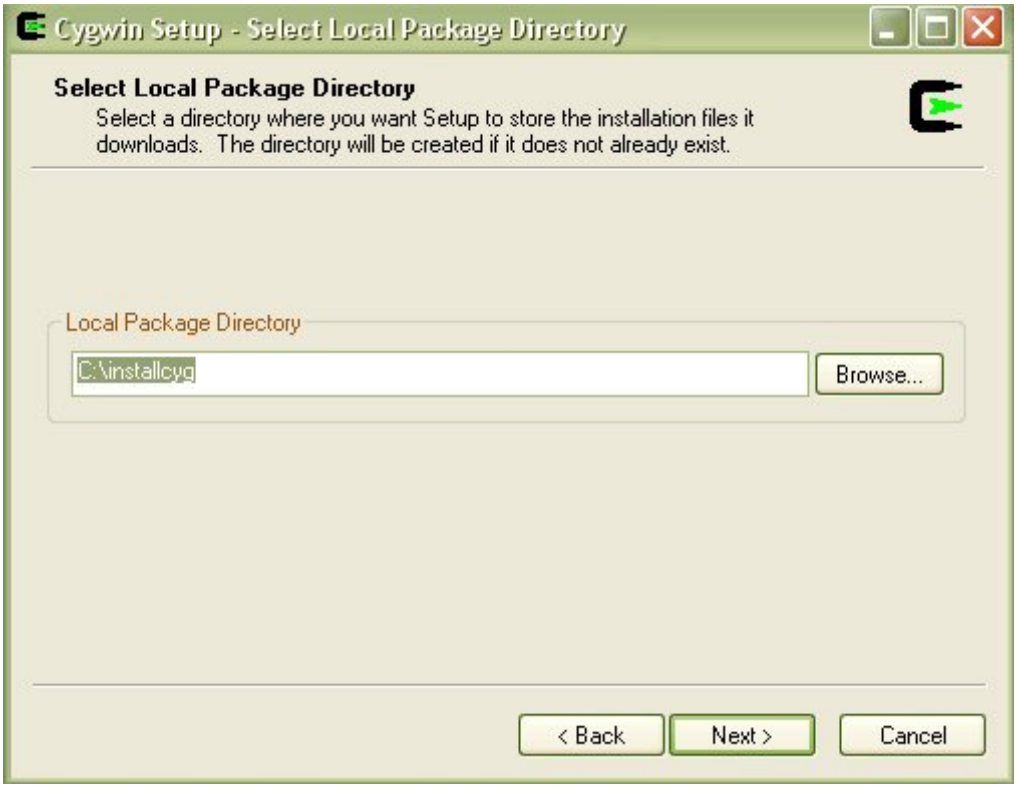


گزینه next را انتخاب می کنیم.

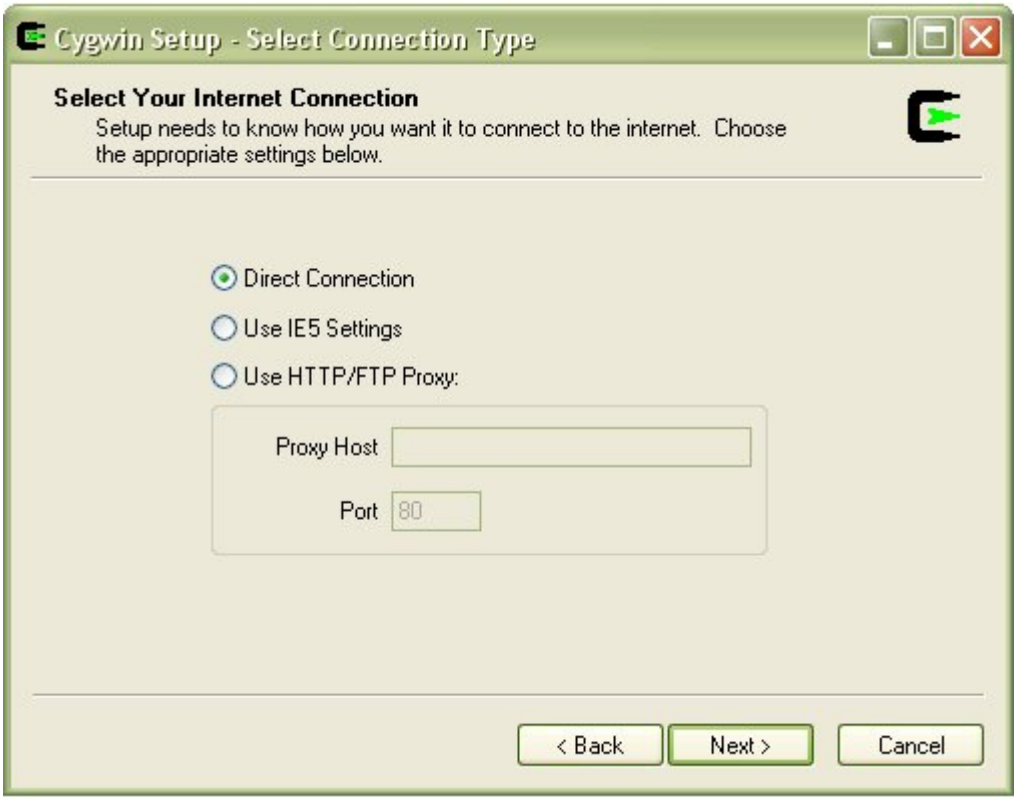


گزینه install from internet را انتخاب می کنیم .این گزینه فایل های مورد نیاز برای نصب cygwin را در پوشه مورد نظر دانلود خواهد کرد.حال دکمه next را فشار می دهیم. در این صورت پنجره زیر خواهد آمد.

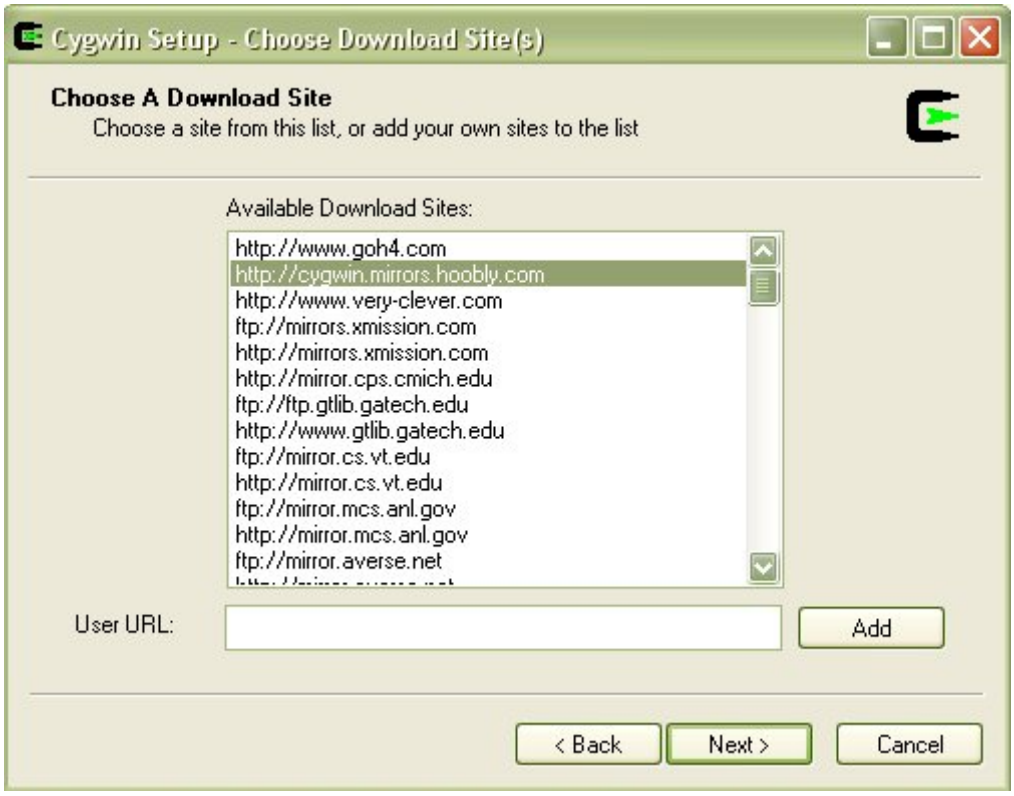
پوشه ای را که قرار است cygwin در ان نصب شود در قسمت root directory می نویسیم.پیشنهاد می شود که این گزینه را نیز به صورت پیش فرض خود رها کنیم. سایر گزینه ها را نیز بصورت پیش فرض رها کرده و دکمه next را انتخاب می کنیم. سپس پنجره زیر را مشاهده می کنیم.



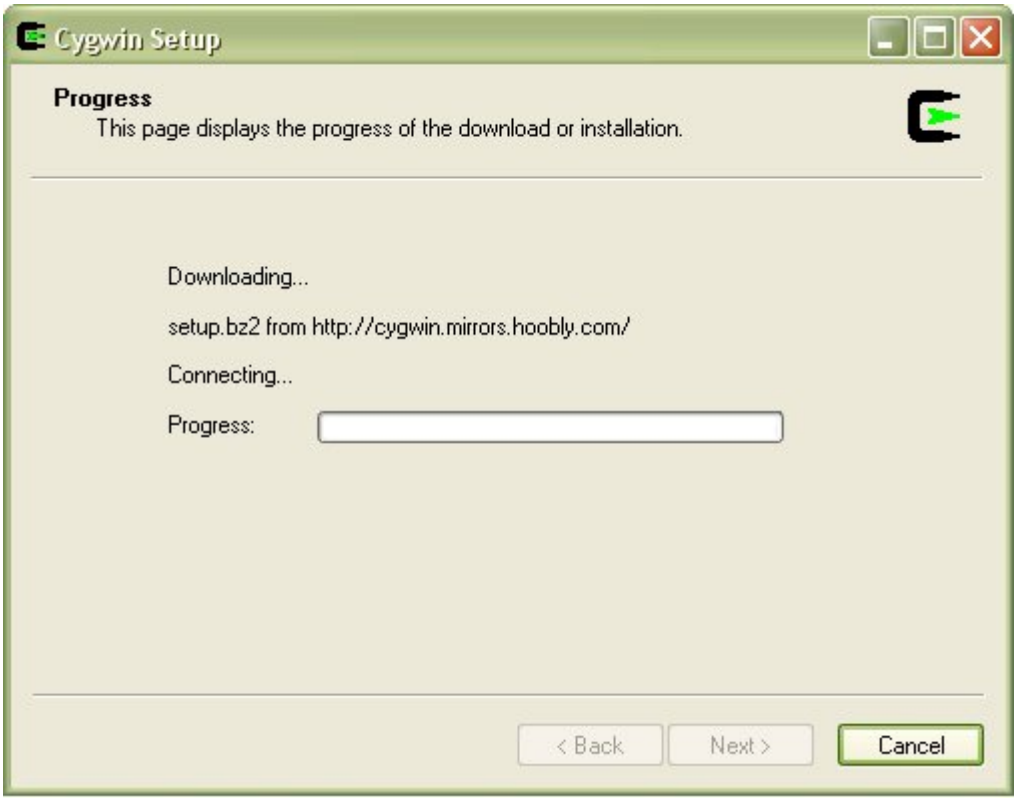
در این مرحله باید نام و مکان پوشه ای را انتخاب کنیم که فایل cygwin.exe در آنجا ذخیره خواهد شد. سپس دکمه next را انتخاب می کنیم. در این حالت پنجره زیر ظاهر می شود. اگر برای ارتباط با اینترنت ازproxy استفاده می کنیم باید در این پنجره تنظیمات لازم را انجام دهیم در غیر این صورت گزینه direct connection را انتخاب می کنیم.

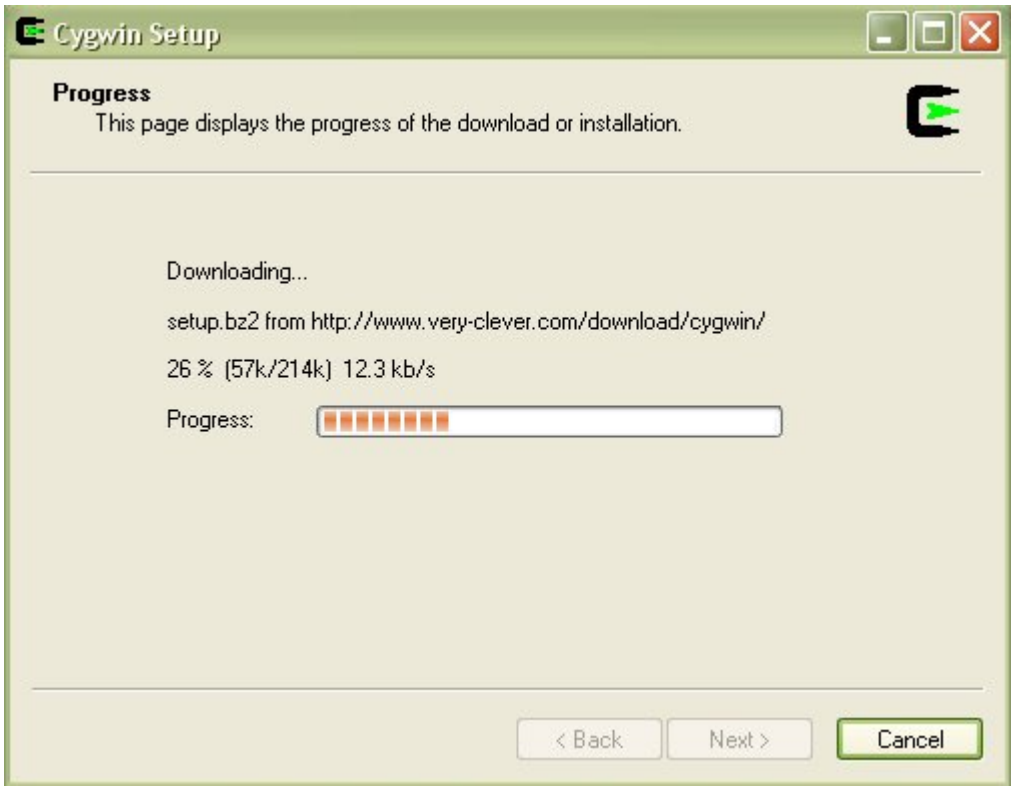


سپس پنجره بعدی باز می شود. در این پنجره یکی از لینکها را انتخاب می کنیم و دکمه next را فشار می دهیم.



در این مرحله شکل زیر ظاهر می شود و لیست بسته های در دسترس برای نصب cygwin دانلود خواهد شد.


Cygwin Setup
Progress
This page displays the progress of the download or installation.
Downloading...
setup.bz2 from http://cygwin.mirrors.hoobly.com/
Connecting...
Progress:
< Back
Next >
Cancel


Cygwin Setup
Progress
This page displays the progress of the download or installation.
Downloading...
setup.bz2 from http://www.very-clever.com/download/cygwin/
26 % (57k/214k) 12.3 kb/s
Progress:
< Back
Next >
Cancel

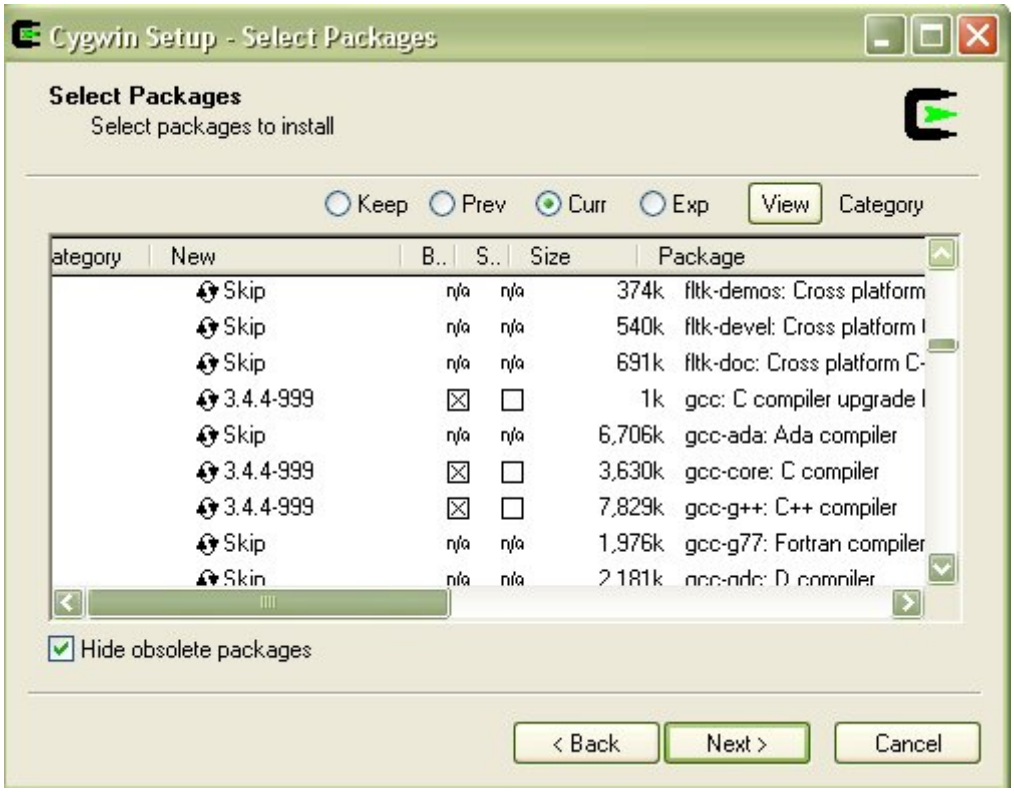
در ادامه پنجره ای برای انتخاب بسته هایی که باید نصب شوند آورده شده تا برای دانلود انتخاب شوند..

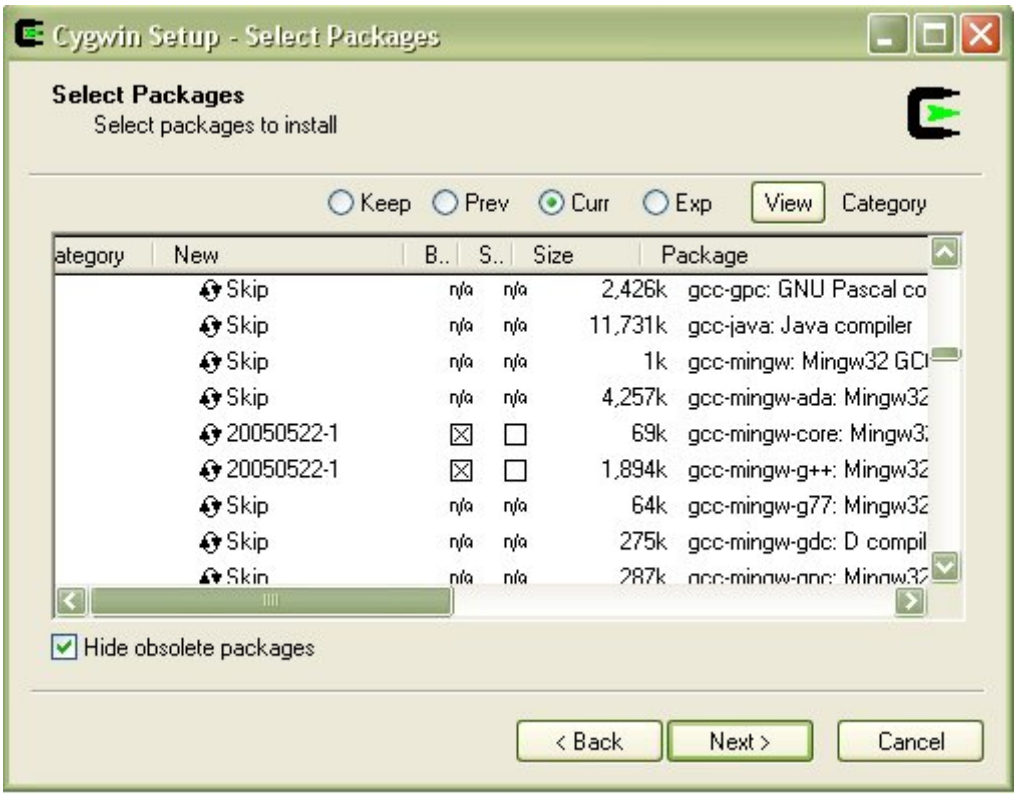
در این پنجره دسته devel را باز و در ان گزینه gcc را پیدا می کنیم.

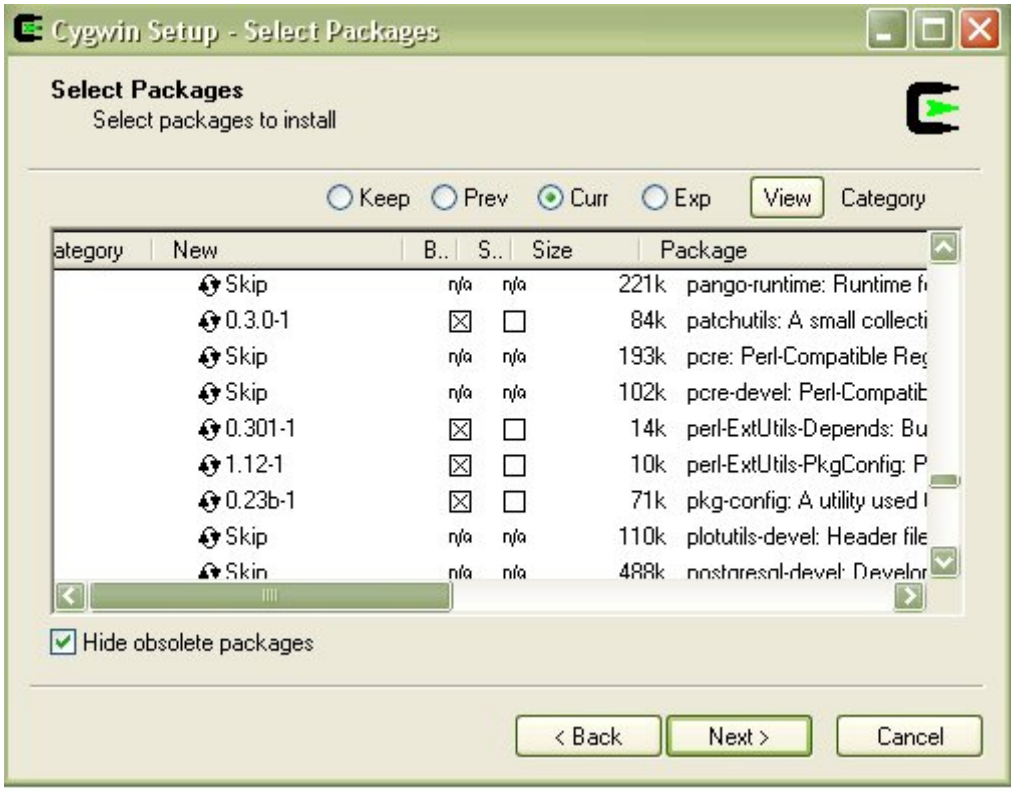
روی گزینه های gcc,gcc-g++,patchutils,gcc-core وگزینه های مربوط به perl کلیک می کنیم تا برای نصب انتخاب گردند. سپس دسته net را باز کرده و گزینه inetutils را برای نصب انتخاب می کنیم.

در ادامه همین دسته گزینه های openssl وopenssh را برای نصب انتخاب می کنیم و در ادامه گزینه make را نیز برای نصب انتخاب نموده و سپس در دسته X11 گزینهXorg-x11 وxorg-x11-devel را انتخاب می کنیم. در دسته editor گزینه vim را انتخاب می کنیم و.سپس دکمه next را می زنیم.

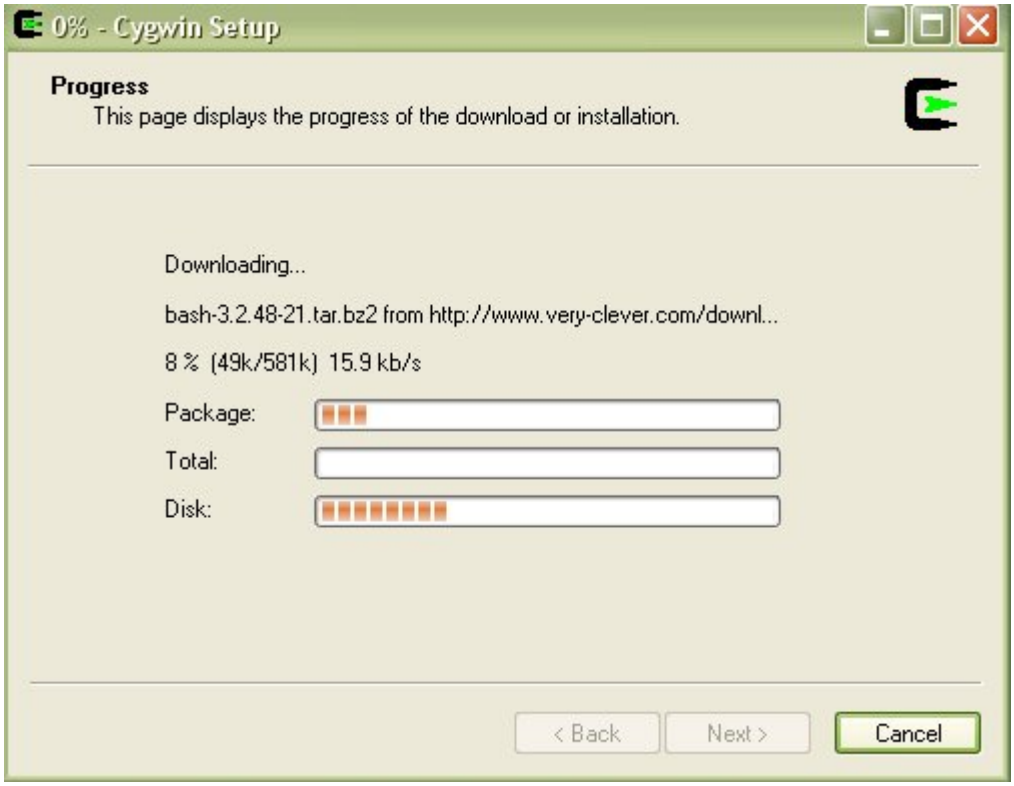
مراحل در شکل های زیر نشان داده شده اند.


Cygwin Setup - Select Packages
Select Packages
Select packages to install
Keep
Prev
Curr
Exp
View
Category
ategory
New
B..
S..
Size
Package
Skip n/a n/a 374k fltk-demos: Cross platform
Skip n/a n/a 540k fltk-devel: Cross platform
Skip n/a n/a 691k fltk-doc: Cross platform C-
3.4.4-999 1k gcc: C compiler upgrade
Skip n/a n/a 6,706k gcc-ada: Ada compiler
3.4.4-999 3,630k gcc-core: C compiler
3.4.4-999 7,829k gcc-g++: C++ compiler
Skip n/a n/a 1,976k gcc-g77: Fortran compiler
Hide obsolete packages
< Back
Next >
Cancel

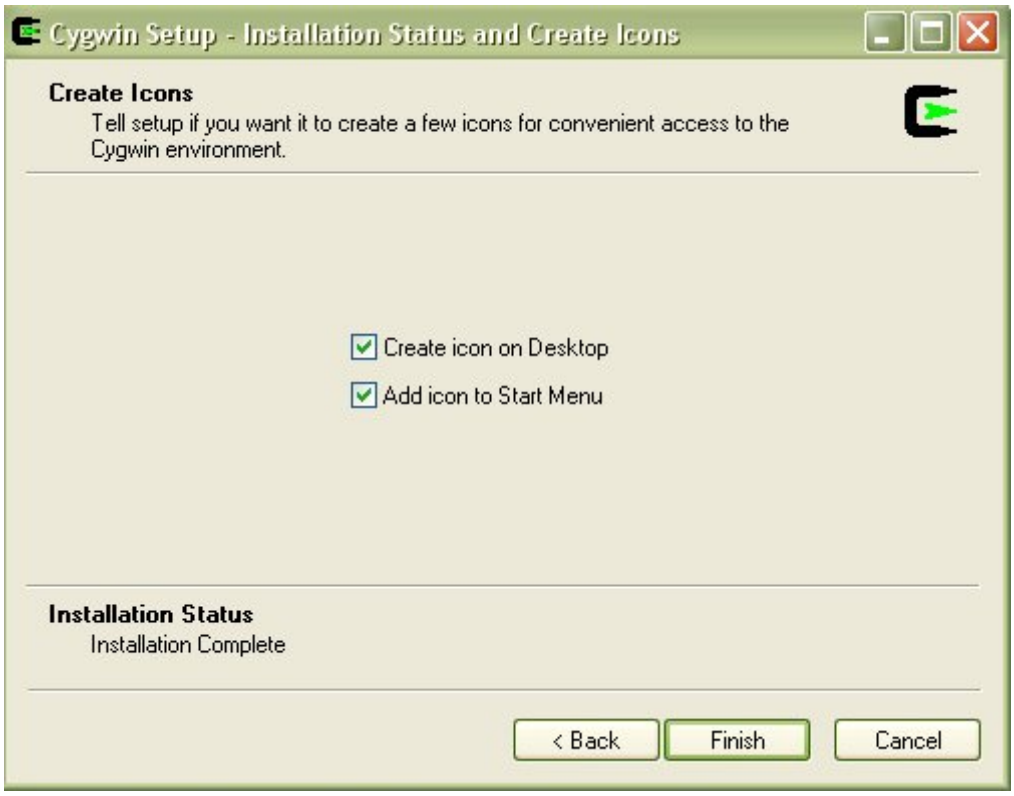

Cygwin Setup - Select Packages
Select Packages
Select packages to install
Keep
Prev
Curr
Exp
View
Category
ategory
New
B..
S..
Size
Package
Skip n/a n/a 2,426k gcc-gpc: GNU Pascal co
Skip n/a n/a 11,731k gcc-java: Java compiler
Skip n/a n/a 1k gcc-mingw: Mingw32 GC
Skip n/a n/a 4,257k gcc-mingw-ada: Mingw32
20050522-1 69k gcc-mingw-core: Mingw3
20050522-1 1,894k gcc-mingw-g++: Mingw32
Skip n/a n/a 64k gcc-mingw-g77: Mingw32
Skip n/a n/a 275k gcc-mingw-gdc: D compil
Hide obsolete packages
< Back
Next >
Cancel



در ادامه پنجره ای نمایان می شود. در این پنجره بسته هایی برای نصب پیشنهاد می شوندکه برای نصب کامل بسته هایی که در مراحل قبل انتخاب کرده ایم مورد نیاز است.بنابراین گزینه پایین پنجره را در حالت اتخاب شده رها کرده ودکمه next را میزنیم. در این مرحله برنامه set up شروع به دانلود کردن بسته های مورد نیاز برای نصب می شود.



این دانلود برای اینترنتی ۱۲۸ کیلو bps حدود ۴۵ دقیقه زمان خواهد برد. بعد از اینکه دانلود فایل ها به پایان رسید با اجرای برنامه setup ، نصب cygwin شروع می شود. پس از نصب برنامه setup از شما می خواهد در صورت تمایل اجازه دهید که در محیط desktop ودر start menu ، shortcut هایی برای دسترسی سریع به bash shell ایجاد کند.



دکمه finish را اتخاب میکنیم.در این مرحله نصب sygwin به پایان رسیده است.

برای اجرای برنامه cygwin روی آیکون

موجود در desktop کلیک می کنیم.

این پنجره محیط shellCygwin می باشد.

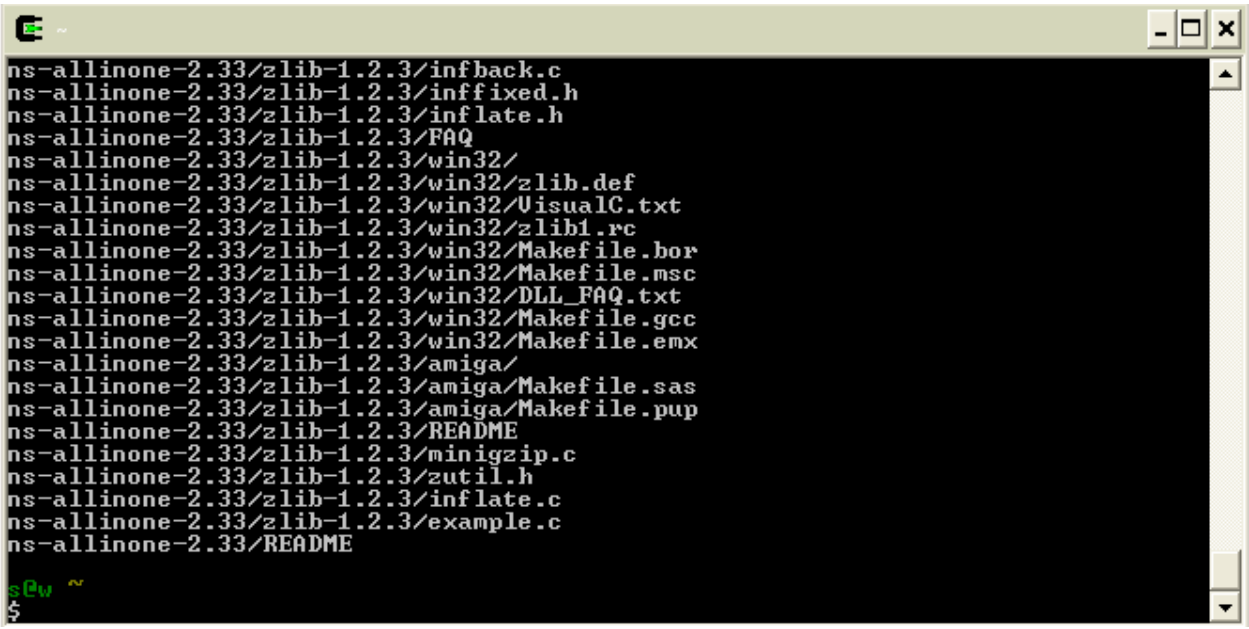
این پنجره محیطshell linux را شبیه سازی می کند. فایل های ذخیره شده و فایل هایی که قرار است ذخیره شوند همگی درون پوشه ای قرار می گیرند که هنگام نصب انتخاب کرده ایم. اگر پیش فرض نصب را تغییر نداده باشیم این فایل ها و پوشه ها درون پوشه c:\cygwin ذخیره می شوند. برای نصب ns در محیط cygwin مراحل زیر را انجام می دهیم. در ابتدا فایل نصبی ns را درون پوشه c:\cygwin\home\s ذخیره می کنیم.

سپس به ترتیب دستورات زیر را اجرا می کنیم.

pwd
cd
Ls
Tar xzvf ns-allinone-2.33.tar.gz

آخرین فرمان به نام tar فایل فشرده شده ns را باز میکند.

sew ~
$


ns-allinone-2.33/zlib-1.2.3/infback.c
ns-allinone-2.33/zlib-1.2.3/inffixed.h
ns-allinone-2.33/zlib-1.2.3/inflate.h
ns-allinone-2.33/zlib-1.2.3/FAQ
ns-allinone-2.33/zlib-1.2.3/win32/
ns-allinone-2.33/zlib-1.2.3/win32/zlib.def
ns-allinone-2.33/zlib-1.2.3/win32/VisualC.txt
ns-allinone-2.33/zlib-1.2.3/win32/zlib1.rc
ns-allinone-2.33/zlib-1.2.3/win32/Makefile.bor
ns-allinone-2.33/zlib-1.2.3/win32/Makefile.msc
ns-allinone-2.33/zlib-1.2.3/win32/DLL_FAQ.txt
ns-allinone-2.33/zlib-1.2.3/win32/Makefile.gcc
ns-allinone-2.33/zlib-1.2.3/win32/Makefile.emx
ns-allinone-2.33/zlib-1.2.3/amiga/
ns-allinone-2.33/zlib-1.2.3/amiga/Makefile.sas
ns-allinone-2.33/zlib-1.2.3/amiga/Makefile.pup
ns-allinone-2.33/zlib-1.2.3/README
ns-allinone-2.33/zlib-1.2.3/minigzip.c
ns-allinone-2.33/zlib-1.2.3/zutil.h
ns-allinone-2.33/zlib-1.2.3/inflate.c
ns-allinone-2.33/zlib-1.2.3/example.c
ns-allinone-2.33/README
sew ~
$

~/ns-allinone-2.33

```
ns-allinone-2.33/zlib-1.2.3/FAQ
ns-allinone-2.33/zlib-1.2.3/win32/
ns-allinone-2.33/zlib-1.2.3/win32/zlib.def
ns-allinone-2.33/zlib-1.2.3/win32/VisualC.txt
ns-allinone-2.33/zlib-1.2.3/win32/zlib1.rc
ns-allinone-2.33/zlib-1.2.3/win32/Makefile.bor
ns-allinone-2.33/zlib-1.2.3/win32/Makefile.msc
ns-allinone-2.33/zlib-1.2.3/win32/DLL_FAQ.txt
ns-allinone-2.33/zlib-1.2.3/win32/Makefile.gcc
ns-allinone-2.33/zlib-1.2.3/win32/Makefile.emx
ns-allinone-2.33/zlib-1.2.3/amiga/
ns-allinone-2.33/zlib-1.2.3/amiga/Makefile.sas
ns-allinone-2.33/zlib-1.2.3/amiga/Makefile.pup
ns-allinone-2.33/zlib-1.2.3/README
ns-allinone-2.33/zlib-1.2.3/minigzip.c
ns-allinone-2.33/zlib-1.2.3/zutil.h
ns-allinone-2.33/zlib-1.2.3/inflate.c
ns-allinone-2.33/zlib-1.2.3/example.c
ns-allinone-2.33/README

s@w ~
$ cd ns-allinone-2.33

s@w ~/ns-allinone-2.33
$
```

~/ns-allinone-2.33

```
rm -f test.gb sample.out test_io test_io.exe test_graph test_graph.exe test_flip
 test_flip.exe test_sample test_sample.exe
echo "Congratulations --- the tests have all been passed."
Congratulations --- the tests have all been passed.
touch certified
==========================================================
* Build GT-ITM
==========================================================
gcc  -I../include -L../lib  -DFBSD   -c -o itm.o itm.c
gcc  -I../include -L../lib  -DFBSD   -c -o geog.o geog.c
gcc  -I../include -L../lib  -DFBSD   -c -o ts.o ts.c
gcc  -I../include -L../lib  -DFBSD   -c -o dfs.o dfs.c
gcc  -I../include -L../lib  -DFBSD -o ../bin/itm itm.o geog.o ts.o dfs.o -lm -lg
b
gcc  -I../include -L../lib  -DFBSD   -c -o sgb2alt.o sgb2alt.c
gcc  -I../include -L../lib  -DFBSD -o ../bin/sgb2alt sgb2alt.o  -lm -lgb
gcc  -I../include -L../lib  -DFBSD   -c -o edriver.o edriver.c
gcc  -I../include -L../lib  -DFBSD   -c -o eval.o eval.c
gcc  -I../include -L../lib  -DFBSD -o ../bin/edriver edriver.o eval.o -lm -lgb
gt-itm has been installed successfully.
gcc  -I../include -L../lib    -c -o sgb2comns.o sgb2comns.c
gcc  -I../include -L../lib  -o ../bin/sgb2comns sgb2comns.o -lm -lgb
gcc  -I../include -L../lib    -c -o sgb2hierns.o sgb2hierns.c
gcc  -I../include -L../lib  -o ../bin/sgb2hierns sgb2hierns.o -lm -lgb
```

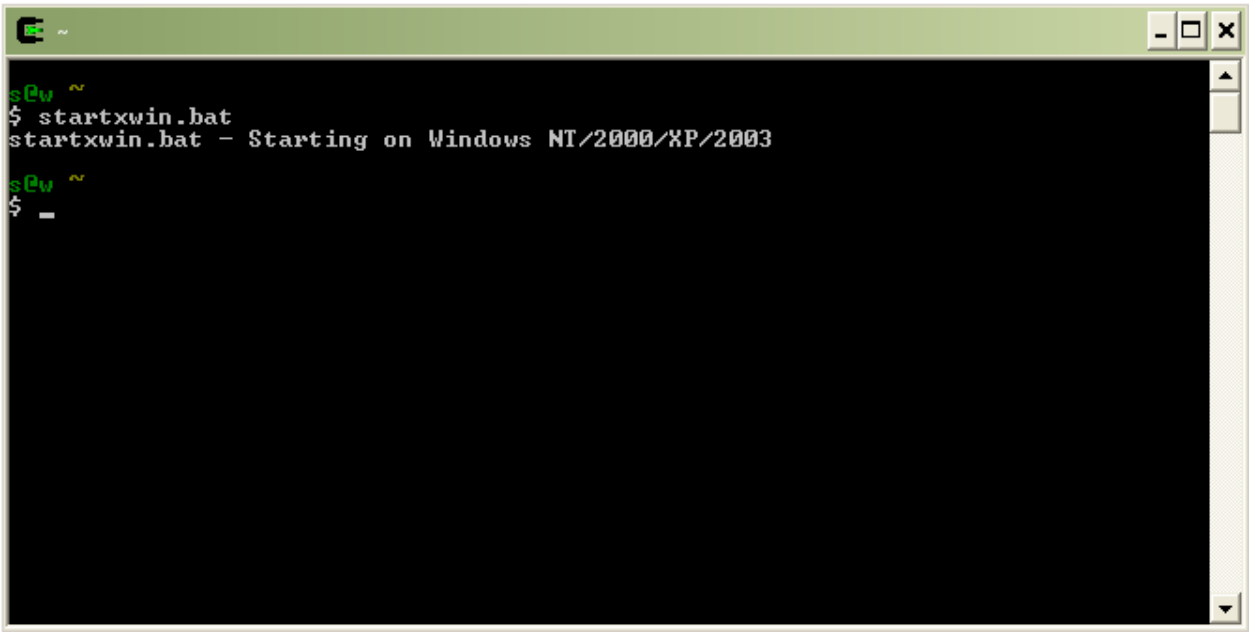
```
~/ns-allinone-2.33
_TIME=1 -DHAVE_TZNAME=1 -DHAVE_GMTIME_R=1 -DHAVE_LOCALTIME_R=1 -DHAVE_TIMEZONE_V
AR=1 -DHAVE_ST_BLKSIZE=1 -DSTDC_HEADERS=1 -DNO_UNION_WAIT=1 -DHAVE_SIGNED_CHAR=1
 -DHAVE_LANGINFO=1 -DHAVE_FTS=1 -DHAVE_SYS_IOCTL_H=1        -DTCL_SHLIB_EXT=\"\"
   /home/s/ns-allinone-2.33/tcl8.4.18/unix/../generic/tclUtf.c
gcc -c -O  -pipe  -DTCL_DBGX= -Wall -Wno-implicit-int -fno-strict-aliasing  -I.
-I/home/s/ns-allinone-2.33/tcl8.4.18/unix/../generic -I/home/s/ns-allinone-2.33/
tcl8.4.18/unix -DNO_VALUES_H=1 -DHAVE_LIMITS_H=1 -DHAVE_UNISTD_H=1 -DHAVE_SYS_PA
RAM_H=1 -DSTATIC_BUILD=1 -DTCL_WIDE_INT_TYPE=long\ long -DHAVE_GETCWD=1 -DHAVE_O
PENDIR=1 -DHAVE_STRSTR=1 -DHAVE_STRTOL=1 -DHAVE_STRTOLL=1 -DHAVE_STRTOULL=1 -DHA
VE_TMPNAM=1 -DHAVE_WAITPID=1 -DUSE_TERMIOS=1 -DHAVE_SYS_TIME_H=1 -DTIME_WITH_SYS
_TIME=1 -DHAVE_TZNAME=1 -DHAVE_GMTIME_R=1 -DHAVE_LOCALTIME_R=1 -DHAVE_TIMEZONE_V
AR=1 -DHAVE_ST_BLKSIZE=1 -DSTDC_HEADERS=1 -DNO_UNION_WAIT=1 -DHAVE_SIGNED_CHAR=1
 -DHAVE_LANGINFO=1 -DHAVE_FTS=1 -DHAVE_SYS_IOCTL_H=1        -DTCL_SHLIB_EXT=\"\"
   /home/s/ns-allinone-2.33/tcl8.4.18/unix/../generic/tclUtil.c
gcc -c -O  -pipe  -DTCL_DBGX= -Wall -Wno-implicit-int -fno-strict-aliasing  -I.
-I/home/s/ns-allinone-2.33/tcl8.4.18/unix/../generic -I/home/s/ns-allinone-2.33/
tcl8.4.18/unix -DNO_VALUES_H=1 -DHAVE_LIMITS_H=1 -DHAVE_UNISTD_H=1 -DHAVE_SYS_PA
RAM_H=1 -DSTATIC_BUILD=1 -DTCL_WIDE_INT_TYPE=long\ long -DHAVE_GETCWD=1 -DHAVE_O
PENDIR=1 -DHAVE_STRSTR=1 -DHAVE_STRTOL=1 -DHAVE_STRTOLL=1 -DHAVE_STRTOULL=1 -DHA
VE_TMPNAM=1 -DHAVE_WAITPID=1 -DUSE_TERMIOS=1 -DHAVE_SYS_TIME_H=1 -DTIME_WITH_SYS
_TIME=1 -DHAVE_TZNAME=1 -DHAVE_GMTIME_R=1 -DHAVE_LOCALTIME_R=1 -DHAVE_TIMEZONE_V
AR=1 -DHAVE_ST_BLKSIZE=1 -DSTDC_HEADERS=1 -DNO_UNION_WAIT=1 -DHAVE_SIGNED_CHAR=1
 -DHAVE_LANGINFO=1 -DHAVE_FTS=1 -DHAVE_SYS_IOCTL_H=1        -DTCL_SHLIB_EXT=\"\"
   /home/s/ns-allinone-2.33/tcl8.4.18/unix/../generic/tclVar.c
```

حال برای اجرا باید دستور install/. را فراخوانی کنیم. پس از اجرای این دستور برنامه ns در همین پوشه نصب می شود. پس از نصب دستوراتی داده می شود که باید آنها را اجرا کنیم.

حال دستورات مقابل را اجرا می کنیم تا فرمان های لازم برای تغییر مسیر اجرا شود.

```
LD_LIBRARY_PATH=/home/s/ns-allinone-2.33/otcl-1.12
LD_LIBRARY_PATH=/home/s/ns-allinone-2.33/lib
TCL_LIBRARY_PATH=/home/ns-allinone-2.33/tcl8.4.13/library
Export LD_LIBRARY_PATH
Export TCL_LIBRARY
```

توجه شود در صورتی که نیاز داریم نتایج شبیه سازی را در نرم افزار nam ببینیم باید دستور startwxwin.bat را اجرا کنیم (شکل زیر)



در این حالت یک پنجره دیگر به شکل زیر باز خواهد شد.

۱-۳- شروع کار با N.S.

برای شروع کار کافی است برنامه موردنظر را که از دستورات موجود در N.S. و همچنین از دستورات TCL برای بیان توپولوژی شبکه و سایر ویژگی‌های شبکه موردنظر تشکیل شده است را در یک ویرایشگر

مثل emacs وارد کرده و آن را با پسوند tcl ذخیره نمود. سپس، برای اجرا کافی است دستور زیر را وارد نمایید:

نام فایل برنامه با پسوند ns tcl

خروجی برنامه را می‌توان به برنامه N.A.M داد، تا نتایج را به صورت Visual نمایش دهد و ما بتوانیم مراحل مختلف شبیه‌سازی را در فواصل زمانی مختلف مشاهده کنیم.

۱-٤- شبیه‌سازی .N.S

ویژگی‌های یک شبکه که باید در برنامه شبیه‌سازی لحاظ شود، عبارتند از:

- تعداد نودها
- مشخصه هر نود
- مشخصه لینک‌های اتصال‌دهنده نودها
- نوع پروتکل‌های متصل به نودهای مبداء و مقصد
- انواع ترافیک ارسالی
- زمان شبیه‌سازی و زمانهای فعال‌شدن هر منبع ترافیکی
- مشخصات فایل‌های مربوط به پایش ترافیک شبکه و استخراج پارامترهای کارآیی شبکه.

در اولین قدم در انجام شبیه‌سازی، باید یک شیء (object) از کلاس شبیه‌ساز ایجاد گردد. این امر با کمک دستور زیر قابل حصول است:

```
Set ns[new simulator]
```

بعد از دستور فوق با کمک دستورات ns و otcl توپولوژی شبکه که شامل نودها، لینک‌ها و عاملان شبکه (Agent) می‌باشند، ایجاد می‌شود. بعد از ایجاد نودها و عاملان شبکه، با کمک لینک‌ها نودهای شبکه به هم متصل می‌شوند و همچنین عاملان شبکه نیز به نودهای مربوطه اتصال می‌یابند. به عنوان مثال، پروتکل‌های مسیریابی دینامیکی، منابع ترافیکی و بسیاری از پروتکل‌های لایه ارسال نمونه‌ای از عاملان شبکه می‌باشند. بعد از ایجاد عاملان شبکه و اتصال آنها به نودها، می‌توان پارامترهای آنها را نیز تنظیم نمود. به عنوان مثال دو دستور زیر یک عامل ارسال از نوع TCL تعریف می‌نمایند و سپس پارامتر طول پنجره ارسال (window) که یک متغیر در پروتکل TCP است را به ۲۵ مقداردهی می‌کنند

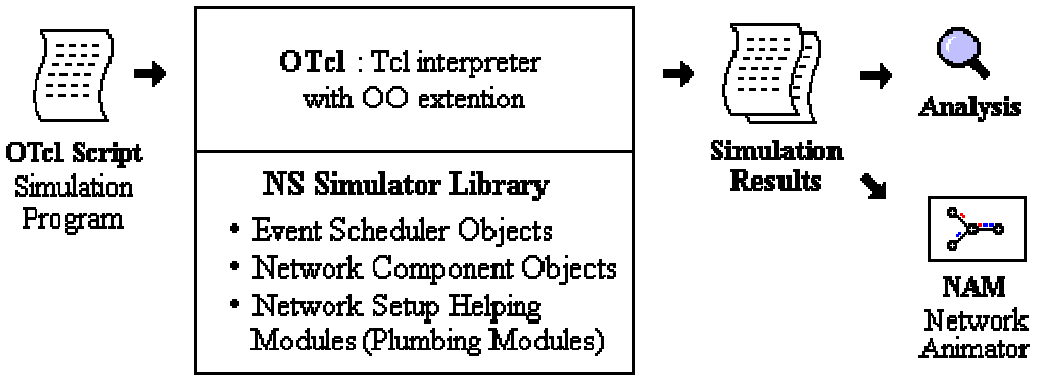
```
Set tcp[new Agent/TCP]
$ tcp set window_25
```

برای زمانبندی رخدادها در .N.S 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 موقت در لینک‌های شبکه در زمانهای خاص، پیکربندی مجدد توپولوژی شبکه و نظیر اینها فراهم می‌آید. به عنوان مثال دستورات زیر باعث می‌شوند که منبع ترافیکی از قبل تعیین شده $cbr0 در زمان 0.5 شروع به ارسال ترافیک نماید و در زمان ۴/۵ متوقف گردد.

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

با کمک دستور run شبیه‌سازی آغاز می‌شود و همچنین دستورات stop و exit باعث خاتمه عملیات شبیه‌سازی می‌شوند. آنچه در بالا آورده شد، توصیف اجمالی نحوه انجام عملیات شبیه‌سازی در .N.S است. در بخش‌های بعدی به توصیف کامل‌تر دستورات .N.S و TCL برای انجام شبیه‌سازیهای مختلف می‌پردازیم. هدف این گزارش آشنا ساختن خواننده با مفاهیم اولیه در ارتباط با شبیه‌سازی شبکه‌ها، چگونگی تنظیم پارامترهای موجود و چگونگی ایجاد اجزاء یک شبکه و غیره می‌باشد. در اینجا سعی می‌شود تا با عنوان مثالهایی نسبتاً ساده و از طریق توضیح آنها اطلاعات ضروری به خواننده داده شود. در ادامه خواننده قادر خواهد بود تا در صورت نیاز، با مراجعه به مراجع پیشرفته‌تر که آدرس آنها در انتهای گزارش آورده شده است، این راه را ادامه دهد.

شکل ۱ نمای کلی از اجزاء NS و طریقهٔ کنار هم قرارگرفتن آنها را نشان می‌دهد. برای کار با NS شما باید یک برنامه به زبان OTCL بنویسید، سپس برای اجرای عملیات شبیه‌سازی باید عمل زمان‌بندی، ایجاد توپولوژی مناسب بین اجزای شبکه، تعریف نوع ترافیک موجود انجام پذیرد. در تمام مراحل فوق می‌توان از توابع کتابخانه‌ای موجود استفاده کرد.



شکل ۱: : اجزای کلی ns

۱-۴-۱- OTCL زبان مورد استفاده کاربر

همانطور که قبلاً ذکر شد، نرم‌افزار NS بر اساس مفسر OTCL است که با کتابخانه‌های شبیه‌سازی شبکه کار می‌کند. دانستن اینکه در OTCL چگونه برنامه‌نویسی کنید برای کار با NS بسیار مفید خواهد بود. در این قسمت مثالی از یک برنامه TCL و OTCL به منظور آشنایی با مفاهیم مقدماتی برنامه‌نویسی در OTCL آورده شده است. این مثالها از پنجمین خودآموز NS آورده شده‌اند. در این قسمت فرض شده است که خواننده NS را نصب کرده است و با زبان‌های برنامه‌نویسی C و C++ آشنایی دارد.

مثال ۱، یک مثال کلی است تا نشان دهد که چگونه می‌توان در TCL یک پروسیجر تولید کرد و چگونه آنرا فرا خواند، همچنین چگونه می‌توان به متغیرها مقدار داد و یا یک حلقه درست کرد. از آنجائیکه OTCL نسخه شیءگرا (Object- Oriented) برنامه TCL است. بنابراین واضح است که تمام دستورات C و C++ است. برای شروع می‌توانید برنامهٔ زیر را که به زبان TCL نوشته شده است در یک فایل بنویسید و آنرا با اسم ex-tcl.tcl ذخیره کنید. برای اجرای این برنامه در صفحهٔ اجرایی عبارت مقابل را تایپ کنید ns ex-tcl.tcl. دستور ns کار تفسیر OTCL را به وسیلهٔ NS آغاز می‌کند. اگر نسخهٔ tcl8.0 یا بالاتر را در اختیار دارید می‌توانید همین کار را با تایپ عبارت ex-tcl.tcl انجام دهید.

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

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

مثال ۱: برنامهٔ TCL ساده

در TCL، کلمهٔ کلیدی proc برای تعریف یک پروسیجر بکار می‌رود. بعد از این کلمهٔ کلیدی اسم پروسیجر و سپس آرگومانهای آن در براکت می‌آیند. برای مقداردهی به یک متغیر از کلمهٔ کلیدی set استفاده می شود. عبارت [expr ...] به مفسر می‌گوید که مقدار عبارت داخل براکت که بعد از کلمهٔ exper آمده است را محاسبه نماید. باید به این نکته توجه داشته باشید که برای در اختیار داشتن مقدار اختصاص یافته به یک متغیر از علامت $ که به همراه اسم متغیر می‌آید استفاده کنید. برای چاپ مقادیر موردنظر

می‌توانید از کلمهٔ کلیدی puts استفاده نمایید و مقادیر موردنظر را داخل " " قرار دهید. با اجرای این برنامه مقادیر زیر را در خروجی ملاحظه خواهید کرد.

K<5, pow=1.0
K<5, pow=1120.0
K<5, pow=1254400.0
K<5, pow=1404928000.0
K<5, pow=1573519360000.0
K<5, mod=0
K<5, mod=4
K<5, mod=0
K<5, mod=0
K<5, mod=4

مثال بعدی یک برنامهٔ شیءگرا است که این بار در OTCL نوشته شده است. این مثال ساده چگونگی ایجاد یک شیء و استفاده از آن را در OTCL نشان می‌دهد. اگر به عنوان یک کاربر معمولی از NS استفاده می‌کنید، استفاده از یک شیء در برنامه ممکن است به ندرت پیش بیاید اما دانستن یک شیء در NS می‌تواند مفید باشد. شیءهای NS که در برنامه‌های شبیه‌سازی مورد استفاده قرار می‌گیرند ممکن است به زبان C++ باشند اما بخاطر پیوندهای موجود بین OTCL و C++ در OTCL براحتی قابل استفاده هستند.

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

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

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

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

مثال ۲: برنامهٔ OTCL ساده

مثال ۲ یک برنامهٔ OTCL است که دو کلاس شیء به اسم‌های mom و kid تعریف می‌کند که کلاس kid یک کلاس فرزند (child) از mom است و هر دو کلاس از یک تابع به نام greet استفاده می‌کنند. بعد از تعریف هر شیء، متغیر age برای کلاس mom به ۴۵ و برای کلاس kid به ۱۵ تنظیم می‌شود و سپس تابع greet برای هر کدام از شیءها صدا زده می‌شود. کلمهٔ کلیدی class برای تولید یک کلاس شیء و instproc برای تعریف تابع هر کلاس شیء مورد استفاده قرار می‌گیرند. کلمهٔ کلیدی superclass برای تعریف خاصیت ارث‌بری بکار می‌رود و همانطور که در مثال قبل ملاحظه می‌کنید برای تعیین اینکه کلاس kid از نوع کلاس فرزند برای mom باشد از این کلمه استفاده شده است. در زبان C++ در تعریف توابع عضو در یک کلاس از اشاره‌گر this استفاده می‌شود و در اینجا $self همان نقش را در OTCL ایفا می‌کند. Instvar چک می‌کند که آیا اسم متغیری که در ادامه می‌آید در کلاس مورد نظر تعریف شده است یا خیر. اگر این متغیر تعریف شده باشد، مورد استفاده قرار می‌گیرد در غیر اینصورت یک متغیر جدید تغریف خواهد شد. برای تولید شیء، همانطور که در مثال ۲ مشاهده می‌کنید، از کلمهٔ new استفاده می‌شود. اگر مثال ۲ را در برنامه‌ای با نام ex-otcl.tcl ذخیره کرده باشید با تایپ ns ex-otcl.tcl مقادیر زیر را مشاهده خواهید کرد.

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

۱-۴-۲- یک مثال ساده از شبیه‌سازی

در این قسمت یک شبیه‌سازی در NS بیان می‌گردد و توضیح داده خواهد شد که هر خط چه کاری انجام می‌دهد. مثال ۳ یک برنامه OTCL است که شبکه ساده شکل ۲را ایجاد می‌کند. برای اجرای شبیه‌سازی، برنامهٔ زیر را در یک فایل به اسم ns-simple.tcl ذخیره کنید و عبارت ns ns-simple.tcl را در محیط اجرایی تایپ کنید. این شبکه از چهار گره به اسم‌های n3, n2, n1, n0 تشکیل شده است. یک لینک از نوع دو طرفه (duplex link) بین n0 و n2 و همچنین بین n1 و n3 با پهنای باند 2Mbps و تأخیر زمانی 10ms ایجاد می‌شود. بین n2 و n3 یک لینک دوطرفه با پهنای باند 1.7Mbps و تأخیر زمانی 20 ms قرار دارد. هر یک از گره‌ها از صف Drop Tail با حداکثر اندازه 10 استفاده می‌کنند.

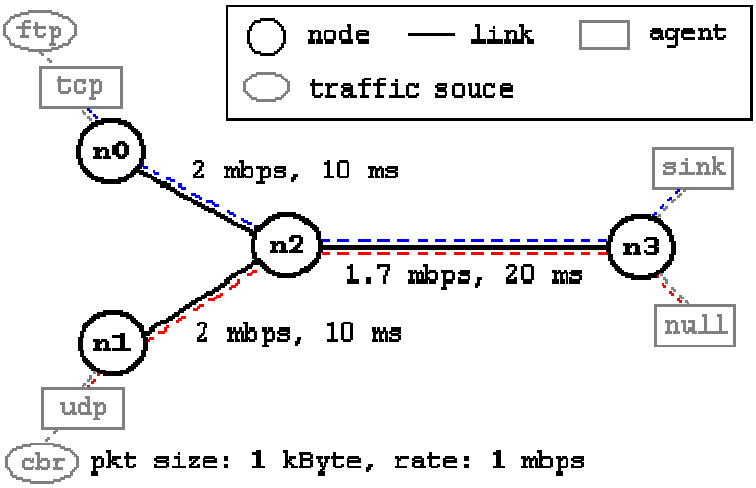
یک عامل (agent) از نوع tcp به گروه n0 متصل می‌شود، همچنین یک اتصال برای عامل Sink متصل شده به n3 ایجاد می‌گردد. بر طبق پیش فرض حداکثر اندازه یک بسته که عامل tcp می‌تواند تولید کند یک کیلوبایت است. عامل sink وظیفهٔ تولید و ارسال بسته‌های ACK برای فرستنده یعنی عامل tcp و نیز آزاد کردن بسته‌های دریافت شده را برعهده دارد. عامل udp متصل شده به n1 به عامل null متصل شده به n3 وصل است. عامل null فقط بسته‌های دریافت شده را آزاد می‌کند. ftp و cbr (Constant bit rate) تولید کننده‌های ترافیک هستند که به ترتیب به عوامل tcp و udp متصل می‌باشند.

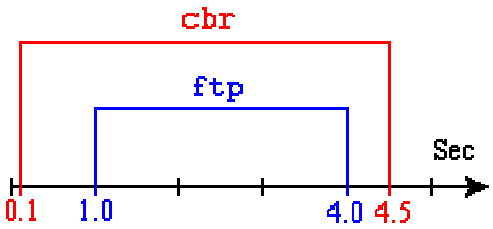
Cbr بسته‌های یک کیلوبایتی را با نرخ 1Mbps تولید می‌کند و طوری تنظیم شده است که در 0.1 ثانیه شروع بکار کرده و در ۴/۵ ثانیه متوقف می‌شود. ftp هم در ۱ ثانیه شروع و در ۴ ثانیه به کار خود پایان می‌دهد.

```
Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#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 NAM trace file
     close $nf
     #Execute NAM on the trace file
     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]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

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

#Give node position (for NAM(
$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

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


#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
```

```
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP


#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 packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false


#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

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

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
   $ns run
```

مثال ۳: یک برنامه شبیه‌سازی ساده در NS





شکل ۲: توپولوژی یک شبکه ساده

بطور کلی یک برنامه NS با تولید یک شیء شبیه‌ساز (Simulator Object) شروع بکار می‌کند.

- Set ns [new simulator]

باعث ایجاد یک شیء شبیه‌ساز می‌شود و آنرا به متغیر ns تخصیص می‌دهد. در اینجا برای متمایزشدن متغیرها و مقادیر از بقیه، آنها بصورت ایتالیک نمایش داده شده‌اند. این خط دستور کارهای زیر را انجام می‌دهد:

- فرمت بسته را مقداردهی اولیه می‌کند.
- یک زمان‌بند (Scheduler) ایجاد می‌کند بصورت پیش فرض این زمان‌بند یک تقویم است.
- فرمت آدرس پیش فرض را انتخاب می‌کند (فعلاً این قسمت را نادیده بگیرید).

شیئی شبیه‌ساز دارای توابعی است که عملیات زیر را انجام می‌دهند.

- شیئی‌های ترکیبی مانند لینک‌ها و گره‌ها را تولید می‌کنند.
- اجزاء شی‌های تولید شده نظیر عامل‌های tcp یا udp را وصل می‌کند.
- پارامترهای اجزای شبکه که معمولاً برای شیءهای ترکیبی مورد استفاده قرار می‌گیرند را تنظیم می‌کند.
- بین عاملها مانند tcp و sink پیوند ایجاد می‌کند.
- نحوهٔ نمایش NAM را معین می‌کند.

- غیره

بیشتر توابع برای تنظیم عملیات شبیه‌سازی و زمان‌بندی هستند هر چند که برخی از آنها برای نمایش NAM بکار می‌روند و می‌توانید این توابع را در قسمت ns-2/tcl/lib/ns-lib.tcl پیدا کنید.

$ns color fid color:

برای تنظیم رنگ بسته‌های جریان مشخص شده بوسیلهٔ جریان id (fid) است. این تابع شیء شبیه‌ساز برای نمایش NAM بوده و بر روی شبیه‌ساز اصلی تأثیری ندارد. بعنوان مثال دستور $ns color 1 Blue باعث می‌شود که جریان داده اول با رنگ آبی مشخص شود.

$ns namtrace-all file-descriptor:

این تابع به شبیه‌ساز می‌گوید که ردیابی (trace) شبیه‌سازی را به شکل فرمت ورودی NAM ذخیره کند. همچنین اسم فایلی که نتایج ردیابی بعداً قرار است به وسیلهٔ دستور $ns flush-trace در آن نوشته شود را می‌گیرد. تابع trace-all هم بطریق مشابه برای ذخیره‌سازی نتایج ردیابی با فرمت عمومی مورد استفاده قرار می‌گیرد.

Proc finish{ }:

بعد از شبیه‌سازی به وسیلهٔ دستور $ns at s.o "*finish*" پروسیجر finish بعد از گذشت پنج ثانیه از شروع شبیه‌سازی فراخوانده می‌شود. در این تابع پروسه‌های بعد از شبیه‌سازی (post-simulation) مورد توجه قرار دارند.

Set n0 [$ns node]:

این تابع گره‌ای را ایجاد می‌کند. در NS یک گره، یک شیء ترکیبی است که از آدرس و طبقه‌بندی کننده پورت (port classifier) که در بخش بعدی توضیح داده می‌شود، تشکیل شده است. کاربرها می‌توانند یک گره را از طریق ایجاد آدرس یا شیء طبقه‌بندی کننده پورت که به طور جداگانه ایجاد شده است و در ادامه به هم متصل می‌شوند، ایجاد نمایند. اما این تابع کار ایجاد یک گره را آسان کرده است. برای اینکه ببینید یک گره چگونه ساخته می‌شود می‌توانید به فایلهای ns-2/tcl/libs/ns-lib.tcl و ns-2/tcl/libs/ns-node.tcl مراجعه نمایید.

$ns duplex-link node I node 2 bandwith delay queue-type:

این دستور دو لینک ساده با تأخیر و پهنای باند معین ایجاد کرده و دو گره مشخص شده را به هم متصل می‌کند. در NS صف خروجی گره بعنوان بخشی از لینک محسوب می‌شود، بنابراین کاربرها باید نوع صف را در هنگام ایجاد لینک‌ها مشخص کنند. مثلاً در مثال فوق صف از نوع Drop Tail تعریف شده است و اگر کاربر بخواهد از صف RED استفاده کند براحتی می‌تواند بجای کلمهٔ Drop Tail کلمهٔ RED را قرار دهد.

لینک هم مانند گره، یک شیء مرکب است و کاربرها می‌توانند اجزاء آن را بهمراه گره‌ها به هم متصل کرده و آن را تولید کنند. برای مشاهده کدهای موجود می‌توانید به فایلهای ns- ns2/tcl/libs/ns-lib.tcl یا ns-2/tcl/libs/ns-link.tcl مراجعه کنید. یک نکته قابل توجه این است که می‌توانید منابع خطا را در اجزاء

یک لینک وارد کنید و یک لینک با تلفات داشته باشید، برای آشنایی می‌توانید به راهنمای NS مراجعه کنید.

$ns queue-limit node 1 node 2 number

این دستور طول صف مربوط به دو لینک ساده، که گره‌های ۱ و ۲ را به هم متصل می‌کند، را با عدد داده شده مشخص می‌نماید. بعنوان مثال خط دستور $queue-limit $n2 $n3 10 اندازهٔ صف بین گره‌های n2 و n3 را برابر ۱۰ قرار می‌دهد.

$ns duplex-link-op node 1 node 2 …

این دستور برای نمایش در NAM مورد استفاده قرار می‌گیرد. به وسیله این دستور می‌توانید موقعیت گره‌ها را نسبت به یکدیگر تعیین کنید. مثلاً خط فرمان:

$ns duplex-link-op $n0 $n2 orient right-down

تعیین می‌کند که گره n2 نسبت به گره n0 در موقعیت راست-پایین (جنوب شرقی) قرار بگیرد. با حذف خط دستور موقعیت گره‌ها و اجرای دوباره شبیه‌سازی می‌توانید تأثیر آنها را بهتر مشاهده نمایید. تا این جا مراحل اساسی شبیه‌سازی انجام شده است. در قسمت بعد باید عوامل ترافیک مانند TCP و UDP و منابع ترافیک مثل FTP و CBR که به ترتیب به گره‌ها و عوامل متصل می‌شوند، تنظیم شوند. بعد از شکل‌بندی شبکه، زمان‌بندی و شبیه‌سازی نهایی انجام می‌شوند و تنها چیزی که باقی مانده است اجرای شبیه‌سازی می‌باشد که به وسیلهٔ دستور $ns run انجام می‌شود.

Set tcp [new Agent/TCP]:

این خط چگونگی تولید یک عامل TCP را نشان می‌دهد. اما بطور کلی، کاربر می‌تواند هر عامل یا منبع ترافیک را بصورت زیر ایجاد کند. عوامل و منابع ترافیک در حقیقت اشیاء پایه‌ای و نه مرکب هستند که معمولاً در C++ ایجاد می‌شوند و سپس در OTCL مورد استفاده قرار می‌گیرند. بنابراین، هیچ تابع بخصوصی وجود ندارد که آنها را تولید کند. برای ایجاد عوامل و منابع ترافیک، کاربر باید اسامی کلاسهای این اشیاء را بداند (Application/FTP, Agent/TCPSink, Agent/TCP و غیره). این گونه اطلاعات را می‌توانید در NS manual پیدا کنید اما برای راحتی کار می‌توانید به فایل ns-2/tcl/libs/ns-defoult.tcl مراجعه کنید. فایل مذکور حاوی مقادیر پارامترهای پیش فرض است و بنابراین می‌تواند کمک کند که چه اشیائی در NS موجود هستند و پارامترهای آنها چه هستند.

$ns attach-agent node agent

این دستور، عامل ایجاد شده را به گرهٔ مورد نظر وصل می‌کند. مثلاً $ns attach-agent $n1 $udp عامل udp را به گرهٔ n1 وصل می‌کند.

.$ns connect agent 1 agent 2

بعد از اینکه دو عامل مانند Null یا Udp ایجاد شدند، باید یک ارتباط منطقی بین آنها ایجاد شود. بعنوان مثال دستور $ns connect $Udp $Null دو عامل ذکر شده را به هم متصل می‌کند. فرض کنید که تمام مراحل مربوط به پیکربندی (Configuration) شبکه انجام شده است، قدم بعدی نوشتن سناریوی

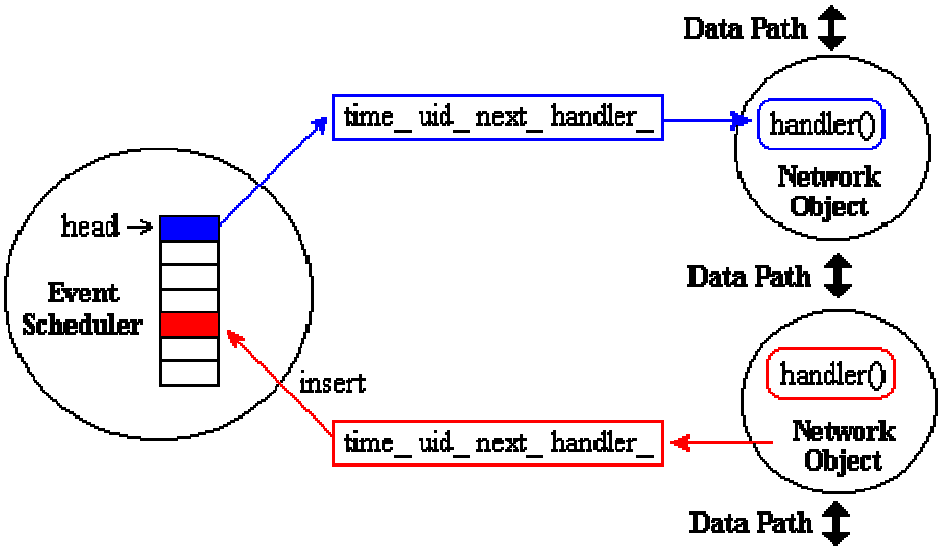
شبیه‌سازی یا بعبارت دیگر زمان‌بندی شبیه‌سازی است. شبیه‌سازی دارای توابع زمان‌بندی گوناگون است که مهمترین آنها در زیر آمده است.

.$ns at time “string”

این دستور باعث می‌شود که زمان‌بند (scheduler) اجرای عبارت مشخص شده در دستور را در زمان تعیین شده انجام دهد. بعنوان مثال دستور $ns at 0.1 “$cbr start” سبب خواهد شد که زمان‌بند منبع ترافیک CBR را فعال ساخته و داده‌ها ارسال شوند. در NS معمولاً منبع ترافیک، داده واقعی را ارسال نمی‌کند، بلکه به عامل، اطلاع می‌دهد که مقداری داده برای ارسال دارد و عامل فقط می‌داند که چه میزان اطلاعات باید ارسال شود و بنابراین بسته‌هایی را تولید و آنها را ارسال می‌کند.

۱-۴-۳- زمان‌بند رخداد (Event Scheduler)

این بخش به بررسی زمان‌بندهای رخداد گسسته در NS اختصاص دارد. همانطور که در قسمت مقدمه ذکر شد، استفاده کننده‌های اصلی از زمان‌بندها، اجزا شبکه هستند که زمان رسیدگی به بسته (Packet-handing) را شبیه‌سازی می‌کنند و یا بخشهایی می‌باشند که به تایمر نیاز دارند. شکل ۳ هر یک از شییءهای شبکه را که از زمان‌بند رخداد استفاده می‌کند نشان می‌دهد. باید توجه داشت که همان شیئی که یک رخداد را انتشار می‌دهد، رسیدگی به رخداد را بعداً در برنامه زمانی برعهده دارد. همچنین توجه کنید که مسیر داده‌ها بین اجزا شبکه با مسیر رخدادها متفاوت است. در حقیقت بسته‌ها از یک جزء شبکه به جزء دیگر به وسیلهٔ Send (packet *P){trage_->recv(P)} ارسال و به وسیلهٔ recv (packet*, Handler *h=0) دریافت می‌شوند.



شکل ۳: زمان‌بند رخداد گستر

NS از دو نوع مختلف زمان‌بند رخداد استفاده می‌کند. این دو نوع عبارتند از زمان‌بندهای زمان واقعی (real-time) و زمان غیرواقعی (non-real-time). برای یک زمان‌بند زمان غیرواقعی، سه اجرای Calender و Heap و List در دسترس هستند هرچند که تمام آنها تقریباً کار یکسانی را انجام می‌دهند. معمولاً Calender در زمان‌بند زمان غیرواقعی با همان مقادیر پیش فرض تنظیم می‌گردد. زمان‌بند زمان واقعی برای تقلید (emulation) بکار می‌رود تا به شبیه‌ساز این اجازه را بدهد که با شبکه واقعی ارتباط داشته باشد. در حال حاضر عمل تقلید در حال گسترش می‌باشد هرچند که نسخه آزمایشی آن هم اکنون در دسترس است. در زیر مثالی از انتخاب یک زمان‌بند رخداد خاص را مشاهده می‌کنید:

```
...
Set ns [new simulatr]
$ns use-scheduler Heap
…
```

یکی دیگر از کاربردهای زمان‌بند برای زمان‌بندی رخدادهای شبیه‌سازی است مثلاً چه موقع کاربرد FTP شروع بکار نماید یا چه زمانی کار شبیه‌سازی خاتمه یابد. زمان‌بند رخداد، خود دارای توابعی نظیر at time “string” می‌باشد که یک رخداد خاص که At Event نامیده می‌شود را در زمان معینی اجرا می‌کند. یک At Event در حقیقت یک کلاس فرزند از Event است که دارای یک متغییر اضافی برای نگهداری رشته داده شده به وسیله String می‌باشد. با این وجود مانند یک رخداد معمولی در زمان‌بند رخداد رفتار می‌کند.

هنگامی که عمل شبیه‌سازی آغاز می‌شود و زمان At Event در زمان‌بند فرا می‌رسد آنگاه At Event به یک At Event handler داده می‌شود که این رسیدگی کننده (handler) یک بار ایجاد می‌شود و برای تمام At Event ها مورد استفاده قرار می‌گیرد و دستور OTCL که به وسیله قسمت Sting مشخص شده است، اجرا می‌شود. در ادامه زمان‌بند رخداد شبیه‌سازی به مثال قبل اضافه شده است.

```
...
Set ns [new simulatr]
$ns use-scheduler Heap
$ns at 300.s “complete-sim”
…
Proc complete-sim { } {
…
}
```

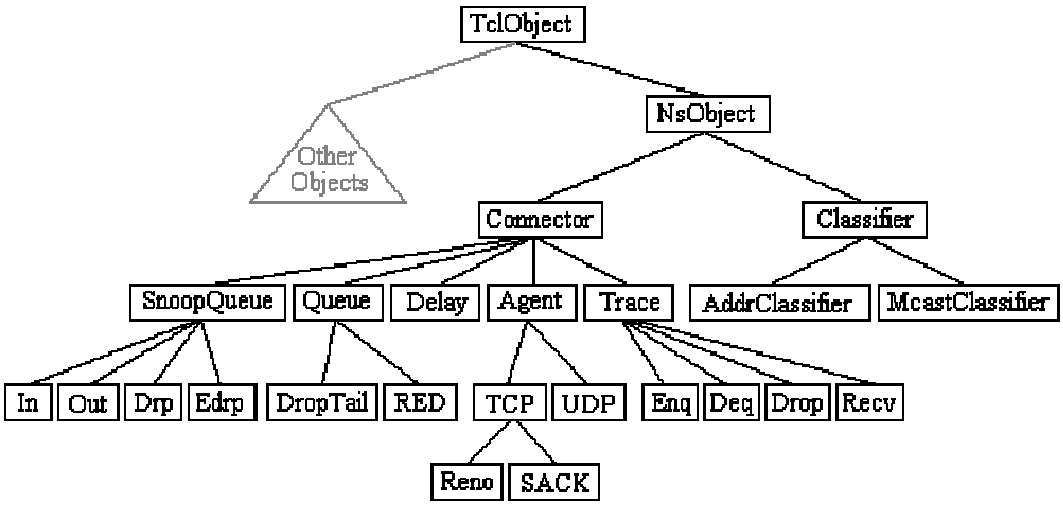
ممکن است به این موضوع توجه کرده باشید که at time “string” یک تابع عضو شیئی شبیه‌ساز set ns[new simulation] است. اما بخاطر داشته باشید که این شیئی شبیه‌ساز فقط بعنوان یک واسط (interface) کاربر عمل می‌کند و در واقع توابع عضو اشیاء شبکه یا شیئی زمان‌بند را صدا می‌زند تا کار واقعی را آنها انجام دهند. در ادامه لیستی از توابع عضو شیئی شبیه‌ساز که با توابع عضو زمان‌بند در ارتباط هستند به همراه توضیح مختصری دربارهٔ آنها آمده است.

Simulator instproc now	# return scheduler’s notion of current time
Simulator instproc at args	# scheduler execution of code specified

```
                                        time
Simulator instproc at-now args          # scheduler execution of code at now
Simulator instproc after n args         # scheduler execution of code after n secs
Simulator instproc run arge             # start scheduler
Simulator instproc halt                 # stop (pause) scheduler
```

۱-۴-۴- اجزای شبکه Network components

این بخش درباره اجزای NS که معمولاً اجزای ترکیبی شبکه هستند صبحت می‌کند. شکل ۴ قسمتی از سلسله مراتب کلاسهای NS را نشان می‌دهد که می‌تواند برای درک اجزای اصلی شبکه مفید باشد. اگر می‌خواهید سلسله مراتب کامل کلاسهای موجود در NS را ببینید می‌توانید به آدرس زیر مراجعه کنید:
http:www.sop.inria.fr/rodeo/personnel/Antoine. Clerget/ns.



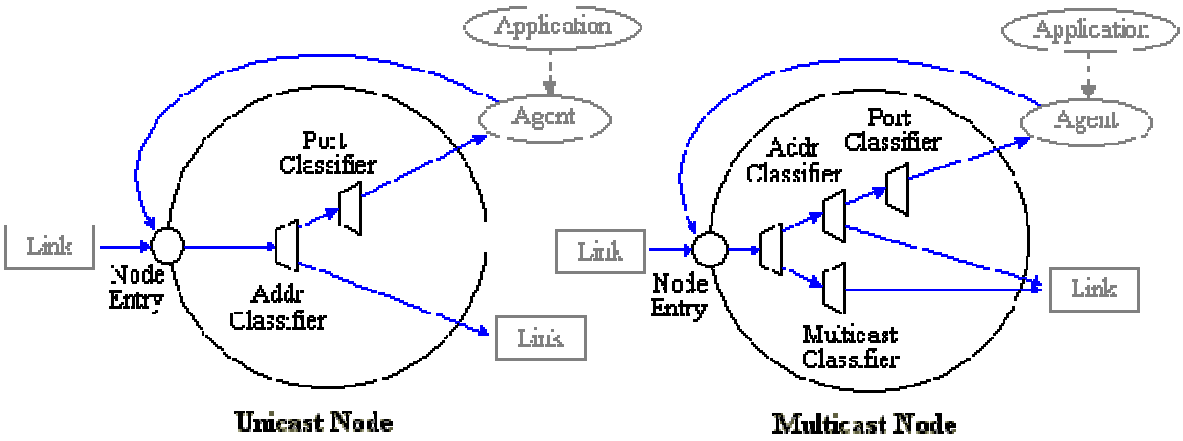
شکل ۴: بخشی از سلسله مراتب کلاس

ریشه اصلی این سلسله مراتب، کلاس OTCL Object است که یک فوق کلاس (Superclass) از تمام اشیاء کتابخانه OTCL از قبیل زمان‌بند، اجزای شبکه، تایمرها و بقیه اشیاء مرتبط با NAM می‌باشد. کلاس NS object بعنوان یک کلاس از اجداد کلاس OTCL object است که خود بصورت فوق کلاس از تمامی اجزای اصلی شبکه مثل گره‌ها و لینک‌ها که کار رسیدگی و جابجایی بسته‌ها را بر عهده دارند، است. اجزای اصلی شبکه هم به نوبهٔ خود به دو زیر کلاس طبقه‌بندی کننده (Classifer) و رابط (Connector) تقسیم می‌شوند. این تقسیم‌بندی بر اساس تعداد مسیرهای ممکن از داده خروجی انجام می‌گیرد. آن اجزایی از شبکه که فقط یک مسیر داده خروجی را دارند در کلاس رابط و آنهایی که دارای چندین مسیر داده خروجی هستند در کلاس طبقه‌بندی کننده قرار می‌گیرند.

۱-۴-۵- گره و مسیریابی

گره یک شیئی ترکیبی است که از دو بخش ورودی و طبقه‌بندی کننده تشکیل شده است (شکل ۵). دو نوع گره در NS وجود دارد. نوع اول تک‌پخشی (unicast) است و دارای طبقه‌بندی کننده آدرس، که عمل مسیریابی تک‌پخشی را انجام می‌دهد، و طبقه‌بندی کننده پورت می‌باشد. نوع دوم گره چندپخشی است که، علاوه بر اجزای گره تک‌پخشی، دارای طبقه‌بندی کننده، که بسته‌های چندپخشی را از بسته‌های تک‌پخشی جدا می‌کند، و نیز طبقه‌بندی کننده چندپخشی است که عمل مسیریابی چندپخشی را بر عهده دارد. در NS گره‌ها بصورت پیش فرض از نوع تک‌پخشی انتخاب می‌شوند و برای ایجاد گره‌های چندپخشی، کاربر باید بصورت صریح در متن برنامه OTCL ورودی اعلام کند که گره‌ها از نوع چندپخشی هستند. این اعلام باید درست بعد از ایجاد شیئی زمان‌بند صورت پذیرد. بعد از اینکه نوع گره مشخص شد، کاربر می‌تواند یک قرارداد مسیریابی مشخص را بجای قرارداد مسیریابی پیش‌فرض انتخاب کند. برای تعریف گره بصورت تک‌پخشی از دستور زیر استفاده کنید:

$ns rtproto type



شکل ۵ :گره چندپخشی و تک پخشی

که در قسمت type می‌توانید از انواع زیر برای گره تک پخشی استفاده کنید:

Static, Session, DB, Cost, multi-path

برای تعریف گره بصورت چندپخشی از دستور زیر استفاده کنید:

$ns multicast
$ns mrtproto type

در قسمت type می‌توان یکی از انواع زیر را انتخاب کنید:

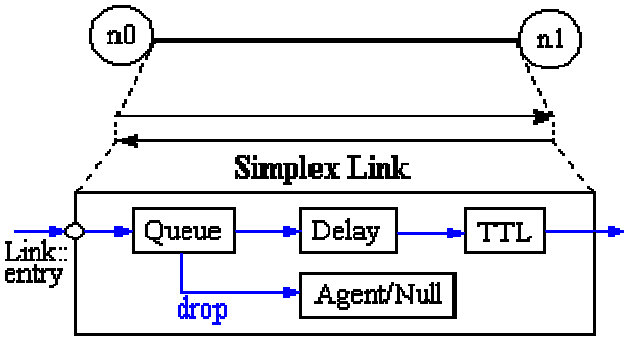
CtrMcast, DB, St, BST

باید به این نکته دقت کنید که دستور $ns multicast باید دقیقاً set $ns [new Scheduler] باشد. برای اطلاعات بیشتر می‌توانید به NS Manual در آدرس زیر مراجعه کنید. در آنجا چند فصل دربارهٔ مسیریابی چندپخشی و تک پخشی وجود دارد.

http://www.isi.edu/nsnam/ns/ns-documentation.html

مانند گره‌ها، لینک‌ها هم جزء اجزای ترکیبی در NS می‌باشند. وقتی که کاربر یک لینک از نوع دوطرفه به وسیله توابع عضو ایجاد کرد، آنگاه دو لینک ساده یک‌طرفه در هر دو جهت مطابق شکل ۶ ایجاد می‌شود.

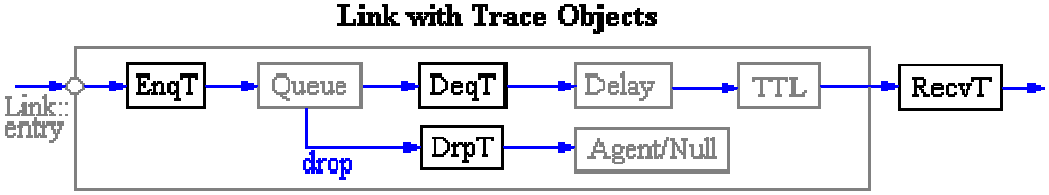
یک نکته قابل توجه این است که صف خروجی از گره در حقیقت بعنوان قسمتی از لینک ساده عمل می‌کند. بسته‌هایی که در صف قرارمی‌گیرند به قسمت تأخیر (Delay) که عمل شبیه‌سازی تأخیر لینک را برعهده دارد، فرستاده می‌شوند و بسته‌هایی که از صف بیرون انداخته می‌شوند به قسمت Null رفته و آنجا آزاد می‌شوند. در انتها قسمت TTL (time to live) پارامترهای زمان حیات هر بسته دریافت شده را محاسبه می‌کند و قسمت TTL بسته‌ها را با مقدار جدیدی تغییر می‌دهد.



شکل ۶ :لینک ساده

۱-۴-۶- ردیابی (Tracing)

در NS، فعالیت‌های شبکه در یک لینک ساده ردیابی می‌شوند. اگر به شبیه‌ساز از طریق دستورهای $ns trace-all file و یا $ns namtrace-all file ردیابی فعالیت‌های شبکه ابلاغ شود، آنگاه لینک‌ها بعد از اجرای دستورهای فوق قسمتهایی را داخل لینک مطابق شکل ۷ وارد می‌کنند. کاربرها هم می‌توانند بطور مخصوص عمل ردیابی بین گره‌های مبدأ (src) و مقصد (dst) را با اجرای دستور create-trace {type file src dst} انجام دهند.

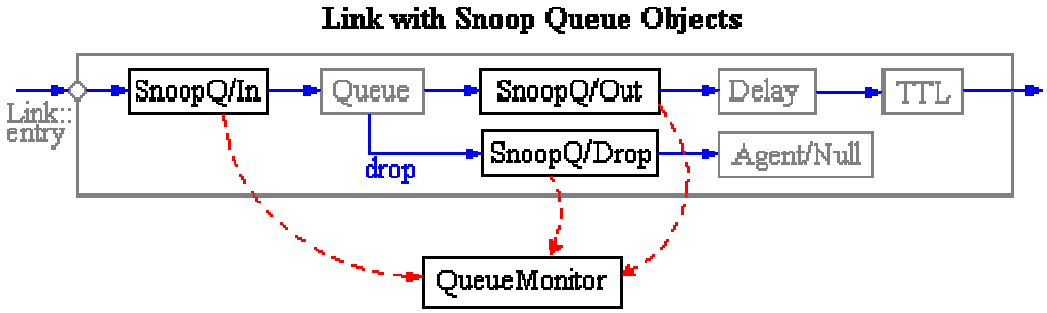


شکل ۷: واردکردن شیئی‌های ردیابی

هنگامی که هر یک از قسمتهای اضافه شده برای ردیابی یعنی DrpT, DeqT, EnqT و RcevT یک بسته را دریافت کردند، آنگاه توضیحی را در یک فایل ردیابی بدون تلف کردن زمان شبیه‌سازی می‌نویسند و بسته را به قسمت بعدی تحویل می‌دهند. قالب ردیابی در قمست آنالیز عمومی مسئله مورد بررسی قرار خواهد گرفت.

۲-۹-۱ پایش صف (Queue Monitor)

اصولاً عمل ردیابی برای ذخیره‌کردن زمان رسیدن بسته به قسمتهای ردیابی طراحی شده است. هرچند یک کاربر می‌تواند اطلاعات کافی را از ردیابی بدست آورد اما ممکن است اتفاقات انجام شده داخل یک صف خروجی خاص برایش جالب باشد. بعنوان مثال یک کاربر به رفتار صف RED علاقه‌مند بوده و می‌خواهد اندازه متوسط صف و اندازه صف جاری خاصی از نوع RED را اندازه‌گیری کند یا به عبارت ساده‌تر نیاز به مشاهده صف دارد. مشاهده صف را می‌توان با استفاده از شیئ‌های مشاهده صف و شیئ‌های جستجوکننده (Snoop) صف که در شکل ۸ نشان داده شده است، انجام داد.



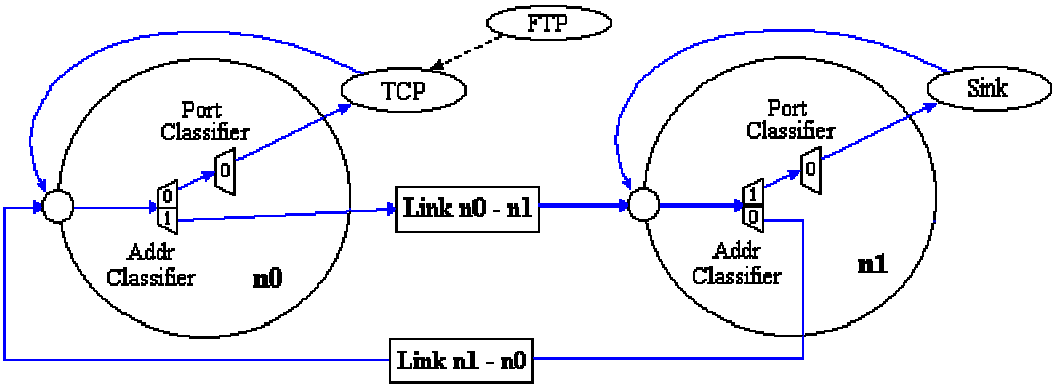
شکل ۸: مشاهده صف

هنگامی که یک بسته دریافت می‌شود، قسمت جستجو کننده صف به قسمت نمایش صف این رخداد را اعلام می‌کند و قسمت نمایش صف با این اطلاعات صف را نشان می‌دهد. نمایش صف RED در یک مثال نشان داده خواهد شد. توجه کنید که قسمتهای جستجوکننده صف می‌توانند بصورت موازی با قسمتهای ردیابی قرار گیرند، اگرچه در شکل ۸ نشان داده نشده است.

۲-۹-۲ مثالی از جریان بسته

تا اینجا دو قسمت مهم شبکه یعنی گره و لینک مورد بررسی قرار گرفته‌اند. شکل ۹ مثالی از شبیه‌سازی حرکت بسته را داخل یک شبکه شامل دو گره n0 و n1 با آدرسهای 0 و 1 نشان می‌دهد. عامل TCP که به

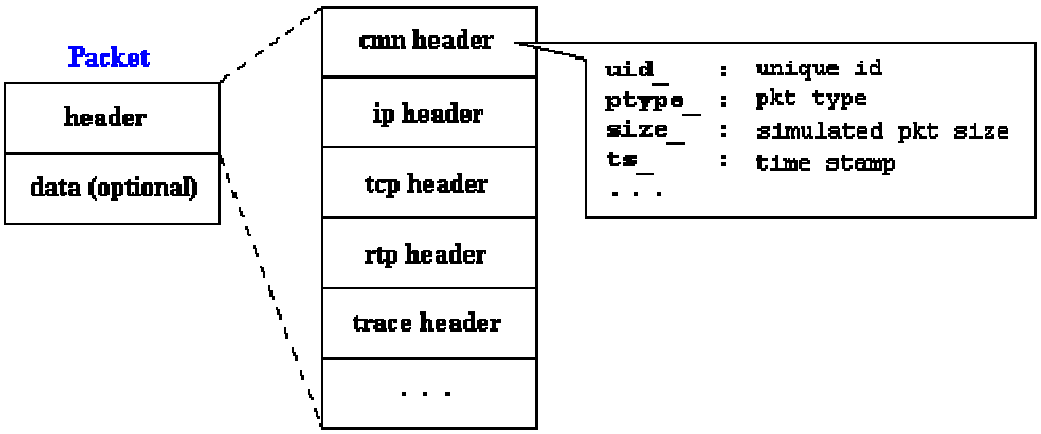
گره n0 متصل است، از طریق پورت صفر با قسمت TCP sink که به پورت صفر گره n1 وصل است، ارتباط دارد. یک کاربرد FTP یا در حقیقت منبع ترافیک به عامل TCP متصل شده است و درخواست ارسال مقداری داده را دارد.



شکل ۹: مثالی از حرکت بسته

۱-۴-۷- بسته

در NS یک بسته متشکل از یک پشته (stack) از سرآمدها و قسمت داده اختیاری می‌باشد. همانطور که در مثال ساده از شبیه‌سازی بطور خلاصه گفته شد، فرمت سرآمد بسته در هنگام ایجاد شیئی شبیه‌ساز، مقداردهی اولیه می‌شود. پشته، شامل سرآمدهای cmm که مورد نیاز همهٔ اشیاء می‌باشد، سرآمد IP، سرآمد TCP، سرآمد RTR (UDP از سرآمد RTP استفاده می‌کند) و بالاخره سرآمد trace است. برای هر یک از سرآمدها در پشته، میزان افست در محلی ذخیره می‌شود. به این معنی که هرگاه یک سرآمد معین مورد استفاده قرار گیرد، هنگام تخصیص بسته به وسیله یک عامل، پشته‌ای از سرآمدهای ذخیره شده تولید خواهد شد و بنابراین یک شیئ شبکه می‌تواند به هر یک از سرآمدهای داخل پشته دسترسی داشته باشد. این دسترسی، از طریق پردازش‌های لازم بر روی مقدار افست مربوطه مقدور خواهد بود.



شکل ۱۰: فرمت بسته در NS

معمولاً یک بسته فقط شامل پشته‌ای از سرآمدها است و اشاره‌گر به فضای داده مقدار Null را دارد. اگر چه یک بسته می‌تواند داده واقعی را از یک کاربر، با تخصیص فضای داده حمل کند، اما تعداد کاربردها و عاملهایی که از این روش استفاده می‌کنند بسیار کم است، چرا که حمل داده در یک شبیه‌سازی بصورت زمان غیرواقعی، بی‌معنی است. اما اگر شما بخواهید یک کاربرد با کاربرد دیگر در شبکه صحبت کند، می‌توانید از روش حمل داده، با کمی تغییر در اجرای عامل مورد نظر استفاده نمایید. روش دیگر، ایجاد یک سرآمد جدید برای کاربرد و تغییر در عامل مورد نظر برای نوشتن داده دریافت شده از کاربرد در آن سرآمد جدید می‌باشد. روش دوم را در قسمت اضافه کردن یک کاربرد و عامل جدید، تحت یک مثال خواهید دید.

۱-۴-۸- مثال تجزیه و تحلیل ردیابی

این قسمت مثالی از تجزیه و تحلیل مربوط به ردیابی را نشان می‌دهد. مثال ۴ یک برنامه OTCL است که شبیه مثال قسمت (مثالی از شبیه‌سازی ساده) می‌باشد، اما فقط چند خط برای باز کردن یک فایل ردیابی و نوشتن نتایج ردیابی در آن فایل به برنامه اضافه شده است. توپولوژی شبکه مانند شکل ۴ در قسمت مثالی از شبیه‌سازی ساده می‌باشد. با اجرای برنامهٔ بالا یک فایل ردیابی NAM ایجاد می‌شود که بعنوان ورودی برای برنامه NAM مورد استفاده قرار می‌گیرد، همچنین یک فایل ردیابی به اسم out.tr نیز تولید می‌شود که برای تحلیل شبیه‌سازی از آن استفاده خواهیم کرد. شکل ۱۱ فرمت ردیابی و مثالی از داده ردیابی فایل out.tr را نشان می‌دهد. هر خط مربوط به ردیابی با یکی از توصیف‌گرهای r, d, -, + شروع می‌شود. سپس زمان شبیه‌سازی رخداد بر حسب ثانیه می‌آید و در ادامه قسمت از گره تا گره قرار دارد و لینکی را که قرار است رخداد بر روی آن اتفاق بیفتد را مشخص می‌کند.

```
* * *
#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Close the Trace file
        close $tf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}
* * *
```

مثال ۴: فعال‌سازی ردیابی در مورد یک برنامه ساده NS

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

```
r : receive (at to_node)
+ : enqueue (at queue)                src_addr : node.port (3.0)
- : dequeue (at queue)                dst_addr : node.port (0.0)
d : drop    (at queue)
```

```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

شکل ۱۱: مثالی از فرمت ردیابی

شکل ۹ در قسمت (اجزای شبکه) نشان می‌دهد که کجای لینک، هر نوع از رخدادها را دنبال می‌کند. توصیف‌گر r به معنی دریافت در گره مشخص شده در قسمت to-node است. توصیف‌گر d هم به معنی انداختن (drop) در صف می‌باشد. توصیف‌گرهای + و – هم به ترتیب برای انجام عملیات enqueue و dequeue در صف هستند. قسمت بعدی در خط ردیابی که قبل از بخش پرچم (flag) قرار دارد، اندازه و نوع بسته هستند که برحسب بایت بیان می‌شوند.

در شکل ۱۱ پرچم‌ها با علام "....." مشخص شده‌اند که بیان می‌کنند هیچ پرچمی هنوز تنظیم نشده است. معمولاً NS فقط با ECN (Explicit Congestion Notification) کار می‌کند و بقیه بیت‌ها را استفاده نمی‌کند. قسمت بعدی شناسایی جریان (flow identification) یا بطور خلاصه fid است که کاربر می‌تواند آنرا در برنامه OTCL برای هر یک از جریانها تنظیم کند. اگر چه قسمت fid ممکن است در شبیه‌سازی استفاده نشود اما کاربرها می‌توانند از این قسمت برای مقاصد تجزیه و تحلیل استفاده نمایند. یکی دیگر از موارد استفاده از قسمت fid وقتی است که در نمایش برنامه NAM از رنگ‌ها استفاده می‌شود. دو قسمت دیگر بعد از fid آدرسهای مبداء و مقصد هستند که به فرم node.port می‌باشند. بعنوان مثال آدرس گره مبداء و مقصد می‌تواند بصورت زیر باشد:

Src-addr: node.port (3.0)
Dst-addr: node.port (0.0)

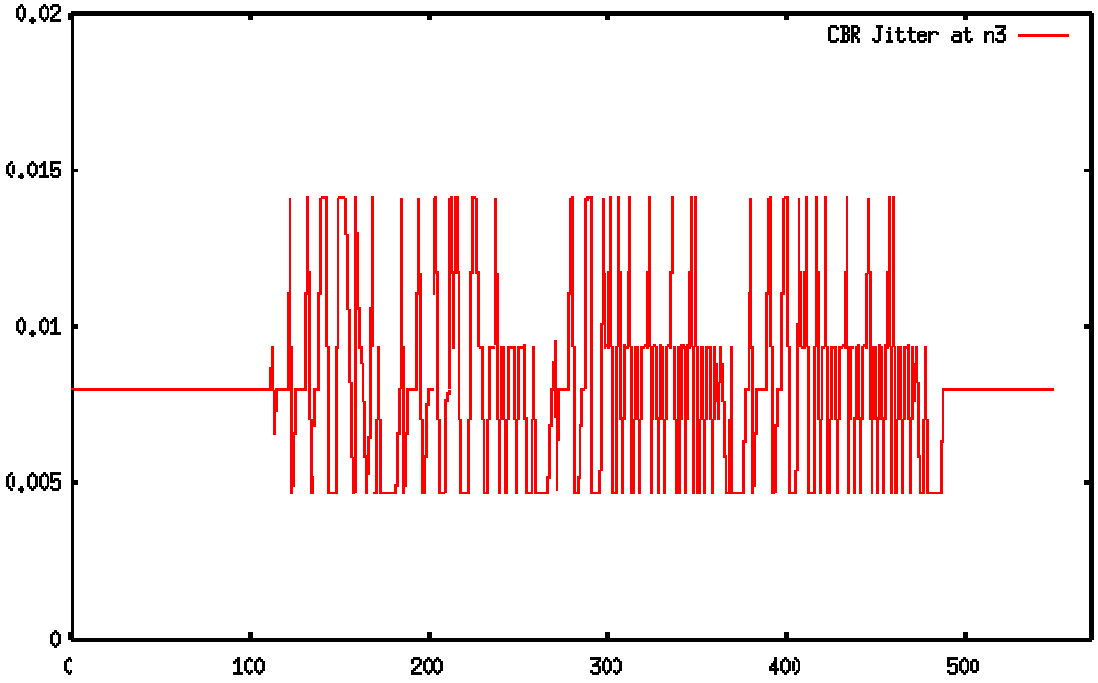
قسمت بعدی شمارهٔ سکانس بسته مربوط به قرارداد لایه شبکه را نشان می‌دهد. توجه داشته باشید که اگر چه UDP از شمارهٔ سکانس استفاده نمی‌کند اما NS شمارهٔ سکانس بسته را نگهداری می‌کند تا برای مقاصد تجزیه و تحلیل مورد استفاده قرار دهد. قسمت آخر هم مربوط به id بسته می‌باشد که یک مقدار منحصر به فرد است و برای هر بسته متفاوت می‌باشد.

حال که داده‌های مربوط به ردیابی را در اختیار داریم، تنها کار لازم تبدیل قسمتی از داده‌های مورد علاقه به اطلاعات قابل درک و سپس تجزیه و تحلیل آنها می‌باشد. در زیر مثالی از تبدیل داده‌ای که مقدار کمی دارد، آورده شده است. در این برنامه از دستور column استفاده شده است که ستونهایی از داده را انتخاب می‌کند. برای اینکه این برنامه روی کامپیوتر شما اجرا شود باید ابتدا column را بصورت قابل اجرا درآورید. برای این کار باید دستور chmod 755 column را در خط فرمان اجرا نمایید. دستور awk هم وظیفهٔ محاسبهٔ لرزش (jitter) ترافیک CBR که در گرهٔ n3 دریافت می‌شود را بر اساس داده فایل out.tr برعهده دارد. این دستور همچنین نتایج را در فایل jitter.txt ذخیره می‌کند.

```
Cat out.tr |grep "2 3 cbr" |grep ^r  | column 1 10 |awk '{dif=2-old2;
If (dif==0) dif=1;
If (dif>0) {printf("%d\t%n" , $2, ($1 –old1) /dif);
Old1=$1; old2=$2}}'>jitter.text
```

این برنامه رخداد دریافت بسته CBR را در گرهٔ n3 انتخاب کرده، همچنین ستون ۱ که مربوط به زمان و ستون ۱۰ که مربوط به شمارهٔ سکانس است را انتخاب می‌کند، سپس اختلاف زمان دریافت بسته را با بسته قبلی محاسبه کرده و بر تفاضل شمارهٔ سکانس آنها تقسیم می‌کند. شکل زیر، گراف لرزش مربوطه که بوسیله gnuplot رسم شده است را نشان می‌دهد. محور افقی X شمارهٔ سکانس بسته و محور Y نشانگر زمان شبیه‌سازی بر حسب ثانیه است.

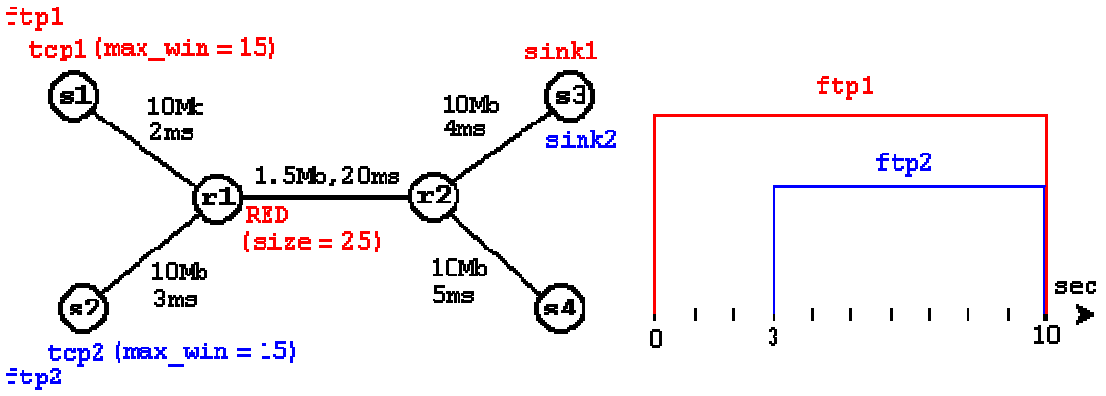
این بخش مثالی را نشان داد که چگونه می‌توان ردیابی را در NS انجام داد و یا چگونه آنها را تفسیر کرد، همچنین چگونه می‌توان اطلاعات مفید را از ردیابی بیرون کشید. در این مثال پروسهٔ شبیه‌سازی بعد از عملیات شبیه‌سازی در خط فرمان اجرا شده است. هرچند که می‌توان این پردازش‌های نهایی را در ورودی OTCL انجام داد که این موضوع را در قمست بعدی مطالعه خواهید کرد.



شکل ۱۲: لرزش CBR در گره n3

۱-۴-۹- مثالی از نمایشگر صف RED

این بخش مثالی از نمایش صف RED را نشان می‌دهد. مثال ۵ یک برنامهٔ OTCL است که توپولوژی شبکه نشان داده شده در شکل ۱۳ را ایجاد کرده و عملیات شبیه‌سازی مربوط را انجام می‌دهد. توجه داشته باشید که برای لینک بین r1 و r2 از صف RED که توانایی داشتن ۲۵ بسته را دارد، استفاده شده است. هدف این است که ببینیم صف RED چگونه کار می‌کند و این عمل بوسیله اندازه‌گیری جریان و متوسط اندازه صف انجام می‌شود. برنامه زیر را در فایل به اسم red.tcl ذخیره کنید و سپس در خط فرمان عبارت ns red.tcl را اجرا کنید.



شکل ۱۳: تنظیمات مربوط به مثال نمایشگر صف RED

دو نکتهٔ قابل توجه در برنامه عبارتند از:

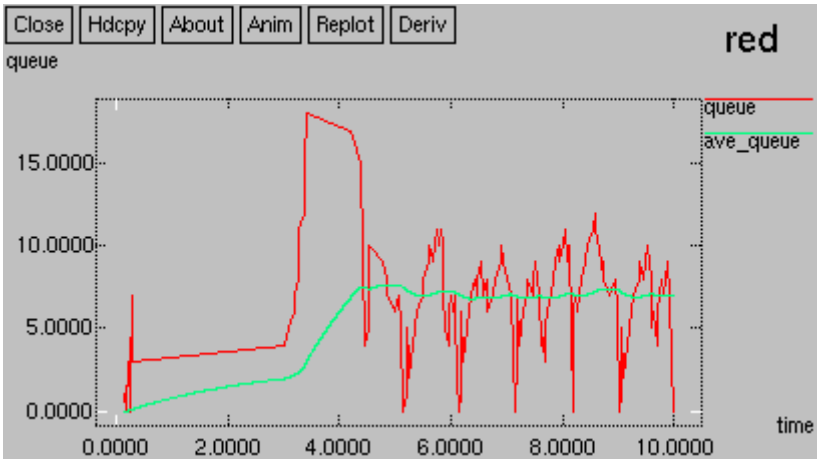
اول اینکه تابع پیشرفته create-connection برای تولید اتصالات Tcp مورد استفاده قرار گرفته است. دوم اینکه با دقت به قسمت ردیابی صف (نمایشگر) در می‌یابیم که این خطوط برنامه، باعث ساختن یک متغیر برای اشاره به شیئ صف RED و فراخوانی توابع ردیابی برای نمایش اندازه صف جاری (curq_) و نیز اندازه متوسط صف (avg_) خواهند شد. همچنین نتایج در فایلی به اسم all.q نوشته می‌شوند. دو فرمت خروجی ردیابی صف برای اندازه متوسط صف و اندازه صف جاری بصورت زیر می‌باشند.

A time ave_q_ size
Q time crnt-q_ size

تا اینجا تمام کارهای لازم برای نمایش صف RED به غیر از بستن فایل all.q هنگام شبیه‌سازی، که جزء پروسه پردازش نهایی است، انجام شده‌اند.

```
set ns [new Simulator]

set node_(s1) [$ns node]
set node_(s2) [$ns node]
set node_(r1) [$ns node]
set node_(r2) [$ns node]
set node_(s3) [$ns node]
set node_(s4) [$ns node]

$ns duplex-link $node_(s1) $node_(r1) 10Mb 2ms DropTail
$ns duplex-link $node_(s2) $node_(r1) 10Mb 3ms DropTail
$ns duplex-link $node_(r1) $node_(r2) 1.5Mb 20ms RED
$ns queue-limit $node_(r1) $node_(r2) 25
$ns queue-limit $node_(r2) $node_(r1) 25
$ns duplex-link $node_(s3) $node_(r2) 10Mb 4ms DropTail
$ns duplex-link $node_(s4) $node_(r2) 10Mb 5ms DropTail
 •
 •
 •
set tcp1 [$ns create-connection TCP/Reno $node_(s1) TCPSink $node_(s3) 0]
$tcp1 set window_ 15
set tcp2 [$ns create-connection TCP/Reno $node (s2) TCPSink $node (s3) 1]
$tcp2 set window_ 15
set ftp1 [$tcp1 attach-source FTP]
set ftp2 [$tcp2 attach-source FTP]

# Tracing a queue
set redq [$ns link $node_(r1) $node_(r2)] queue]
set tchan_ [open all.q w]
$redq trace curq_
$redq trace ave_
$redq attach $tchan_


$ns at 0.0 "$ftp1 start"
$ns at 3.0 "$ftp2 start"
$ns at 10 "finish"

# Define 'finish' procedure (include post-simulation processes)
proc finish {} {
    global tchan_
    set awkCode {
        {
            if ($1 == "Q" && NF>2) {
                print $2, $3 >> "temp.q";
                set end $2
            }
            else if ($1 == "a" && NF>2)
            print $2, $3 >> "temp.a";
        }
    }
    set f [open temp.queue w]
    puts $f "TitleText: red"
    puts $f "Device: Postscript"

    if { [info exists tchan_] } {
        close $tchan_
    }
    exec rm -f temp.q temp.a
    exec touch temp.a temp.q

    exec awk $awkCode all.q

    puts $f \"queue
    exec cat temp.q >@ $f
    puts $f \n\"ave_queue
    exec cat temp.a >@ $f
    close $f
    exec xgraph -bb -tk -x time -y queue temp.queue &
    exit 0
}

$ns run
```

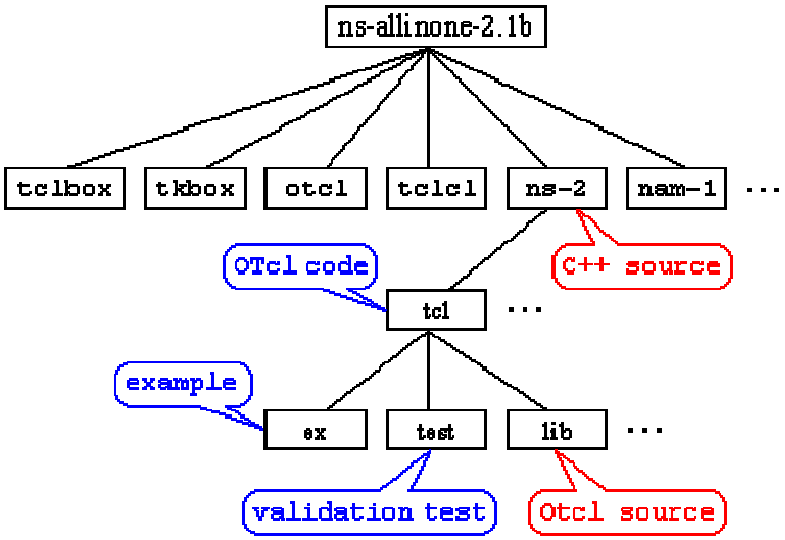
مثال ۵: برنامه‌ای برای ایجاد نمایشگر صف



شکل ۱۴: گراف ردیابی صف Red

۱-۴-۱۰- چه چیزی را کجا پیدا کنیم؟

قبل از اینکه وارد بحث گسترش NS شویم، اجازه دهید بطور مختصر بررسی کنیم چه اطلاعاتی در کدام شاخه یا فایل ذخیره شده‌اند. شکل ۱۵ساختار شاخه‌ای شبیه‌ساز NS را نشان می‌دهد. در اینجا فرض شده است که شما ns-allinone-2.1b را نصب کرده‌اید. در بین زیرشاخه‌های ns-2, ns-allinone-2.1b مکان قرار گرفتن تمام اجرائیات شبیه‌ساز، چه در محیط C++ و یا چه در محیط OTCL ، تست معتبربودن برنامه‌های OTCL و مثالهایی از OTCL است. در این شاخه تمام برنامه‌های OTCL چه تست و چه مثال در داخل یک زیرشاخه از آن به نام TCL قرار گرفته است. اغلب کدهای برنامه C++ که برای زمانبندی رخداد و کلاسهای شیئ اجزا شبکه هستند در شاخهٔ اصلی ns-2 قرار دارند. بعنوان مثال اگر شما می‌خواهید کاربرد عوامل UDP را ملاحظه کنید، باید به زیرشاخه ns-allinone-2.1b/ns-2 بروید و فایل udp.h یا udp.cc را ببینید.



شکل ۱۵: ساختار شاخه‌ها در NS

برای سلسله مراتب کلاس مربوط به اجزاء شبکه، می‌توانید به شکل ۶ در بخش اجزای شبکه مراجعه کنید. از حالا به بعد، فرض می‌شود که شما در شاخه‌ها ns-allinone-2.1b قرار دارید. شاخه tcl دارای چندین زیرشاخه نظیرLib است که شامل کدهای OTCL برای اغلب قسمتهای ضروری و برای اجرای NS (عامل، گره، لینک، بسته، آدرس، مسیریابی و غیره) می‌باشد. بقیه کدهای OTCL برای قسمتهای مانند LAN ، Web و اجرای چندپخشی در زیرشاخهٔ مجزا از tcl قرار دارند. آنها در زیر لیستی از فایلهای داخل شاخه ns-2/cl/lib آورده شده‌اند.

ns-lib.tcl

کلاس شبیه‌ساز و بیشتر تعاریف توابع عضو آن بجز LAN و Web و مربوط به چندپخشی در این فایل قرار دارند. اگر می‌خواهید ببینید که کدام یک از توابع کلاس شیئ شبیه‌ساز در دسترس هستند و چگونه کار می‌کنند، باید به این قسمت مراجعه کنید.

ns-packet.tcl

در اینجا اجرای مقداردهی اولیه برای فرمت سرآمد بسته‌ها، انجام می‌شود. وقتی یک سرآمد جدید برای بسته تولید می‌کنید، باید این سرآمد را در این فایل ذخیره کنید تا در هنگام مقداردهی اولیه به سرآمدها، سرآمد موردنظر شما هم در پشته سرآمدها قرار گیرد و افست سرآمد به شما اطلاع داده شود. مثال ایجاد سرآمد جدید در قسمت (اضافه کردن کاربرد یا عامل جدید) آورده شده است.

بقیه فایلهای OTCL:

سایر فایلهای OTCL موجود در این شاخه، شامل اجرای اشیاء ترکیبی شبکه یا اشیاء کنترلی شبکه در C++هستند. کاربرد FTP در OTCL قابل اجرا بوده و کد برنامه مربوط به آن در فایل ns-source.tcl آمده است. دو زیرشاخهٔ دیگر از tcl که ممکن است برای کاربری که می‌خواهد طراحی انجام دهد مفید باشند، زیر شاخه‌های ex و test هستند. شاخه اول شامل برنامه‌های متعدد شبیه‌سازی بوده و دومی دارای برنامه‌هائی

است که نصب درست NS را بر روی کامپیوتر شما نشان خواهد داد. هنگامی که این شبیه‌سازیها اجرا شوند و نتایج آنها با نتایج مورد انتظار مقایسه گردد، صحت نصب صحیح NS مشخص خواهد شد.

۱-۴-۱۱- پیوند OTCL

گسترش NS بوسیله افزودن یک شیئ شبکه پایه‌ای جدید، اغلب مستلزم کار پیوند OTCL با کد C++ است چرا که کلاس شیئ در مسیر داده باید به دلیل کارآیی در C++ نوشته شود. این بخش پیوندهای بین C++/OTCL که در NS موجود هستند را به کمک یک مثال از تولید عامل ساده و کند (dull) که My Agent نامیده شده است و هیچ بسته‌ای تولید و ارسال نمی‌شود بیان می‌کند. شکل ۱۶ تا شکل ۱۹ بخشهایی از فایل C++ برای My Agent را نشان می‌دهند که با یکدیگر یک اجرای کامل را می‌سازند. در انتهای این بخش، یک برنامه OTCL به کمک آن می‌توانید My Agent را تست کنید، آورده شده است.

۱-۴-۱۲- صادر کردن کلاس از C++ به OTCL

فرض کنید که شما یک کلاس جدید شیئ شبکه تولید کرده‌اید با فراخوانی My Agent که از کلاس Agent اخذ شده است. می‌توانید این امکان را فراهم آورید که نمونه‌ای از این کلاس را در OTCL تولید نمایید. برای انجام آن مجبورید که یک شیئ پیوند تعریف کنید، بعنوان مثال بگویید My Agent Class که باید از Tcl Class اخذ شده اشد. این پیوند یک شیئ OTCL با اسم خاصی (در این مثال Agent/My Agent OCL) را ایجاد می‌کند و بین شیئ OTCL و شیئ C++ (در این مثال My Agent) یک پیوند برقرار می‌نماید. شکل ۱۶ تعریف کلاس My Agent و تعریف کلاس پیوند را نشان می‌دهد.

وقتی که NS شروع بکار می‌کند، سازنده (constructor) را برای متغیر استاتیک class-my-agent اجرا می‌کند و بنابراین یک مثال از My Agent Class را تولید می‌کند. در این پروسه، کلاس Agent/My Agent OTCL و توابع عضو در محیط OTCL ایجاد می‌شوند.

```
class MyAgent : public Agent {
public:
        MyAgent();
protected:
        int command(int argc, const char*const* argv);
private:
        int    my_var1;
        double my_var2;
        void   MyPrivFunc(void);
};
```

```
static class MyAgentClass : public TclClass {
public:
        MyAgentClass() : TclClass("Agent/MyAgentOtcl") {}
        TclObject* create(int, const char*const*) {
                return(new MyAgent());
        }
} class_my_agent;
```

شکل ۱۶: مثال C++ از جزء شبکه و شیئ پیوند

هنگامیکه NS برای اولین بار اجرا می‌شود، سازنده (constructor) را برای متغیر استاتیک class-my-agent اجرا می‌کند و بنابراین MyAgent ایجاد می‌گردد. در طی این عملیات کلاس Agent/My AgenOtcl و توابع کتابخانه‌ای آن در محیط OTCL تولید می‌شوند.

۱-۴-۱۳- صادر کردن متغیر از C++ به OTCL

فرض کنید که شیئ جدید C++ برنامه شما یعنی MyAgent دارای دو متغیر my_var1 و my_var2 است که شما می‌خواهید آنها را از طریق برنامه OTCL تغییر بدهید. برای این منظور باید برای هر کدام از متغیرهایی که می‌خواهید آنها را صادر کنید، یک تابع تخصیص بدهید. شکل ۱۷چگونگی این کار را نشان می‌دهد. شبیه‌ساز NS چهار نوع مختلف از توابع را برای پنج نوع از متغیرها مورد حمایت قرار می‌دهد:

```
MyAgent::MyAgent() : Agent(PT_UDP) {
        bind("my_var1_otcl", &my_var1);
        bind("my_var2_otcl", &my_var2);
}
```

شکل ۱۷: مثالی از تخصیص متغیرها

- Bind( ) :real or integer variables
- Bind_time ( ) :time variable
- Bind_bw ( ) :bandwith variable
- Bind_bool ( ):Boolean variable

۱-٤-۱۳-۱- صادر کردن فرمانهای کنترلی از C++ به OTCL

در هنگام صادر کردن متغیرها از C++ به OTCL، ممکن است بخواهید کنترل آنها را نیز به بسپارید. این کار با تعریف تابع محلی command در MyAgent که به عنوان مفسر دستورات OTCL است، انجام می‌شود شکل ۱۸مثالی از تعریف تابع command برای شیئ شکل ۱۶ را نشان می‌دهد.

```
int MyAgent::command(int argc, const char*const* argv) {
        if(argc == 2) {
                if(strcmp(argv[1], "call-my-priv-func") == 0) {
                        MyPrivFunc();
                        return(TCL_OK);
                }
        }
        return(Agent::command(argc, argv));
}
```

شکل ۱۸: مثالی از مفسر دستور OTCL

۱-٤-۱۳-۲- اجرای یک دستور OTCL از C++

هنگامیکه یک شیئ جدید شبکه را در C++ ایجاد می‌کنید، ممکن است بخواهید یک دستور OTCL را از شیئ C++ اجرا نمایید. شکل ۱۹ نشان می‌دهد که تابع MyPrivFunc که یک تابع محلی از MyAgent در شکل ۱۶ است، چگونه مفسر OTCL را مجبور به چاپ مقدار متغیرهای محلی my_var1 و my_var2 می‌کند.

```
void MyAgent::MyPrivFunc(void) {
        Tcl& tcl = Tcl::instance();
        tcl.eval("puts \"Message From MyPrivFunc\"");
        tcl.evalf("puts \"      my_var1 = %d\"", my_var1);
        tcl.evalf("puts \"      my_var2 = %f\"", my_var2);
}
```

شکل ۱۹: اجرای دستور OTCL از شیئ C++

برای اجرای یک دستور OTCL از C++، باید به Tcl::instance() که به عنوان یک متغیر ایستا از قبل معرفی شده است، رجوع کنید. این مثال دو راه برای انتقال یک دستور OTCL به مفسر بیان می‌کند. برای داشتن لیست کاملی از توابعی که این انتقال را انجام می‌دهند، به N.S. Manual رجوع کنید.

۱-٤-۱۳-۳- اجرا، کامپایل و تست کردن MyAgent

تا این قسمت، متغیرهای ضروری پیوند OTCL در N.S. را به کمک مثال MyAgent آزمایش کردیم. از آنجایی که اجرا و آزمایش این مثال به فهم بیشتر خواننده کمک می‌کند، یک روش که برای کامپایل، اجرا و تست MyAgent لازم است، پیشنهاد می‌کنیم.

۱- فایل ex.linkage.cc را در شاخه ns-2 ذخیره کنید.

۲- Makefile را باز کرده و فایل ex.linkage.o را به انتهای لیست اشیاء اضافه کنید.

۳- به کمک دستور make شبیه‌ساز .N.S را مجدداً کامپایل کنید.

۴- فایل ex.linkage.tcl که شامل دستورات OTCL برای تست MyAgent است، را به کمک دستور ex.linkage.tcl اجرا کنید.

برنامه ex.linkage.cc و نتیجه آن را در زیر ملاحظه می‌کنید.

```
 //Jae Chung 7-13-99
 //Example of a aimple and dull Agent that
 //illustrates the use of OTcl linkages


#include <stdio.h>
#include <string.h>
#include "agent.h"


class MyAgent : public Agent {
public:
        MyAgent;()
protected:
        int command(int argc, const char*const* argv);
private:
        int    my_var1;
        double my_var2;
        void   MyPrivFunc(void);
};


static class MyAgentClass : public TclClass {
public:
        MyAgentClass() : TclClass("Agent/MyAgentOtcl"){}
        TclObject* create(int, const char*const*) {
                return(new MyAgent;(();
        }
 {class_my_agent;


MyAgent::MyAgent() :             Agent(PT_UDP)}
       bind("my_var1_otcl", &my_var1);
       bind("my_var2_otcl", &my_var2);
}


int MyAgent::command(int argc, const char*const* argv){
      if(argc == 2) {
           if(strcmp(argv[1], "call-my-priv-func") == 0){
                  MyPrivFunc;()
                  return(TCL_OK;(
           }
      }
      return(Agent::command(argc, argv();
}
```

```
void MyAgent::MyPrivFunc(void ){
      Tcl& tcl = Tcl::instance;();
      tcl.eval("puts \"Message From MyPrivFunc""\);
      tcl.evalf("puts \"     my_var1 = %d\"", my_var1);
      tcl.evalf("puts \"     my_var2 = %f\"", my_var2);
}
```

ex-linkage.tcl

```
# Create MyAgent (This will give two warning messages that
# no default vaules exist for my_var1_otcl and my_var2_otcl)
set myagent [new Agent/MyAgentOtcl]


# Set configurable parameters of MyAgent
$myagent set my_var1_otcl 2
$myagent set my_var2_otcl 3.14

# Give a command to MyAgent
$myagent call-my-priv-func
```

شکل ۲۰:ex.linkage.tcl

result

```
warning: no class variable Agent/MyAgentOtcl::my_var1_otcl

        see tcl-object.tcl in tclcl for info about this warning.

warning: no class variable Agent/MyAgentOtcl::my_var2_otcl

Message From MyPrivFunc
     my_var1 = 2
     my_var2 = 3.140000
```

شکل ۲۱: نتیجه مربوط به برنامه ex.linkage.tcl

۱-۴-۱۴- اضافه کردن یک کاربرد و یک عامل (Agent) جدید

۱-٤-١٤-١- هدف

می‌خواهیم یک کاربرد چند رسانه‌ای (Multimedia) بسازیم که روی یک ارتباط UDP اجرا شود و بتوانیم رفتار یک کاربرد چندرسانه‌ای را بصورت مجازی مشاهده و شبیه‌سازی کنیم.

۱-٤-١٤-٢- توصیف کاربرد

در این کاربرد، فرض شده است وقتی یک اتصال برقرار می‌شود، فرستنده و گیرنده روی ۵ مجموعهٔ مختلف از نحوهٔ کد کردن و ارسال، که آنها را با اعداد ۰ تا ۴ نشان می‌دهیم، اتفاق نظر دارند. برای راحتی فرض کنیم که هر یک از فرستنده یا گیرنده‌ها دارای نرخ دریافت و ارسال ثابتی هستند و طول بسته‌ها نیز یکسان است و به نحوهٔ کدکردن ربطی ندارند.

این کاربرد چند رسانه‌ای به این صورت کار می‌کند که در ابتدا فرستنده با نرخ ارسال صفر شروع می‌کند و سپس نرخ ارسال خودش را با مقداری که گیرنده به آن اطلاع می‌دهد. گیرنده مسئول تراکم شبکه و تعیین نرخ ارسال مناسب می‌باشد. برای نمایش تراکم شبکه، از یک بسته ساده که در هر RTT از آن استفاده می‌شود، کمک گرفته می‌شود. اگر تراکم در شبکه مشاهده شود آنگاه گیرنده مقدار پارامتر Scale را به نصف کاهش می‌دهد و این مقدار کاهش یافته را به اطلاع فرستنده نیز می‌رساند. اما اگر بسته ساده گم نشود آنگاه پارامتر Scale را یکی افزایش داده و این مقدار را به فرستنده نیز اطلاع می‌دهد.

۱-٤-١٤-٣- تحلیل مشکلات

قبل از پیاده‌سازی این کاربرد، کاربرد مربوط به عامل UDP آزمایش شده و یک مشکل اساسی مشاهده شد. از آنجائی که عامل UDP وظیفهٔ ارسال بسته‌های شبکه را برعهده دارد، تمام اطلاعات لازم برای انتقال، باید به صورت رشته‌ای از داده، در اختیار عامل UDP قرار بگیرید.

UDP فقط بسته‌هایی را که دارای پشته سرآمد هستند ملاحظه می‌کند، بنابراین باید پیاده‌سازی UDP را تغییر دهیم تا قادر باشیم که داده‌های دریافت شده از کاربردها را نیز اضافه کنیم. از آنجائیکه ما بعداً می‌خواهیم از این کاربرد در مکانیزم مدیریت صف مسیریاب IP نیز استفاده کنیم، بنابراین بدنبال راهی هستیم تا این رشته چندپخشی را از سایر رشته‌ها متمایز کنیم. برای این منظور باید عوامل UDP طوری تغییر کند که بتواند داده‌ها را در یکی از سرآمدهای IPای که مورد استفاده قرار نگرفته است، ذخیره کند.

۱-٤-١٤-٤- طراحی و پیاده‌سازی

برای این کاربرد از CBR و اعمال تغییراتی روی آن برای داشتن پنج سطح Scale (0-4) استفاده شده است. نام این کلاس از کاربرد را MmApp می‌گذاریم که یک کلاس فرزند از کلاس Application است. در شکلهای زیر مراحل طراحی و پیاده‌سازی نمایش داده شده است.

```
// Multimedia Header Structure
struct hdr_mm {
      int ack;      // is it ack packet?
      int seq;      // mm sequence number
      int nbytes;   // bytes for mm pkt
      double time;  // current time
      int scale;    // scale (0-4) associated with data rates

      // Packet header access functions
      static int offset_;
      inline static int& offset() { return offset_; }
      inline static hdr_mm* access(const Packet* p) {
              return (hdr_mm*) p->access(offset_);
      }
};
```

```
// Multimedia Header Class
static class MultimediaHeaderClass : public PacketHeaderClass {
public:
 MultimediaHeaderClass() : PacketHeaderClass("PacketHeader/Multimedia",
                                              sizeof(hdr_mm)) {
            bind_offset(&hdr_mm::offset_);
      }
} class_mmhdr;
```

شکل ۲۲: ساختار سرآمد MM و کلاس آن

```
enum packet_t {
        PT_TCP,
        ...
        PT_Multimedia,
        PT_NTYPE // This MUST be the LAST one
};

class p_info {
public:
        p_info() {
                name_[PT_TCP]= "tcp";
                ...
                name_[PT_Multimedia] = "Multimedia";
                name_[PT_NTYPE]= "undefined";
        }
        ...
};
```

شکل ۲۳: اضافه کردن به فایل packet که از نوع C++ است.

```
foreach prot {
        AODV
        ...
        Multimedia
} {
        add-packet-header $prot
}
```

شکل ۲۴: اضافه کردن به فایل ns-packet.tcl که از نوع OTCL است.

```
class SendTimer : public TimerHandler {
 public:
        SendTimer(MmApp* t) : TimerHandler(), t_(t) {}
        inline virtual void expire(Event*);
 protected:
        MmApp* t_;
};

void SendTimer::expire(Event*)
{
  t_->send_mm_pkt();
}
```

```
class MmApp : public Application {
public:
        MmApp();
        ...
private:
        ...
        SendTimer snd_timer_;
        ...
};
```

```
MmApp::MmApp() : running_(0), snd_timer_(this), ack_timer_(this)
{
  bind_bw("rate0_", &rate[0]);
  ...
  bind_bw("rate4_", &rate[4]);
  bind("pktsize_", &pktsize_);
  bind_bool("random_", &random_);
}
```

```
void MmApp::send_mm_pkt()
{
  hdr_mm mh_buf;

  if (running_) {
    ...
    agent_->sendmsg(pktsize_, (char*) &mh_buf);  // send to UDP

    // Reschedule the send_pkt timer
    double next_time_ = next_snd_time();
    if(next_time_ > 0) snd_timer_.resched(next_time_);
  }
}
```

شکل ۲۵: ارسال اجرای تایمر

```
class Agent : public Connector {
 public:
        Agent(int pktType);
        ...
        virtual int supportNM() { return 0; }
        virtual void enableNM() {}
        virtual void sendmsg(int nbytes, const char *flags = 0);
        virtual void send(int nbytes) { sendmsg(nbytes); }
        ...
}
```

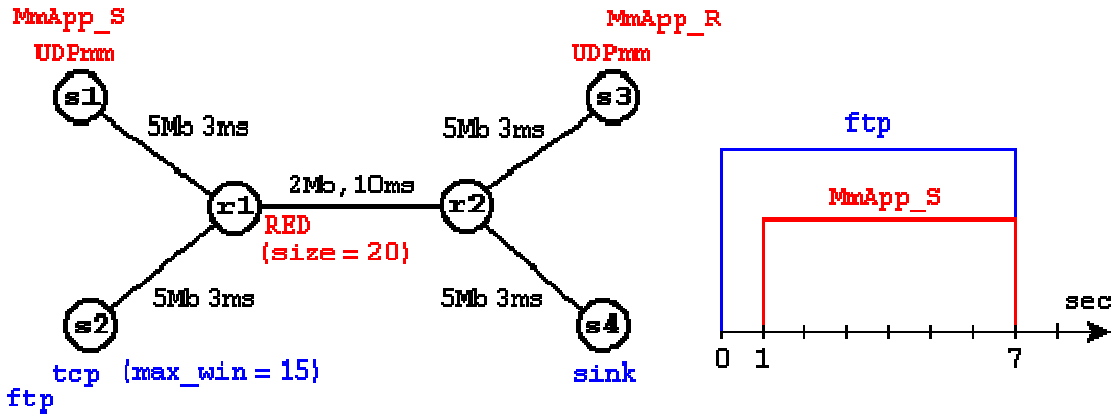
شکل ۲۶: افزودن دو تابع محلی به کلاس Agent

```
class Application : public Process {
public:
        Application();
        virtual void send(int nbytes);
        virtual void recv(int nbytes);
        virtual void recv_msg(int nbytes, const char *msg = 0){};
        virtual void resume();
        ...
};
```

شکل ۲۷: افزودن دو تابع محلی به کلاس Application

```
...
Application/MmApp set rate0_ 0.3mb
Application/MmApp set rate1_ 0.6mb
Application/MmApp set rate2_ 0.9mb
Application/MmApp set rate3_ 1.2mb
Application/MmApp set rate4_ 1.5mb

Application/MmApp set pktsize_ 1000
Application/MmApp set random_ false
...
```

شکل ۲۸: تنظیم مقادیر اولیه پارامترها



شکل ۲۹: توپولوژی و سناریوی شبیه‌سازی برنامه Test

۱-۴-۱۵- اضافه‌کردن یک صف جدید

۱-٤-۱۵-۱- هدف

اضافه کردن یک صف جدید برای ساخت صف خروجی مربوط به مسیریاب (router) drop-tail از خاصیت زمان‌بندی round-robin dequeue برای حق تقدم ۱۵ بسته موجود در یک صف استفاده می‌کنند. این وضعیت هنگامی اتفاق می‌افتد که ۱۵ بسته که دارای حق تقدم بیشتری هستند به همراه دیگر بسته‌ها در یک صف قرار دارد، با این کار قدیمی‌ترین بسته از هر نوع با اولویت، به ترتیب از صف خارج می‌شوند.

۱-٤-۱۵-۲- طراحی

صف مورد نظر دارای دو صف از نوع FIFO است که آنها را LQ1 و LQ2 می‌نامند. اندازهٔ کلی این صفها مساوی اندازهٔ صف فیزیکی (PQ) است. بعبارت دیگر LQ1+LQ2=PQ می‌باشد. برای ایجاد رفتار متناسب با صف drop-tail، وقتی یک بسته وارد صف می‌شود (enqueue)، مدیریت صف چک می‌کند که آیا اندازه LQ1+LQ2 از حداکثر مجاز که PQ است، کمتر می‌باشد یا خیر. اگر این اندازه کمتر بود، این بسته وارد صف مناسب خواهد شد. برای انجام عملیات خارج کردن بسته‌ها از صف (dequeue) بر اساس زمان‌بندی round-robin، مدیریت صف سعی می‌کند یک بسته را از یکی از صفهای LQ1 یا LQ2 خارج کند و بستهٔ دیگر را از صف دیگر خارج نماید. بعبارت دیگر بسته‌های موجود در صف‌های LQ1 و LQ2 با نرخ یک به یک خارج می‌شوند و این عمل تا هنگامی که هر دو صف دارای بسته باشند ادامه پیدا خواهد کرد.

۱-٤-۱۵-۳- پیاده‌سازی

این شیئ صف DtRrQueue (Drop-Tail Round-Robin Queue) با یک اسم C++ نامگذاری می‌شود که از کلاس Queue اخذ شده است. نام OTCL معادل آن Queue/DTRR است. هنگامی که تابع recv که در کلاس Queue اجرا شده است، یک بسته دریافت می‌کند، تابع enqueue را صدا می‌زند و اگر شیئ اتصال مسدود نشده باشد تابع dequeue را نیز صدا می‌کند. بنابراین مجبور هستیم توابع enqueue و dequeue را برای کلاس DtRrQueue بنویسیم. شکل ۲۹ تعریف کلاس DtRrQueue و توابع آنرا نشان می‌دهد.

```
class DtRrQueue : public Queue {
 public:
         DtRrQueue() {
                q1_ = new PacketQueue;
                q2_ = new PacketQueue;
                pq_ = q1_;
                deq_turn_ = 1;
        }

 protected:
         void enque(Packet*);
         Packet* deque();

         PacketQueue *q1_;    // First  FIFO queue
         PacketQueue *q2_;    // Second FIFO queue
         int deq_turn_;       // 1 for First queue 2 for Second
};
```

```
void DtRrQueue::enque(Packet* p)
{
  hdr_ip* iph = hdr_ip::access(p);

  // if IPv6 priority = 15 enqueue to queue1
  if (iph->prio_ == 15) {
    q1_->enque(p);
    if ((q1_->length() + q2_->length()) > qlim_) {
      q1_->remove(p);
      drop(p);
    }
  }
  else {
    q2_->enque(p);
    if ((q1_->length() + q2_->length()) > qlim_) {
      q2_->remove(p);
      drop(p);
    }
  }
}
```

```
Packet* DtRrQueue::deque()
{
  Packet *p;

  if (deq_turn_ == 1) {
        p =  q1_->deque();
        if (p == 0) {
            p = q2_->deque();
            deq_turn_ = 1;
        }
        else
            deq_turn_ = 2;
  }
  else {
        p =  q2_->deque();
        if (p == 0) {
            p = q1_->deque();
            deq_turn_ = 2;
        }
        else
            deq_turn_ = 1;
  }

  return (p);
}
```

شکل ۳۰: پیاده‌سازی کلاس DtRr Queue

۱-٤-۱۵-٤- آزمایش

در این قسمت از برنامهٔ شبیه‌سازی برای آزمایش MmApp روی UDPmm که در قسمت قبل آمده است استفاده می‌کنیم، فقط صف RED را به صف DRTT تغییر می‌دهیم که این صف مربوط به لینک بین r1-r2 است. تغییرات اعمال شده در شکل ۳۱ آمده است.

```
Set ns [new Simulator]
...
$ns duplex-link $node_(s1) $node_(r1) 5Mb 3ms DropTail
$ns duplex-link $node_(s2) $node_(r1) 5Mb 3ms DropTail
$ns duplex-link $node_(r1) $node_(r2) 2Mb 10ms DTRR
$ns duplex-link $node_(s3) $node_(r2) 5Mb 3ms DropTail
$ns duplex-link $node_(s4) $node_(r2) 5Mb 3ms DropTail

#Set DTRR queue size to 20
$ns queue-limit $node_(r1) $node_(r2) 20
...
#Simulation Scenario
$ns at 0.0 "$ftp start"
$ns at 1.0 "$mmapp_s start"
$ns at 7.0 "finish"

$ns run
```

شکل ۳۱: برنامه آزمایش Dr Rr Queue

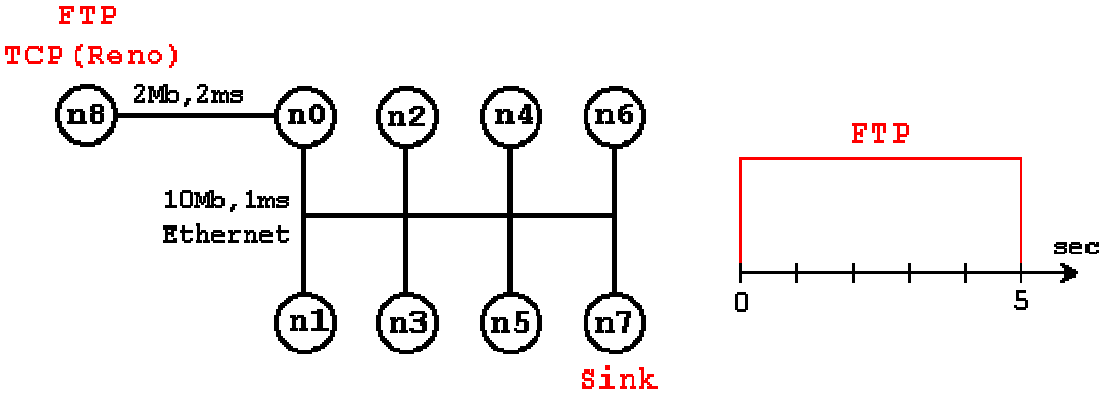
۱-۴-۱۶- مثال LAN

این بخش شامل مثالی از برنامهٔ شبیه‌سازی مربوط به LAN است. توپولوژی این شبکه و مراحل شبیه‌سازی آن به نمایش درآمده است. برنامهٔ زیر را اجرا کنید و نتایج آنرا ملاحظه کنید.

```
 #Author: Jae Chung
 #Date:   7/20/99
#
 #This file is modified from
" #ns-2/tcl/ex/lantest.tcl"

set opt(tr)    "out.tr"
set opt(namtr) "out.nam"
set opt(seed)  ۰
set opt(stop)  ۵
set opt(node)  ۸

set opt(qsize) ۱۰۰
set opt(bw)    ۱۰Mb
set opt(delay) ۱ms
set opt(ll)    LL
set opt(ifq)   Queue/DropTail
```

```
set opt(mac)   Mac/Csma/Ca
set opt(chan)  Channel
set opt(tcp)   TCP/Reno
set opt(sink)  TCPSink

set opt(app)   FTP


proc finish {} {
     global ns opt trfd ntrfd

     $ns flush-trace
     close $trfd
        close $ntrfd
     exec nam $opt(namtr)&
     exit 0
}

proc create-trace {} {
     global ns opt

     set trfd [open $opt(tr) w]
     $ns trace-all $trfd
     return $trfd

}
proc create-namtrace {} {
        global ns opt

        set ntrfd [open $opt(namtr) w]
$        ns namtrace-all $ntrfd
}

proc create-topology {} {
     global ns opt
     global lan node source node0

     set num $opt(node(
     for {set i 0} {$i < $num} {incr i} {
          set node($i) [$ns node[
          lappend nodelist $node($i)
     }

     set lan [$ns newLan $nodelist $opt(bw) $opt(delay)\
              -llType $opt(ll) -ifqType $opt(ifq)\
              -macType $opt(mac) -chanType $opt(chan)]

     set node0 [$ns node]
     $ns duplex-link $node0 $node(0) 2Mb 2ms DropTail

     $ns duplex-link-op $node0 $node(0) orient right

}

 ##MAIN##

set ns [new Simulator]
set trfd [create-trace]
```

```
set ntrfd [create-namtrace]

create-topology

set tcp0 [$ns create-connection TCP/Reno $node0 TCPSink $node(7) 0]
$tcp0 set window_ 15

set ftp0 [$tcp0 attach-app FTP]

$ns at 0.0 "$ftp0 start"
$ns at $opt(stop) "finish"

$ns run
```



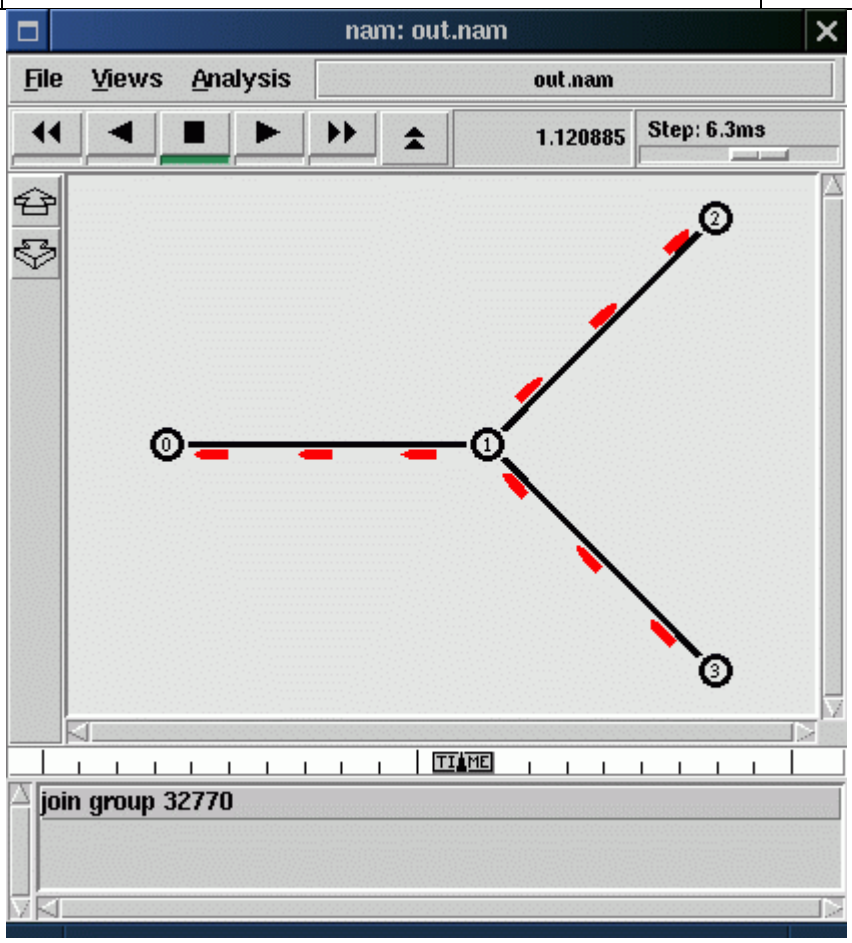
شکل ۳۲: توپولوژی شبکهٔ LAN و شبیه‌سازی آن

۱-۴-۱۷- مثال چندپخشی (Multicasting)

این قسمت شامل مثالی از شبیه‌سازی عملیات ارسال بصورت چندپخشی می‌باشد که متن برنامه و نتایج NAM آن را مشاهده خواهید کرد. برنامهٔ زیر را با اسم ex.mcast.tcl ذخیره و اجرا کنید. این برنامه از پنجمین خودآموز VINT/NS آورده شده است.

```
 #Polly Huang 8-7-98

set ns [new Simulator]
$ns multicast

set f [open out.tr w]
$ns trace-all $f
$ns namtrace-all [open out.nam w]

$ns color 1 red
 #prune/graft packets
$ns color 30 purple
$ns color 31 green

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
```

```
set n3 [$ns node]

 #Use automatic layout
$ns duplex-link $n0 $n1 1.5Mb 10ms DropTail
$ns duplex-link $n1 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n1 $n3 1.5Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n1 $n3 orient right-down
$ns duplex-link-op $n0 $n1 queuePos 0.5

set mrthandle [$ns mrtproto DM{} ]

set cbr0 [new Application/Traffic/CBR]
set udp0 [new Agent/UDP]
$cbr0 attach-agent $udp0
$ns attach-agent $n1 $udp0
$udp0 set dst_ 0x8001

set cbr1 [new Application/Traffic/CBR]
set udp1 [new Agent/UDP]
$cbr1 attach-agent $udp1
$udp1 set dst_ 0x8002
$udp1 set class_ 1
$ns attach-agent $n3 $udp1

set rcvr [new Agent/LossMonitor]
$#ns attach-agent $n3 $rcvr
$ns at 1.2 "$n2 join-group $rcvr 0x8002"
$ns at 1.25 "$n2 leave-group $rcvr 0x8002"
$ns at 1.3 "$n2 join-group $rcvr 0x8002"
$ns at 1.35 "$n2 join-group $rcvr 0x8001"

$ns at 1.0 "$cbr0 start"
$ns at 1.1 "$cbr1 start"

$ns at 2.0 "finish"

proc finish {} {
     global ns
     $ns flush-trace

     puts "running nam"...
     exec nam out.nam&
     exit 0
}

$ns run
```



شکل ۳۳: صفحهٔ NAM مربوط به شبیه‌سازی ارسال چندپخشی

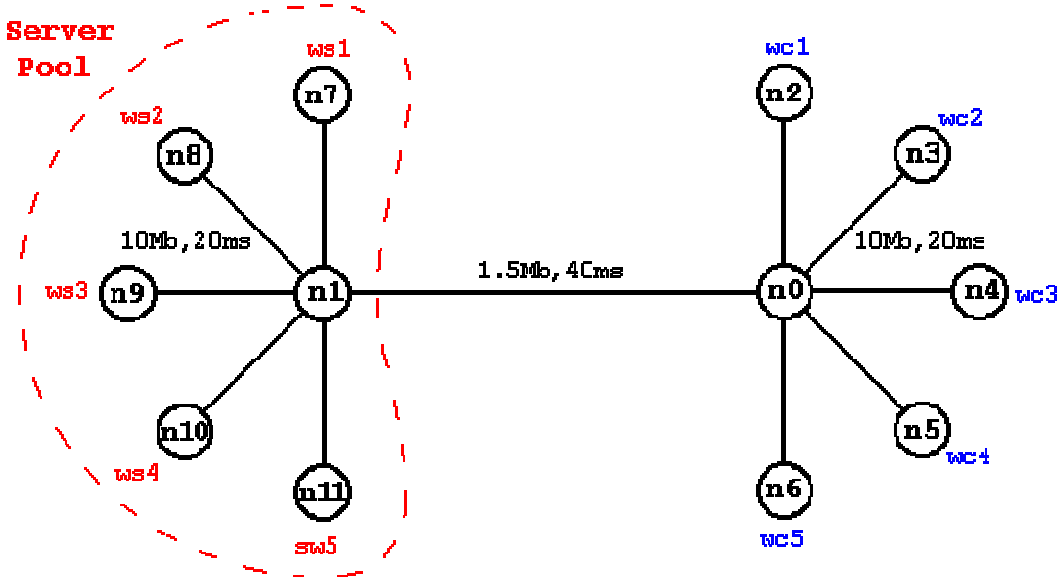
۱-۴-۱۸- مثال وب سرور

برنامهٔ مربوط به شبیه‌سازی web server، توپولوژی ونتایج مربوط به آن در این قسمت آورده شده است. برنامه‌های زیر را با اسم‌های ex.web.tcl و dumbbell.tcl ذخیره و اجرا کنید و نتایج آنها را ملاحظه کنید. این مثالها هم از همچنین خودآموز NS آورده شده‌اند. توجه کنید که باید در ابتدای برنامه مسیر مربوط به http-mod.tcl را تغییر دهید تا این برنامه روی کامپیوتر شما اجرا شود.

```
 #Polly Huang 8-7-98

 #Initial setup
source ~/ns-allinone-2.1b4a/ns-2/tcl/http/http-mod.tcl
source dumbbell.tcl
global num_node n

set ns [new Simulator[
$ns set-address 7 24      ;# set-address <bits for node address> <bits for
port>
```

```
 #set up colors for nam
for {set i 1} {$i <= 30} {incr i} {
    set color [expr $i % 6]
    if {$color == 0} {
     $ns color $i blue
      {elseif {$color == 1} {
     $ns color $i red
      {elseif {$color == 2} {
     $ns color $i green
      {elseif {$color == 3} {
     $ns color $i yellow
      {elseif {$color == 4} {
     $ns color $i brown
      {elseif {$color == 5} {
     $ns color $i black
    }
}

 #Create nam trace and generic packet trace
$ns namtrace-all [open out.nam w[
 #trace-all is the generic trace we've been using
$ns trace-all [open out.tr w[

create_topology

 ##########################Modify From Here#####################
 ##Number of Pages per Session
set numPage 10
set httpSession1 [new HttpSession $ns $numPage [$ns picksrc[[
set httpSession2 [new HttpSession $ns $numPage [$ns picksrc[[

 ##Inter-Page Interval
 ##Number of Objects per Page
 ##Inter-Object Interval
 ##Number of Packets per Object
 ##have to set page specific attributes before createPage
 ##have to set object specific attributes after createPage
$httpSession1 setDistribution interPage_ Exponential 1 ;#in sec
$httpSession1 setDistribution pageSize_ Constant 1 ;# number of
objects/page
$httpSession1 createPage
$httpSession1 setDistribution interObject_ Exponential 0.01 ;# in sec
$httpSession1 setDistribution objectSize_ ParetoII 10 1.2 ;# number of
packets

 #uses default
$httpSession2 createPage

$ns at 0.1 "$httpSession1 start" ;# in sec as well
$ns at 0.2 "$httpSession2 start"

$ns at 30.0 "finish"

proc finish {} {
     global ns
     $ns flush-trace
     puts "running nam"...
 #    exec to run unix command
```

```
     exec nam out.nam&
     exit 0
}

 #Start the simualtion
$ns run
```



شکل ۳۴: توپولوژی شبکه web server و شبیه‌سازی آن

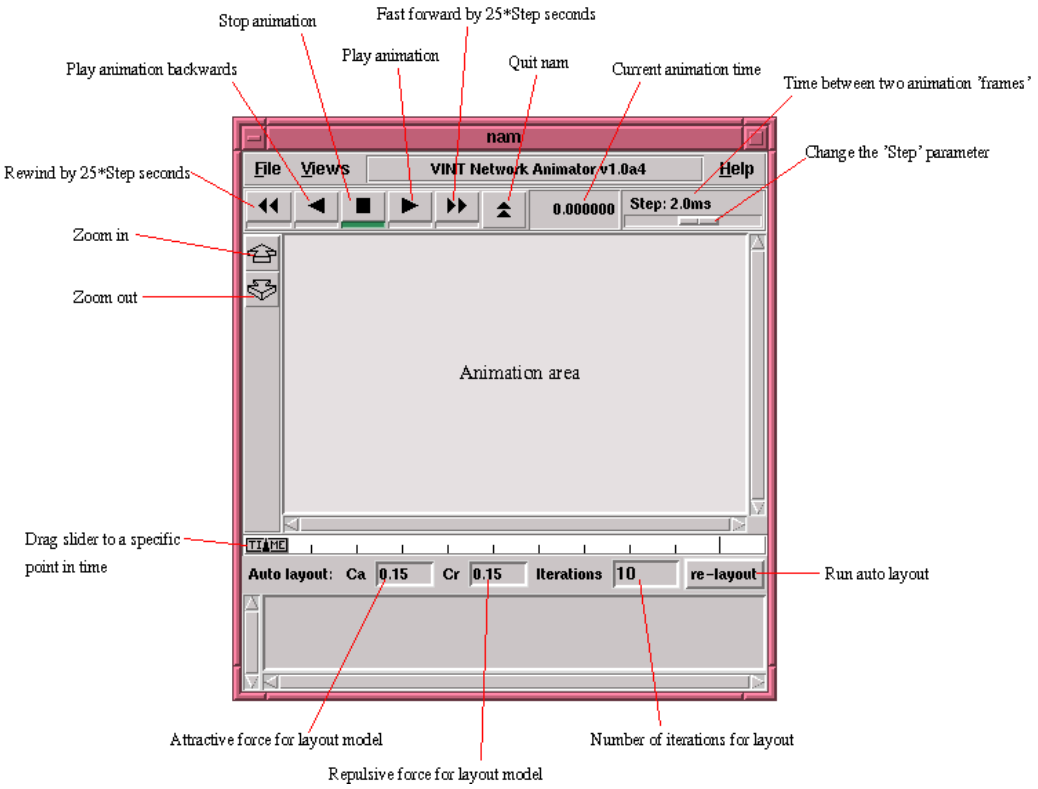
```
 #Created by Polly Huang
 #Simple 4-node star topology
#          8   7       2   3
#            \ |       | /
#          9 --1 ----- 0 - 4
#            / |       | \
#          10  11      6   5

proc create_topology {} {
global ns n num_node
set num_node 12

for {set i 0} {$i < $num_node} {incr i} {
     set n($i) [$ns node[
}

$ns set src_ [list 2 3 4 5 6]
$ns set dst_ [list 7 8 9 10 11]

 #EDGES (from-node to-node length a b):
$ns duplex-link $n(0) $n(1) 1.5Mb 40ms DropTail
$ns duplex-link $n(0) $n(2) 10Mb 20ms DropTail
$ns duplex-link $n(0) $n(3) 10Mb 20ms DropTail
$ns duplex-link $n(0) $n(4) 10Mb 20ms DropTail
```

```
$ns duplex-link $n(0) $n(5) 10Mb 20ms DropTail
$ns duplex-link $n(0) $n(6) 10Mb 20ms DropTail
$ns duplex-link $n(1) $n(7) 10Mb 20ms DropTail
$ns duplex-link $n(1) $n(8) 10Mb 20ms DropTail
$ns duplex-link $n(1) $n(9) 10Mb 20ms DropTail
$ns duplex-link $n(1) $n(10) 10Mb 20ms DropTail
$ns duplex-link $n(1) $n(11) 10Mb 20ms DropTail

$ns duplex-link-op $n(0) $n(1) orient left
$ns duplex-link-op $n(0) $n(2) orient up
$ns duplex-link-op $n(0) $n(3) orient right-up
$ns duplex-link-op $n(0) $n(4) orient right
$ns duplex-link-op $n(0) $n(5) orient right-down
$ns duplex-link-op $n(0) $n(6) orient down
$ns duplex-link-op $n(1) $n(7) orient up
$ns duplex-link-op $n(1) $n(8) orient left-up
$ns duplex-link-op $n(1) $n(9) orient left
$ns duplex-link-op $n(1) $n(10) orient left-down
$ns duplex-link-op $n(1) $n(11) orient down
}
 #end of create_topology
```

۱-۵- معرفی برنامه NAM

Nam یک TCL/TK بر مبنای ابزار animation برای دیدن ترسیم شبیه‌سازی و دنیای واقعی پیگیری بسته‌های داده می‌باشد. قدم اول در استفاده از nam، ایجاد فایل trace می‌باشد. این فایل باید شامل اطلاعات توپولوژی گره‌ها، لینک‌ها و packet trace باشد. غالباً فایل trace بوسیله ns تولید می‌شود. در طول شبیه‌سازی ns، کاربران می‌توانند شکل توپولوژی، اطلاعات طرح‌بندی و packet trace را با استفاده از وقایع فایل trace در ns تولید کنند. هنگامیکه فایل trace تولید می‌شود، آماده نمایش از طریق nam می‌باشد. به محض شروع trace ، برنامه nam فایل را می‌خواند و توپولوژی را ایجاد می‌کند. سپس پنجره را pop up می‌کند و اگر لازم باشد طرح‌ریزی می‌کند و تا زمان رسیدن اولین بسته در فایل trace صبر می‌کند

Nam یک نمایش دهندهٔ نموداری توپولوژی شبکه و جریان داده است. با استفاده از این ابزار می‌توان مسیر تک تک بسته ها و حتی نوع و طول آنها را در حین عملکرد سیستم مشاهده کرد.



شکل ۳۵: معرفی محیط ابزار Network Animator

شکل ۳۵ محیط ابزار Nam را نشان می‌دهد. با این نرم افزار می‌توان علاوه بر نمایش نتیجهٔ شبیه‌سازی، توپولوژی های ساده را نیز تولید نمود. ابزارهای بهتری نیز نظیر Nscript برای تولید توپولوژی و ترافیک شبکه وجود دارند که از GUI[1] مناسبی برخوردار هستند و تا حد زیادی کاربر را از نوشتن کدهای Tcl بی‌نیاز می‌کنند. با این وجود کُد Tcl کارامدترین روش پیاده سازی شبکه های پیچیده بوده و امکانات بیشتری را در اختیار کاربر قرار می‌دهد. شکل زیر یک نمونه کُدِ Script و تصویری از نمایشِ در حال جریانِ NAM از این ارسال دادهٔ ساده را نمایش می‌دهد.

در NS امکان تعریف پهنای باند، تاخیر، نوع پروتکل لایهٔ Data link (مانند ARQ)، نوع مدیریت صف برای هر خط و نوع آدرس دهی (MAC) و نحوهٔ ارسال داده روی خط مشترک (CSMA/CD) نیز وجود دارد. امکان اعمال رویداد در حین شبیه‌سازی و پویا نمودن شبیه سازی نیز وجود دارد. برای مثال می‌توان یک اتصال را در لحظه ای مشخص قطع نمود و یا یک خط lossy تعریف نمود.

LAN و Wireless LAN در NS به عنوان یک نود در نظر گرفته می‌شوند، چرا که در توپولوژی LAN تمامی رایانه ها به یک خط واحد متصل هستند که می‌توان آنرا یک نود میانی تصور نمود. بنابراین نوع

[1] Graphical User Interface

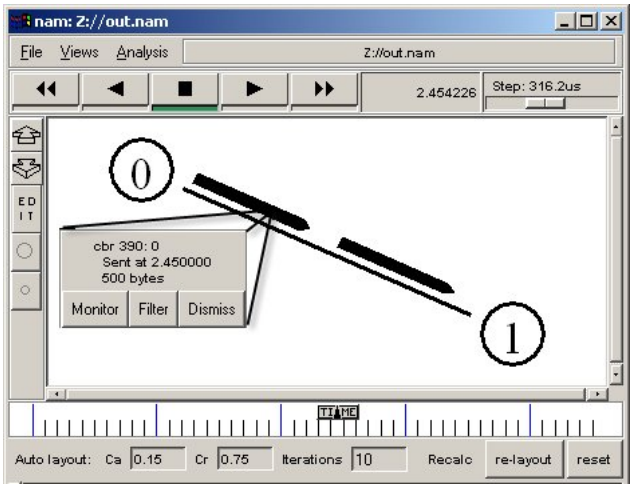
دیگری از نود بنام LAN node در NS تعریف شده است که امکان شبیه سازی شبکه های محلی را فراهم می‌کند.

مسیریابی (Routing) بطور پیش فرض توسط NS انجام می‌شود. NS برای آدرس دهی و راهیابی در شبکه از یک سیستم تعیین آدرسِ خاص استفاده می‌کند. آدرس در NS بطور پیش فرض یک رشتهٔ ۶۴ بیتی است که ۳۲ بیت با ارزش کمتر[۱] آن معرف port-id و از ۳۲ بیت نیمهٔ بالاتر، بیت با ارزش بالاتر برای مقصود Multicasting و ۳۱ بیت بقیه برای مشخص نمودن node-id بکار می‌رود. کاربر معمولاً با این آدرس دهی درونی سر و کار ندارد چراکه نام نود ها، معرف آدرس آنهاست.

[۱] Least Significant Bits

loaded duplex connection Tcl Code –

```
#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 $nf
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

#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

# Create a CBR traffic and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize  500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

#Create a traffic sink and attach it to node n1
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0

#Connect the traffic source to the traffic sink
$ns connect $udp0 $null0

#Schedule events for the CBR agent
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of
#simulation time
$ns at 5.0 "finish"

#Run the simulation
$ns run
```



شکل ۳۶: خروجی Tcl Code در Nam . بسته ها بصورت پیکانهایی نمایش داده می-شوند

xGRAPH وسیله رسم نمودار در NS است. این ابزار همانند Nam بر روی فایل خروجی حاصل از شبیه سازی عمل می‌کند. Xgraph با استفاده از فایل trace تولید شده توسط NS قادر است مقادیر عددی را به شکل نموداری نشان دهد. این ابزار تنها برای نسخهٔ Linux، تهیه شده و کاربران Windows باید از ابزار وابسته به MATLAB ای بنام Gnuplot و یا Tracegraph استفاده نمایند.

۱-۶- دستورات اولیه در NS2

دستوراتی که در این بخش می‌اید باید توسط ویرایشگر pseditor در یک فایل وارد شود و در انتها با پسوند tcl ذخیره شود. برای شبیه‌سازی، در ابتدا باید یک نمونه از شی simulator ایجاد گردد. این کار با دستور زیر انجام می‌گیرد.

```
Set ns [new simulator  ]
```

برای ذخیره اطلاعات مورد استفاده از namtrace باید فایلی باز شود. برای این کار از دستور زیر استفاده می‌شود.

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

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

برای اینکه تمام خروجی‌ای مرتبط با nam به فایل nf منتقل شود از دستور زیر استفاده می‌شود.

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

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

```
proc finish {}{
global nf ns
$ns flush-trace
close $nf
exec nam out.nam
exit 0
}
```

در این تابع، پس از خاتمه شبیه‌سازی و بستن فایل out.nam برنامه nam با ورودی out.nam صدا زده می‌شود. برای خاتمه شبیه‌سازی باید تابه finish صدا شود.

```
$ns at 5.0 “finish”
```

همانطور که دیده می‌شود برای زمانبندی رویدادها از کلمه کلیدی at می‌توان استفاده کرد.

در انتها برای شروع شبیه‌سازی از دستور زیر استفاده می‌شود.

```
$ns run
```

۱-۶-۱-۱- مدل کردن یک توپولوژی ساده: دو گره با یک لینک

برای شروع از یک توپولوژی ساده که در آن دو گره بایک لینک به هم متصل شده‌اند استفاده می‌شود. کد این دستورات باید فبل از $ns run یا حتی بهتر است قبل از $ns at 5.0 “finsh” اضافه شوند.

در ابتدا باید گره*ها تعریف کرد:*

```
set n0 [$ns node]
```

```
set n1 [$ns node]
```

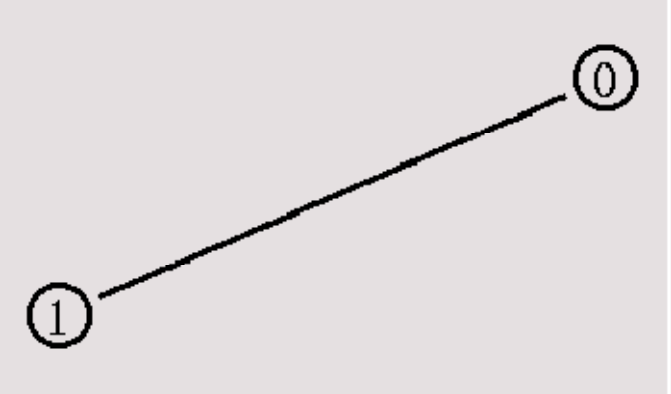
این دو دستور، گره‌ا را ایجاد کرده و آنها را به n0 و n1 نسبت می‌دهند.

برای تعریف لینک بین دو گره از دستور زیر استفاده می‌شود:

```
$ns duplex-link $n0 $n1 1mb 10ms DropTail
```

این دستور یک لینک بین دو گره n0 و n1 ایجاد می‌کند که از نوع دو طرفه می‌باشد. پهنای باند $1\ \frac{mbit}{s}$ و تاخیر 10ms داشته و دارای صفی از نوع DropTail است. در صف DropTail در صورت ارسال ترافیک بیش از ظرفیت خط، بسته‌ها از انتهای صف حذف خواهند شد. با انواع دیگر صف بعدا اشنا خواهیم شد.

حال این دستورات ذخیره کرده و برنامه را اجرا کنید. خروجی به صورت زیر مشاهده می‌شود:



در این بخش می‌خواهیم ترافیکی را از n0 به n1 ارسال کنیم.

در ns برا یارسال و دریافت داده‌ها از عاملها استفاده می‌شود. برای منبع ترافیک۱ و مقصد ترافیک۲ عامل‌های جداگانه‌ایی تعریف می‌شود.

```
#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
#Create a CBR traffic source and attach it to udp0
set cbr0 [new Aplication/Traffic/CBR]
$cbr0 set packetsize_500
$cbr0 set interval_0.005
$cbr0 attach-agent $udp0
```

سه خط اول سبب ایجاد یک UDP agnet و نسبت دادن آن به n0 می‌شود. خطوط بعد یک منبع ترافیک CBR ایجاد کرده و به udp0 نسبت می‌دهد. اندازه بسته‌ها ۵۰۰ بایت بوده و هر ۰.۰۰۵ ثانیه یک بسته ارسال می‌شود.

دو دستور بعد یک عامل null که به عنوان گیرنده ترافیک عمل می‌کند ایجاد کرده و ان را به n1 نسبت می‌دهند.

---

[1] Traffic Source

[2] Traffic Sink

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

حال دو عامل به هم متصل می‌شوند تا ترافیک ارسالی توسط n0 در n1 دریافت شود:

```
$ns connect $udp0 $null0
```

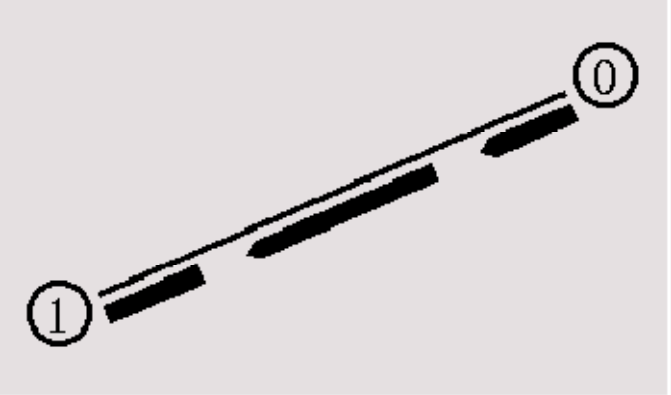
شاید این سئوال مطرح شود که ترافیک چه زمانی شروع به ارسال شده و چه زمانی متوقف شود؟

پاسخ این سئوال در زیر داده شده است.

برای کنترل منبع ترافیک از دستورات زیر استفاده می کنیم. قبل از ($ns at 5.0 finish)

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

حال می توانیم شبیه سازی را انجام دهیم . وقتی در پنجرهٔ باز شده توسط nam بر روی دکمهٔ play کلیک کنیم مشاهده می شود در ثانیه ۰.۵ گره n0 شروع به ارسال ترافیک برای گره n1 می‌کند. در ثانیه ۴/۵ ارسال ترافیک موقف می شود .شکل زیر ارسال ترافیک از گره n0 به گره n1 را نشان می‌دهد.



می توانید با کلیک روی لینک اطلاعات آماری را بدست آورید . همین طور می توانید اندازهٔ بسته ها و زمان بین ارسال بسته ها[1] را تغییر داده و حاصل تغییرات را مشاهده کنید.

۱-۶-۱-۲- مشخص کردن جریانهای ترافیک و پایش کردن لینکها

در این بخش از یک توپولوژی با ۴ گره استفاده خواهیم کرد . دو گره به عنوان منبع ترافیک عمل می کنند .یک گره دریافت کننده ترافیک می باشد .و یک گره فقط جریانهای ترافیک را به مقصد می فرستد. برای مشخص کردن جریانهای ترافیک و مونیتور کردن صف مربوط به لینک روشهایی ارایه خواهد شد

[1] Interval Time

۱-٦-۱-۳- تعریف توپولوژی

حال برای تعریف گره ها خطوط زیر را اضافه کنید:

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

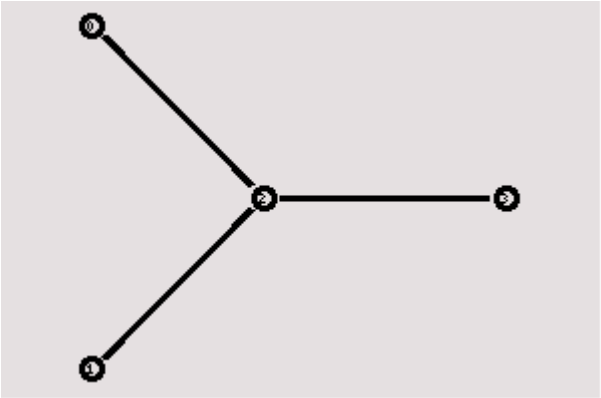
برای تعریف لینک بین گره ها کدهای زیر را بیفزایید:

```
$ns duplex-link $n0 $n2 1mb 10ms DropTail
$ns duplex-link $n1 $n2 1mb 10ms DropTail
$ns duplex-link $n3 $n2 1mb 10ms DropTail
```

اکنون فایل را ذخیره و اجرا کنید . البته ممکن است شکل آن دلخواه نباشد . می توانید دکمه re-layout را بزنید تا شکل آن بهتر شود . اما بهتر است کنترل بیشتری روی آن داشته باشید . برای این منظور از پیکربندی مستقیم ها می توان استفاده کرد.

```
$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 righ
```

با افزودن این دستورات ها شکل توپولوژی در nam بصورت زیر خواهد شد.



در دستورهای بالاجهت لینک ها را با right,left,up,down و یا ترکیبی از اینها می توان مشخص کرد.

۱-٦-۱-٤- تعریف منابع ترافیک و گیرنده های ترافیک

گره های n0 و n1 را بعنوان منابع ترافیک CBR و گره n3 را بعنوان گیرنده ترافیک تعریف می کنیم. گره n2 تنها کار ارسال را برای جریانهای ترافیک انجام می دهد.

```
#create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
#create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetsize_500
```

```
$cbr0 set interval _0.005
$cbr0 attach-gent $udp0
#create a UDP agent and attach it to noder n1
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
#create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetsize_500
$cbr1 set interval _0.005
$cbr1 attach-gent $udp1
#create a null aget for traffic sink n3
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
```

حال باید اتصال عاملها برقرار گردد

```
$ns connect $udp0 $null0
$ns connect $udp1 $null0
```

می خواهیم منبع ترافیک cbr0 در ثانیه ۰.۵ شروع به ارسال ترافیک کرده و در ثانیه ۴ ارسال ترافیک را متوقف کند و منبع ترافیک cbr1 در ثانیه 1 شروع به ارسال ترافیک کرده و در ثانیه ۴/۵ ارسال ترافیک را متوقف کند .بنابراین به صورت زیر عمل میکنیم.

```
$ns at 0.5 $cbr0 start
$ns at 1.0 $cbr1 start
$ns at 4.0 $cbr0 stop
$ns at 4.5 $cbr1 stop
```

با اجرای این اسکریپت ، مت وجه می شوید که مجموع دو جریان ترافیک ، بیش از ظرفیت خط بین n2 و n3 بوده و سبب حذف شدن برخی از بسته ها می شود .یک محاسبه ساده این مطلب را تایید می کند.

اندازه هر بسته ۵۰۰ بایت بوده و در هر ثانیه ۲۰۰ بسته ارسال می شود . پس هر جریان ترافیک پهنای باند ۰.۸ مگا بیت در ثانیه نیاز دارد . و خط ۱ مگابیت در ثانیه بین n2 و n3 جوابگوی ترافیک ۱/۶ مگا بیت در ثانیه ارسالی برای ان نیست.

پس بخشی از بسته ها حذف می شوند . اما کدام بسته ها ؟ چون رنگ تمام بسته ها مشکی است تمایز بین آنها امکان پذیر نمی باشد .برای ایجاد تمایز بین جر یانهای ترافیک در nam، از دستورات پیکربندی خاصی استفاده می شود .این دستورات در بخش بعد توضیح داده شدهاند.

۱-۶-۱-۵- مشخص کردن جریانهای ترافیک

دو خط زیر را به اسکریپت خود اضافه کنید:

```
$ns color 1 blue
$ns color 2 red
```

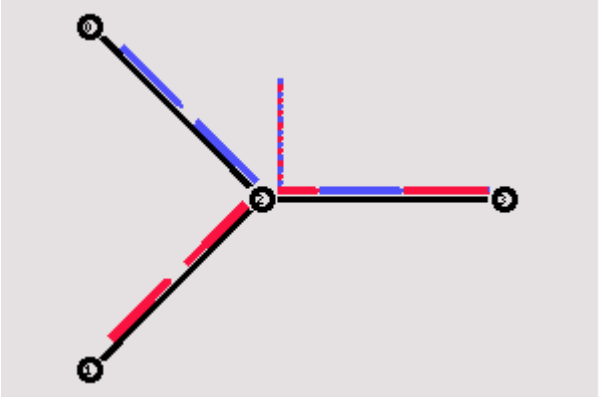
اگر اسکریپت را دوباره اجرا کنید، متوجه می شوید که جریان ترافیک n0 آبی رنگ و جریان ترافیک n1 قرمز رنگ می باشد . اگر لینک بین گره های n2 و n3 را مشاهده کنی د خواهید دید که بعد از مدتی نحوهٔ حذف بستهها زیاد عادلانه نیست.

۱-۶-۱-۶- مونیتور کردن یک لینک

برای مونیتور کردن صف مربوط به لینک بین n2 و n3 دستور زیر را به اسکریپت خود بیفزایید.

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

Ns را دوباره اجرا کنید .پس از چند لحظه تصویری مشابه تصویر زیر مشاهده خواهید کرد.



شکل ۳۷: مونیتور کردن یک لینک

اکنون می توانید قرار گرفتن بسته ها را در صف ببینید و بسته هایی که حذف می شوند را مشاهده کنید همانگونه که قبلا گفتیم نحوهٔ حذف بسته‌ها عادلانه نیست.

چون صف از نوع DropTail است بنا براین نمی توان انتظار داشت که عدالت را در حذف بسته ها رعایت کند .برای اینکه با هر دو نوع بسته به صورتی یکسان برخورد شود می توانید از صف SFQ استفاده کنید.

به این منظور تعریف لینک بین n2 و n3 را به این صورت تغییر دهید.

```
$ns duplex-link $n3 $n2 1Mb 10ms SFQ
```

صف بندی باید اکنون عادلانه باشد .بسته های آبی و قرمز به یک میزان حذف میشوند..

معرفی شبکه‌های VANET

۲-۱- مقدمه

۲-۲- بررسی شبکه‌های VANET

شبکه‌های VANET نوعی از شبکه های Ad_hoc هستند که ارتباط بین وسایل نقلیه مجاور، همچنین بین وسایل نقلیه و تجهیزات ثابتی که معمولا کنار جاده‌ها نصب می‌شوند را فراهم می‌آورند، هدف اصلی در شبکه‌های vanet ایجاد راحتی و امنیت بیشتر برای مسافران می‌باشد. به این منظور یک وسیله الکترونیکی در هر وسیله نقلیه نصب می‌شود که اتصال به شبکه Ad_hoc را برای مسافران فراهم می‌آورد، به این ترتیب هر وسیله نقلیه‌ای که با تجهیزات vanet مجهز شده است بعنوان یک نود در شبکه Ad_hoc عمل می‌کند و می‌تواند پیغام‌های دیگران را از طریق شبکه بی‌سیم دریافت و یا به نودهای دیگر پیغام ارسال نماید، این پیغام‌ها بیشتر برای اهداف امنیتی و کنترل ترافیک به‌کار می‌روند از جمله می‌توان به موارد زیر اشاره نمود :

- هشدار تصادف
- اعلان علائم جاده‌ای
- مشاهده ترافیک
- پرداخت عوارض راهداری
- پرداخت هزینه پارکینگ
- تفریحات و سرگرمی
- سایر موارد

این پیغام‌ها برای رانندگان ابزارهای مناسبی جهت تصمیم‌گیری در خصوص مسیر مناسب ارائه می‌دهند. علاوه بر این امکانات چندرسانه‌ای و اینترنت نیز برای مسافران تعبیه شده است. پرداخت عوارض راهداری و هزینه پارکینگ نیز از سایر خدمات این نوع شبکه‌هاست. امروزه کارخانه‌های بزرگ خودروسازی از جمله تویوتا و BMW پروژه‌های مختلفی را در خصوص این نوع شبکه‌ها و تجهیز خودروهای خود به قابلیت‌های شبکه‌های VANET در دستور کار دارند. شبکه های بیسیم تک کاره وسایل نقلیه (VANETs) یک گروه فوق العاده چالشی از شبکه های بیسیم تک کاره تغییرپذیر (MANETs) هستند که در حال حاضر توجه گسترده تحقیقات در زمینه شبکه سازی بیسیم و همچنین صنایع اتومبیل سازی را به خود جلب کرده اند.

برقراری ارتباط در MANETs به برنامه های کاربردی متنوع و پراکنده در میان گره های سیار در محیطهای فاقد ساختار زیربنایی کمک میکند. به علت داشتن قابلیت تغییرپذیری نسبتاً زیاد ، برقراری ارتباط در VANETs چالشهای قویتری را نسبت به ارتباطهای دیگر در MANETs عمومی به نمایش

میگذارد. محیطهای فاقد ساختار زیربنایی و توپولوژی شبکه ای با پویایی بالا موجب تقسیم بندی مداوم شبکه ها میشود. شبکه‌های vanet را می‌توان جزئی از سیستم‌های انتقال هوشمند ITS[1] مطرح کرد

همکاری دولتها، گروههای استانداردسازی و تولیدکنندگان سراسر جهان پیشرفت سیستمهای انتقال هوشمند (ITS) را تسریع میکند. موفقیت بالقوه تحقیقات VANETs ، به عنوان بخش ضروری تحقیقات ITS، وابسته به توسعه سیستم ارتباط وسایل نقلیه است که توزیع مناسب، پایدار و مقرون به صرفه اطلاعات را در جهت بهره گیری از امنیت و آسایش "در راه انتقال" امکانپذیر میسازد.

در میان برنامه های کاربردی ارتباطی گوناگون در VANETs، طیف گسترده ای از برنامه های مهم وجود دارد که امنیت ترافیک، کنترل ترافیک و برنامه های کاربردی وسایل نقلیهٔ فاقد راننده را که نیازمند ارتباط محدود به زمان در شبکه های بیسیم تک کاره هستند، شامل میشود. این برنامه های توزیع شدهٔ دارای محدودیت زمانی را ارتباط همزمان مینامیم.

همانطور که در بالا بررسی شد، گره های بسیار پویا و سِرورِ فاقد ایستگاه مرکزی موجب بروز تصادم در رسانه بیسیم در ارتباط VANETs میشوند. در رسانه مورد نظر، تاخیر و مفقود شدن بسته ها مرتبا اتفاق می افتد. ارتباط همزمان به راحتی در این سناریوها دچار بی حاصلی میشود. بنابراین، توسعه یک سری سازوکارهای موثر در تضمینِ اتمام این ارتباطهای حساس به زمان، در یک بازه زمانی محدود ضروری است

شبکه‌های VANET در بسیاری از موارد شبیه شبکه‌های MANET هستند از جمله می‌توان به موارد زیر اشاره نمود:

- دامنه کوتاه انتقال امواج رادیویی[2]
- پهنای باند کوتاه[3]
- انتشار امواج در جهت مناسب[4]
- ظرفیت ذخیره سازی محدود[5]

اما در جزئیات با این نوع شبکه‌ها تفاوت‌هایی دارند از جمله این‌که حرکت در شبکه‌های VANET ساختاریافته است زیرا معمولا حرکت خودروها در مسیرهای مشخصی در طول جاده‌ها و خیابان‌ها انجام می‌شود. بطور کلی میتوان برقراری ارتباط در VANETs ویژگیهای چالشی به خصوصی را در بر میگیرد:

- تغییر سریع توپولوژی[6]

---

[1] Intelligent Transportation System

[2] Short radio transmission range

[3] Low bandwidth

[4] Omni directional broadcast

[5] Limited storage capacity

[6] Rapid topology changes

- طبقه بندی مداوم شبکه[1]
- شبکه میانبر کوچک و موثر[2]
- افزونگی محدود به زمان و عملکرد[3]
- قابلیت پیش‌بینی موقعیت[4]
- قدرت به نسبت کافی[5]

قابلیت پیشبینی موقعیت و قدرت به نسبت کافی را میتوان در جهت پشتیبانی ارتباط میان وسایل نقلیه و میان وسیله نقلیه و حاشیه جاده به کار گرفت، در حالی که تغییر سریع توپولوژی، طبقه بندی مداوم شبکه، شبکه میانبر کوچک و موثر و افزونگی محدود در زمان و عملکرد مشکلات ارتباط را در VANETs تشدید میکند.

در سناریوهای VANETs سه چالش تعیین کننده وجود دارد که نقشی اساسی در دستیابی به ارتباط پایدار و موثر را ایفا میکنند که عبارتند از:

- چگونگی بهره گیری موثر از پهنای باند کوتاه
- چگونگی حفظ و نگهداری توپولوژی قطعه بندی شده به صورت پویا[6]
- چگونگی دستیابی به حداقل تاخیر در انتقال همزمان اطلاعات در موقعیتهای گوناگون

۲-۳- تکنولوژی‌ها و پروتکل‌های شبکه‌های VANET

شبکه‌های VANET تکنولوژی‌های مختلف بیسیم مانند DSRC[7] را که نوعی WIFI است همچنین تکنولوژی سلولی و WIMAX را به‌کار می‌گیرد. DSRC کانال‌های مخابراتی با برد کوتاه و متوسط به‌صورت یک‌طرفه و دو طرفه است که مشخصا برای اهداف متحرک مطابق با یک‌سری استانداردها و پروتکل‌ها طراحی شده است.

---

[1] Frequent network partition

[2] Small effective network diameter

[3] Limited redundancy in time and in function

[4] Position predictability

[5] Relatively sufficient power

[6] Dynamically fragmented topology

[7] Dedivated Short Range Communication

پروتکل‌های بی‌سیم باند کوتاه دیگر شامل IEEE 802.11 ، Blue tooth و CALM در این نوع شبکه‌ها قابل استفاده می‌باشند.

بکارگیری پروتکل‌های مسیریابی مختلف مانند DSR[1] و DSDV[2] در شبکه‌های vanet مورد بررسی قرار دارد ولی به نظر می‌رسد پروتکل‌های مسیریابی شبکه‌های ad_hoc مانند AODV[3] و DYMO در این نوع شبکه‌ها کارایی بیشتری دارند.

همچنین شبکه‌های VANET را می‌توان بعنوان جزئی از سیستم‌های انتقال هوشمند[4] مطرح کرد.

[1] Dynamic Source Routing

[2] Destination Sequence Distance Vector

[3] Ad hoc On-demand Distance Vector

[4] ITS: Intelligent Transporatation Systems

شبیه‌سازی شبکه‌های VANET

در NS2

۵-۱- مقدمه

پروتکلهای مختلف بی‌سیم در NS2 تعریف شده است این شبیه ساز امکان تنظیم بسیاری از پارامترهای مختلف را در توپولوژی های بیسیم در اختیار قرار می‌دهد. برای مثال می‌توان مدل انتشاری محیطی که شبیه سازی در آن صورت می‌گیرد را از بین مدلهای افت فضای آزاد[۱] و دوشعاعه[۲] به همراه سایه[۳] انتخاب نمود. تنها پترن آنتن تعریف شده برای NS، Omni-Directional است. با توجه به ماهیت Open-Source بودن NS، انواع دیگر آنتن و کانال قابل تعریف است.

همچنین انواع مختلفی از پروتکلهای Routing برای شبکه Wireless تعریف شده است که از آن جمله می‌توان به Destination Sequence Distance Vector (DSDV)، Dynamic Source Routing (DSR)، Temporally ordered Routing Algorithm (TORA) و Adhoc On-demand Distance Vector (AODV) اشاره نمود.

یکی از ویژگیهای منحصر به فرد NS، امکان ایجاد تحرک[۴] در شبیه سازی شبکه های بیسیم است. بدین معنی که با تعریف موقعیت مکانی اولیهٔ Station ها، سرعت و در نهایت بردار جهت حرکت آنها، می‌توان در حین شبیه سازی، گیرنده و فرستنده ها را جابجا نمود. NS خود با در نظر گرفتن تاخیر ذاتی مسیر بین نود ها و مدل افت محیط، توان دریافتی (میزان خطا) و سایر پارامتر های شبیه سازی را لحظه به لحظه محاسبه می‌کند. می‌توان برای media یک مدل خطا نیز در نظر گرفت (Error Model). علاوه بر این امکان تعریف حرکت تصادفی (بصورت آماری) نیز برای Station ها وجود دارد.

پروتکل DCF[۵] لایهٔ MAC برای IEEE 802.11 در NS2 پیاده سازی شده است. این پروتکل برای حالت Unicast از الگوی RTS/CTS/DATA/ACK استفاده می‌کند (حالت معمولی) و برای حالت Broadcast بسته را به همهٔ گیرنده ها ارسال می‌کند. علاوه بر این پروتکل، پروتکلی بنام TDMA-MAC نیز برای لایهٔ MAC طراحی شده است که در دست توسعه می‌باشد. در این روش، پروتکل، بازه های زمانی (Time-slot) متفاوتی را در یک فریم برای ارسال و دریافت داده های نود ها استفاده می‌کند.

برای شبیه سازی شبکه های مرکب (Wired+Wireless) مجموعه رویه هایی به NS اضافه گردید که از آن جمله نودِ "BaseStationNode" است که مانند یک Gateway بین شبکهٔ Wired و Wireless عمل می‌کند. با تعریف این شئ هر مجموعه از Station های بیسیم در یک Domain با یک شماره مشخصهٔ یکتا قرار می‌گیرند و BaseStationNode آن حوزه، مسوول رساندن بسته ها به مقاصد Wireless می‌باشد. پس از تعریف این شئ، امکان پیاده سازی پروتکل های مسیر یابی در توپولوژی های مختلط (سیمی و بیسیم)

---

[1] Free Space

[2] Two-Ray

[3] Shadowing

[4] Mobility

[5] Distributed coordination function (DCF)

میسر شده و نیز پروتکلهایی مانند MobileIP برای شبکهٔ Wireless تعمیم داده شد. (پروتکل MobileIP توسط SUN Micro Systems ابتدا برای توپولوژی های Wired تعریف و پیاده سازی شده بود)

در مجموع امکانات NS برای شبیه سازی Wireless محدود می‌باشد. اگرچه بعضی ویژگیهای آن همانند قابلیت تحرک کاربران منحصر به فرد می‌باشد، اما همهٔ حالات ممکن، (مثلاً Roaming) در آن لحاظ نشده است. علاوه بر این، NS انواع اندکی از ترافیک های از پیش تعریف شده (مانند CBR، VBR، FTP و ...) را در اختیار قرار می‌دهد و با توجه به رابط کاربری مبتنی بر text آن، ایجاد توپولوژی یک شبکهٔ بزرگ و شبیه سازی ترافیک نزدیک به واقعیت روی آن بسیار پیچیده و وقتگیر خواهد بود.

ایجاد مدل حرکتی نودها با جزئیات کامل که شامل حرکت خودروها در مسیرهای خاص و در جهات رفت و برگشت ، سرعت حرکت خودروها و غیره می‌شود به راحتی امکان‌پذیر نمی‌باشد برای حل این مشکل معمولا از شبیه‌سازهای کمکی دیگری مانند SUMO و MOVE استفاده می‌شود. در این بخش از گزارش نمونه‌های شبیه‌سازی شده‌ای از VANET ارائه شده است در نمونه ۱و ۲ این شبیه‌سازی‌ها فقط با استفاده از قابلیت‌های شبیه‌ساز NS2 انجام شده است و در نمونه ۳ مدل حرکتی نودها با استفاده از شبیه‌سازهای SUMO , MOVE انجام شده است.

۵-۲- نمونه شبیه‌سازی شده ۱

در این بخش یک برنامه tcl و نحوه خروجی گرفتن از برنامه ارائه شده است.

۵-۲-۱- طرز اضافه کردن یک پروتکل جدید به NS-2

برای استفاده از هر پروتکل در برنامه TCL باید به مشخصات ان پروتکل کاملا اطلاع داشته باشید. برای اینکه بتوان در برنامه TCL پارامترهای مختلفی که در یک پروتکل تعریف گردیده مقداردهی کرد. بایستی ابتدا به پوشه protocol.cc در برنامه NS-2 نگاهی بیندازیم این پارامترها با دستور bind به اصطلاح به برنامه tcl چسبانده شود. در زیر یک مثال آورده شده است.

protocol.cc

protocol:protocol ()
{
bind("DelayCount_“,&DelayCount);
}

simulation.tcl
Agent/Protocol set DelayCount_ 1

حال اگر میخواهید پروتکلی را که تا کنون در برنامه NS شما وجود نداشته به ان اضافه کنید. مجبورید بعضی از فایلهای NS را تغییر دهید. در این قسمت کارهای مورد نیاز را برای تغییر فایلها را شرح می دهد.

common/packet.h
class p_info { public:

```
p_info() {
name_[PT_TCP]= "tcp";
name_[PT_PROTOCOL] = “protocol”;
```

tcl/lib/ns-default.tcl

```
Agent/Protocol set packetSize_ 32
```

tcl/lib/ns-packet.tcl

```
foreach prot {
AODV
Protocol
```

Makefile

```
OBJ_CC = \
random.o rng.o ranvar.o misc.o timer-handler.o \
...
protocol/protocol.o
$(OBJ_STL)
```

گاهی اوقات مجبور هستیم که فایلهای cmu-trace.cc و cmu-trace.h را نیز تغییر دهیم تا پروتکل شبیه‌سازی بیسیم فایلها را تولید می کند.

cmu-trace.cc

```
#include <protocol/protocol.h>
// command
void CMUTrace::format_protocol(Packet *p, int offset)
{
// packet
}
// another syntax
void CMUTrace::format(Packet* p, const char *why)
hdr_cmn *ch = HDR_CMN(p);
int offset = 0;
/*
* Log the MAC Header
*/
format_mac(p, why, offset);
if (namChan_)
nam_format(p, offset);
offset = strlen(wrk_);
switch(ch->ptype()) {
case PT_MAC:
break;
case PT_ARP:
format_arp(p, offset);
break;
```

```
default:
format_ip(p, offset);
offset = strlen(wrk_);
switch(ch->ptype()) {
case PT_AODV:
format_aodv(p, offset);
break;
case PT_TORA:
format_tora(p, offset);
break;
case PT_IMEP:
format_imep(p, offset);
break;
case PT_DSR:
format_dsr(p, offset);
break;
case PT_MESSAGE:
case PT_UDP:
format_msg(p, offset);
break;
case PT_TCP:
case PT_ACK:
format_tcp(p, offset);
break;
case PT_CBR:
format_rtp(p, offset);
break;
case PT_PROTOCOL:
format_protocol(p, offset);
break;
default:
fprintf(stderr, "%s - invalid packet type (%s).\n",
__PRETTY_FUNCTION__, packet_info.name(ch->ptype()));
exit(1);
}
}
}
}
```

cmu-trace.h

```
class CMUTrace : public Trace {
public:
CMUTrace(const char *s, char t);
void recv(Packet *p, Handler *h);
void recv(Packet *p, const char* why);
private:
char tracename[MAX_ID_LEN + 1];
int nodeColor[MAX_NODE];
```

```
int tracetype;
MobileNode *node_;
int newtrace_;
int initialized() { return node_ && 1; }
int node_energy();
int command(int argc, const char*const* argv);
void format(Packet *p, const char *why);
void nam_format(Packet *p, int offset);
void format_mac(Packet *p, const char *why, int offset);
void format_ip(Packet *p, int offset);
void format_arp(Packet *p, int offset);
void format_dsr(Packet *p, int offset);
void format_msg(Packet *p, int offset);
void format_tcp(Packet *p, int offset);
void format_rtp(Packet *p, int offset);
void format_tora(Packet *p, int offset);
void format_imep(Packet *p, int offset);
void format_aodv(Packet *p, int offset);
void format_protocol(Packet *p, int offset);
};
```

همانطور که در بخش های قبلی اشاره شد روش متداول برای تولید برنامه tcl برای شبکه های vanet استفاده از نرم افزار move و sumo می باشد. در ادامه به بیان چگونگی تولید برنامه Tcl در شبیه‌ساز ns2 اشاره می‌شود. البته در یک روش دیگر می‌توان از نرم افزار مطلب نیز استفاده کرد. برای این کار حرکت گره ها به صورت تصادفی در مطلب تولید می‌شود پس از آن به نرم افزار ns2 بار می‌شود.

در ابتدا برنامه TCL باید ابتدا مشخصات مختلف به صورت زیر اورد شود. مشخصات آمده در جدول زیر برای یک شبکه بیسیم می‌باشد. اکثر برنامه‌های TCL از استاندارد زیر استفاده می‌کنند. از دستور SET برای تعیین مشخصات مختلف استفاده می‌شود. برای مثال در اخر آخر پارامتر nn مقدار ۲ را می‌گیرد که در ادامه از آن استفاده می‌شود. یا در خط زیر

set val(rp) DSDV ;# ad-hoc routing protocol

rp مقدار DSDV را می‌گیرد. در کل برنامه هر وقت rp را می‌نویسیم مقدار آن برابر با DSDV است.

Define options

```
set val(chan) Channel/WirelessChannel ;# channel type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(ant) Antenna/OmniAntenna ;# Antenna type
set val(ll) LL ;# Link layer type
set val(ifq) Queue/DropTail/PriQueue ;# Interface queue type
set val(ifqlen) 50 ;# max packet in ifq
set val(netif) Phy/WirelessPhy ;# network interface type
set val(mac) Mac/802_11 ;# MAC type
set val(rp) DSDV ;# ad-hoc routing protocol
set val(nn) 2 ;# number of mobilenodes
```

در برنامه Tcl مورد نظر باید ابتدا ns_ node-config تعریف شود. در ادامه یک فایل ns-node-config ساده آورده شده است.

```
$ns_ node-config

    -addressingType flat or hierarchical or expanded
    -adhocRouting DSDV or DSR or TORA
    -llType LL
    -macType Mac/802_11
    -propType "Propagation/TwoRayGround"
    -ifqType "Queue/DropTail/PriQueue"
    -ifqLen 50
    -phyType "Phy/WirelessPhy"
    -antType "Antenna/OmniAntenna"
    -channelType "Channel/WirelessChannel"
    -topoInstance $topo
    -energyModel "EnergyModel"
    -initialEnergy (in Joules)
    -rxPower (in W)
    -txPower (in W)
    -agentTrace ON or OFF
    -routerTrace ON or OFF
    -macTrace ON or OFF
    -movementTrace ON or OFF
```

فرمت بالا حالت استاندارد است. در غیر این صورت خطا خواهد داد. فاصله بین خطوط اصلا نباید آورده شود. بعضی اوقات ممکن برنامه بدون هیچ دلیلی خطا دهد برای این کار بهتر است از یک برنامه پیش فرض استاندارد استفاده شود. تا حد امکان از copy/paste خودداری کنید. زیرا بعضی اوقات برنامه خطا می‌دهد بدون اینکه دلیل ان را متوجه باشید. فیلد addressType نشان دهنده همبندی شبکه است. بیان گر این است که شبکه به صورت سلسله مراتبی یا معمولی باشد. جلوی addressType ، یکی از پرامترها نوشته شود برای مثال flat انتخاب شود. adhocRouting نشان دهنده نوع مسیریابی است که باید استفاده شود. انواع مسیریابی های مختلف را در فایل Lib می‌توانید مشاهده کنید.

شبکه‌های vanet براساس استاندارد 802.11 می‌باشد در نتیجه باید پروتکل لایه mac شما 802.11 انتخاب شود. در غیر این صورت اگر پروتکل mac دیگری تعریف کرده‌اید آن را می‌توانید ذکر کنید. بقیه قسمتها همانند نوع آنتن و انرژی مصرف طبق فرمت استاندارد باید ذکر شود. برای موبایل کردن گره‌ها قسمتهای قرمز رنگ باید حتما ON شود.

ایجاد یک توپولوژی جدید

```
set ns_ [new Simulator]
set tracefd [open simple.tr w]
$ns_ trace-all $tracefd
set topo [new Topography]
$topo load_flatgrid 500 500
```

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

در هر برنامه tcl باید خط set ns_[new simulator] آورده شود. در خط دوم به فایل خروجی برنامه ns2 اشاره می‌شود. فایل خروجی با نام simple و به فرمت tr می‌باشد.

خط سوم نشان دهنده این است که همه مشخصات اجرای برنامه در فایل خروجی نشان دهد. در خط چهارم یک توپولوژی جدید ایجاد می‌شود و در خط پنجم نیز مساحت شبیه‌سازی را نشان می‌دهد که در این مثال 500x500 می‌باشد.

در خط آخر نیز نوع توپولوژی را نشان می‌دهد که به صورت خوشه بندی است و تعداد سر خوشه ها برابر با ۲ است. همانطور که قبلا بیان شد مقدار nn برابر با ۲ در نظر گرفته شده است.

در ادامه به بررسی گره‌های موبایل اشاره می‌شود.

متحرک کردن گره‌ها

```
for {set i 0} {$i < $val(nn) } {incr i} {
set node_($i) [$ns_ node ] $node_($i) random-motion 0 ;
motion }  # disable random

# # Provide initial (X,Y, for now Z=0) co-ordinates
# # for node_(0) and node_(1)
$node_(0) set X_ 5.0
$node_(0) set Y_ 2.0
$node_(0) set Z_ 0.0
$node_(1) set X_ 390.0
$node_(1) set Y_ 385.0
$node_(1) set Z_ 0.0
node_(0)  # # Node_(1) starts to move towards
15.0"   $ns_ at 50.0 "$node_(1) setdest 25.0 20.0
$ns_ at 10.0 "$node_(0) setdest 20.0 18.0 1.0"


node_(0)  # Node_(1) then starts to move away from
$ns_ at 100.0 "$node_(1) setdest 490.0 480.0 15.0"
$ns_ at 50.0 "$node_(1) setdest 25.0 20.0 15.0"
```

در این قسمت دو گره موبایل در نظر گرفته شده است که در حلقه for برای هر کدام از گره‌ها یک اسم در نظر گرفته می‌شود. حلقه For اغلب برای تعداد بیشتر گره‌ها استفاده می‌شود. برای دو حلقه می‌توانید از دستور set استفاده کنید. پارامترهای دیگر برای مشخص کردن جایگاه اولیه گره‌ها استفاده می‌شود که با پارامترهی x و y و z مشخص شده است. در ادامه به بررسی مهم ترین دستور که برای شبکه‌های vanet استفاده می‌شود پرداخته می‌شود.

برای تولید متحرک بودن یک گره، یک دستور اجرایی setdest استفاده می‌شود. فرمت کلی دستور setdest به صورت زیر می‌باشد که در ادامه به آن اشاره شده است.

```
setdest -v 1 -n 20 -p 2.0 -M 10.0 -t 200 -x 500 -y 500
```

```
setdest -v 2 -n 20 -s 1 -m 1 -M 10.0  -t 200 -P 1 -p 2.0 -x 500 -y 500
```

در جدول زیر فایلهای خروجی آن نشان داده شده است.

	Function	Range	Default	Ver 1	Ver 2
-M	maximum speed	≥ 0	0.0	Yes	Yes
-m	minimum speed	[0, maximum speed]	0.0	No, always zero	Yes
-n	number of nodes	1,2,...	0	Yes	Yes
-P	pause type	1 (constant), 2 (uniform)	1	No	Yes
-p	pause time	≥ 0	0.0	Yes	Yes
-s	speed type	1 (uniform), 2 (normal)	1	No	Yes
-t	maximum time	≥ 0	0.0	Yes	Yes
-v	version	1 (1999 CMU), 2 (2003 UM)	1	Yes	Yes
-x	width of space	≥ 0	0.0	Yes	Yes
-y	height of space	≥ 0	0.0	Yes	Yes

برای اجرای دستورات مربوط به setdest به پوشه زیر بروید. در مثال زیر فرض بر این است شبیه‌ساز ns2 در پوشه زیر نصب شده است. طبق آدرس زیر به فایل setdest بروید.

en\setdest E:\cgywin\home\Administrator\ns-allinone-2.28\ns-2.28\indep-utils\cmu-scen-

بعد از ان مرحله دستور زیر را تایپ کنید.

```
setdest -v 1 -n 20 -p 2.0 -M 10.0 -t 200 -x 500 -y 500 > scen-20_v1.tcl
setdest -v 2 -n 20 -s 1 -m 1 -M 10.0  -t 200 -P 1 -p 2.0 -x 500 -y 500 >
scen-20_v2.tcl
```

بعد از این کار دو فایل خروجی scen-20_v1.tcl و scen-20_v2.tcl را می‌دهد که می توانید برای متحرک بودن گره ها استفاده کنید. در ادامه به برنامه tcl و نحوه خروجی گرفتن از برنامه پرداخته می‌شود.

۵-۲-۲- برنامه tcl با چهار گره موبایل

```
#
==========================================================================
# Define options
#
==========================================================================
set val(chan) Channel/WirelessChannel ;# channel type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(netif) Phy/WirelessPhy ;# network interface type
set val(mac) Mac/802_11 ;# MAC type
set val(ifq) Queue/DropTail/PriQueue ;# interface queue type
set val(ll) LL ;# link layer type
set val(ant) Antenna/OmniAntenna ;# antenna model
set val(ifqlen) 50 ;# max packet in ifq
set val(nn) 4 ;# number of mobilenodes
set val(rp) AODV ;# routing protocol
set val(x) 500 ;# X dimension of the topography in meters
set val(y) 500 ;# Y dimension of the topography in meters
set val(time) 2.0 ;# Simulation time in seconds
set data_interval 0.2 ;# CBR Traffic Interval
set start_time 0.5 ; # traffic start time
proc finish {} {
global ns_ tracefile namfile
$ns_ flush-trace
close $tracefile
close $namfile
exit 0
}
#
==========================================================================
# Main Program
#
==========================================================================
#
# Initialize Global Variables
#
set ns_ [new Simulator]
# Trace Files
set tracefile [open out.tr w]
$ns_ use-newtrace
$ns_ trace-all $tracefile
set namfile [open out.nam w]
```

```
$ns_ namtrace-all-wireless $namfile $val(x) $val(y)
# set up topography object
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
#
# Create God
#
create-god $val(nn)
set chan [new $val(chan)]
# configure node
$ns_ node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace ON \
-movementTrace ON \
-channel $chan
for {set i 0} {$i < $val(nn) } {incr i} {
set node_($i) [$ns_ node]
$node_($i) random-motion 0 ;# disable random motion
$ns_ initial_node_pos $node_($i) 20
}
# Provide initial (X,Y, for now Z=0) co-ordinates for mobilenodes
# and produce some simple node movements
$node_(0) set X_ 190
$node_(0) set Y_ 190
$node_(0) set Z_ 0.0
$node_(1) set X_ 100
$node_(1) set Y_ 100
$node_(1) set Z_ 0.0
$node_(2) set X_ 280
$node_(2) set Y_ 100
$node_(2) set Z_ 0.0
$node_(3) set X_ 50
$node_(3) set Y_ 230
$node_(3) set Z_ 0.0
# Initial Movement
for {set i 0} {$i < $val(nn) } {incr i} {
```

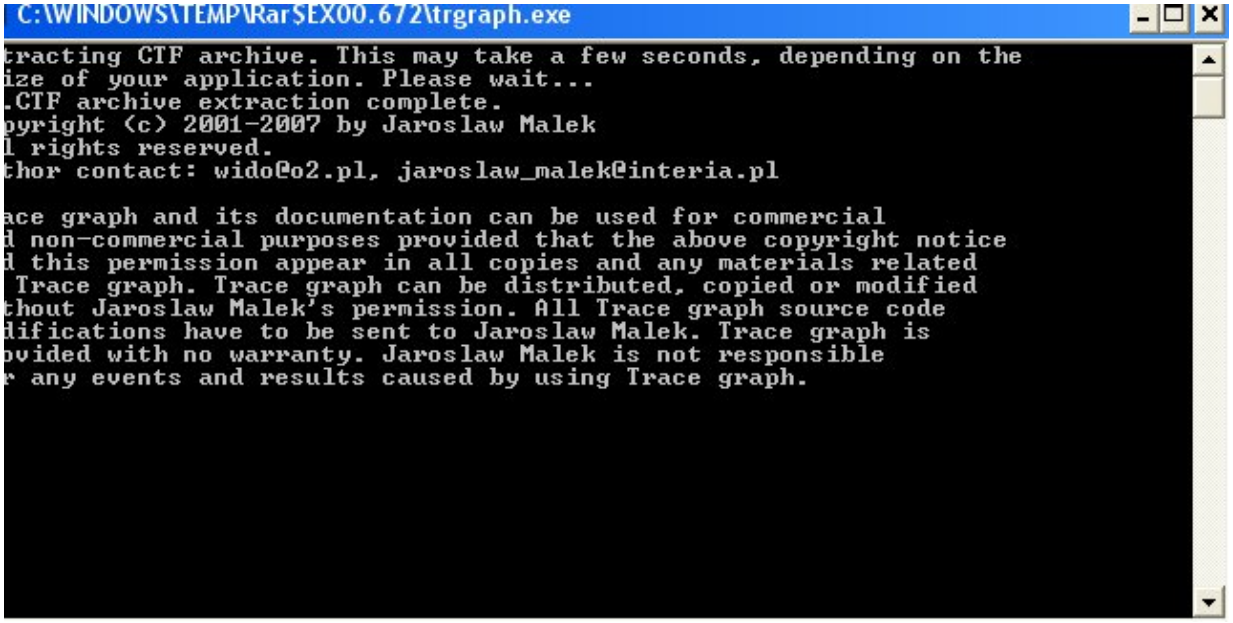
```
$ns_ at 0.0 "$node_($i) setdest 10.0 10.0 0.0"
}
# Null Agent to receive Packets for Node 0
set null_(0) [new Agent/Null]
$null_(0) set fid_ 1
$ns_ attach-agent $node_(0) $null_(0)
# Null Agent to receive Packets for Node 3
set null_(1) [new Agent/Null]
$null_(1) set fid_ 2
$ns_ attach-agent $node_(3) $null_(1)
# Data Source , Nodes
set udp_(0) [new Agent/UDP]
$udp_(0) set fid_ 0
$ns_ attach-agent $node_(1) $udp_(0)
# Constant Bit rate traffic genrator
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 1024
$cbr_(0) set interval_ $data_interval
$cbr_(0) set random_ 0
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at $start_time "$cbr_(0) start"
set udp_(1) [new Agent/UDP]
$udp_(1) set fid_ 0
$ns_ attach-agent $node_(2) $udp_(1)
set cbr_(1) [new Application/Traffic/CBR]
$cbr_(1) set packetSize_ 1024
$cbr_(1) set interval_ $data_interval
$cbr_(1) set random_ 0
$cbr_(1) set maxpkts_ 10000
$cbr_(1) attach-agent $udp_(1)
$ns_ connect $udp_(1) $null_(1)
$ns_ at $start_time "$cbr_(1) start"
# Tell nodes when the simulation ends
for {set i 0} {$i < $val(nn) } {incr i} {
$ns_ at $val(time) "$node_($i) reset";
}
$ns_ at $val(time) "finish"
$ns_ at [expr $val(time) + 0.01] "puts \"NS EXITING...\"; $ns_ halt"
puts "Starting Simulation..."
$ns_ run
```

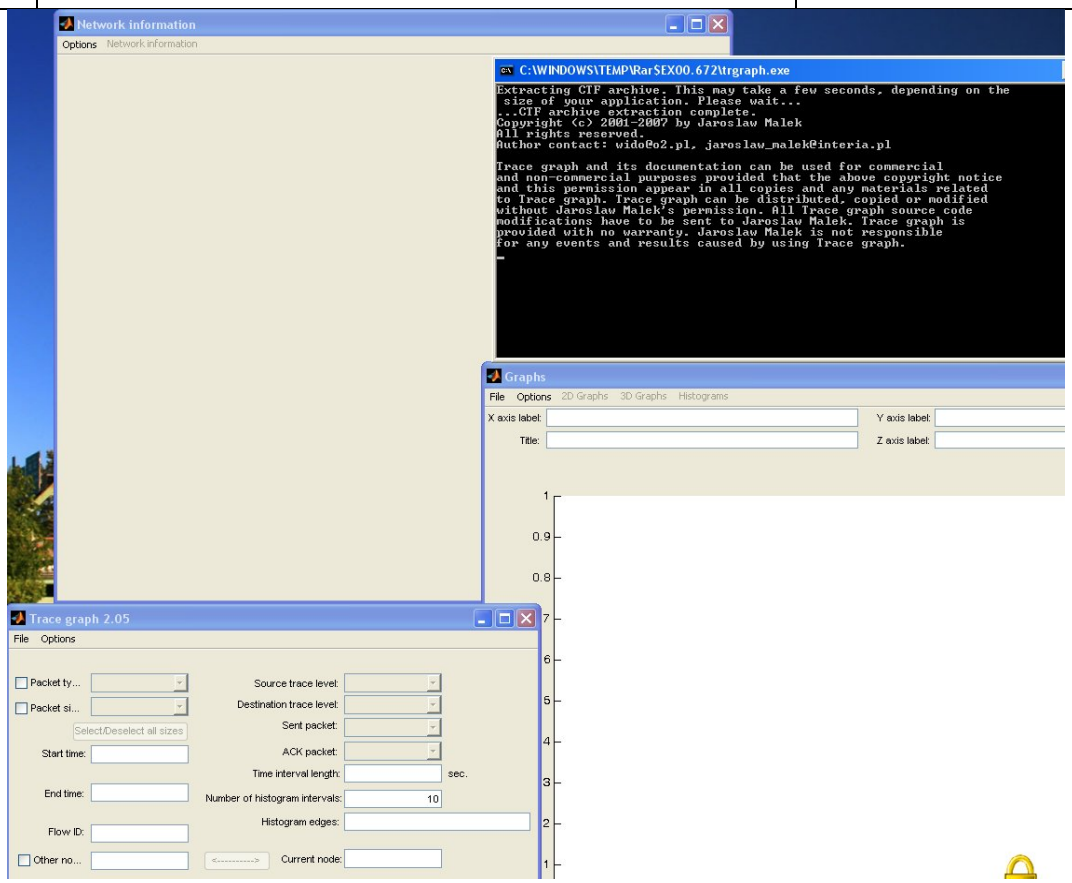
شبیه‌ساز ns2 باید یا برنامه مطلب نصب شود یا کتابخانه MCRInstaller.msi نصب شود. برای دانلود کتابخانه MCRInstaller.msi می توانید از سایت زیر استفاده کنید.

http://www.tracegraph.com/

پس از آن فایل tracegaraph را از سایت بالا دانلود کنید. همانطور که قبلا اشاره شد خروجی برنامه Tcl یک فایل tr است که ورودی برنامه xgraph می‌باشد.

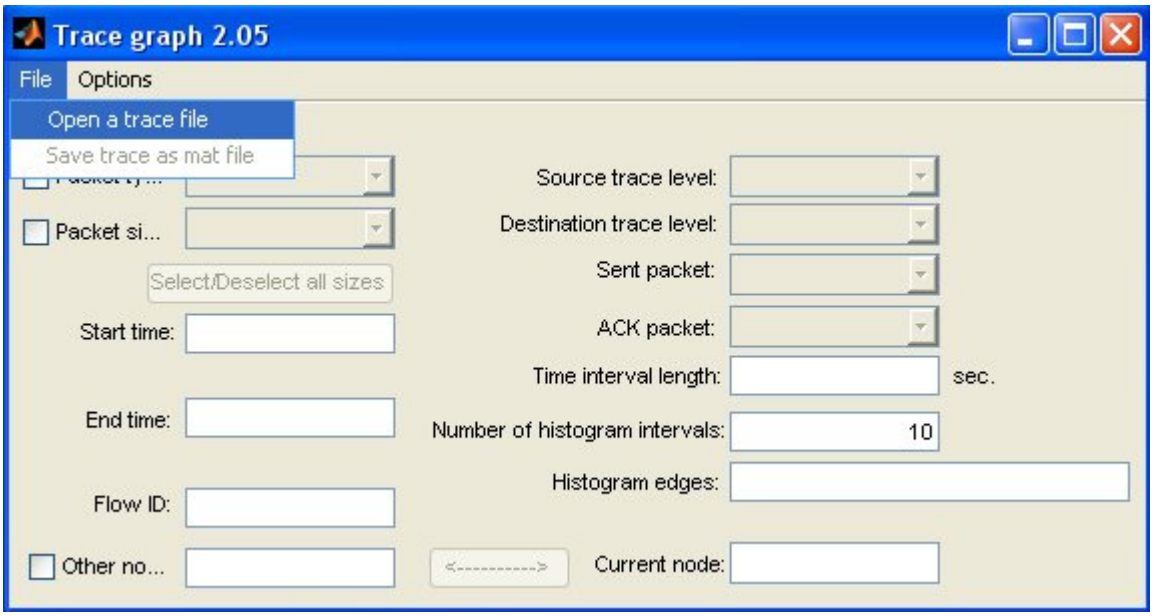
برنامه xgraph را از حالت زیپ شده خارج کنید. در داخل آن یک فایل به نام trgraph.exe وجود دارد بر روی آن کلیک کنید. ابتدا صفحه زیر ظاهر می‌شود.

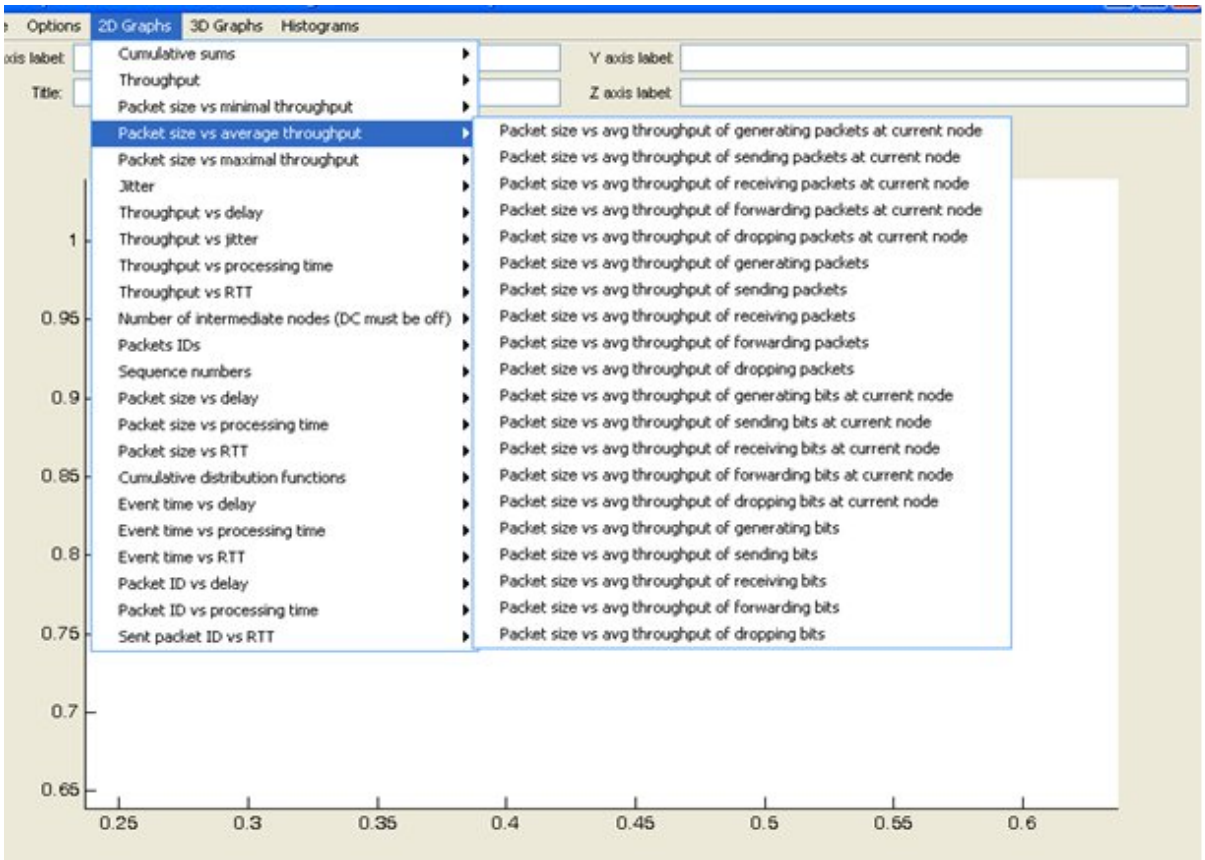




شکل ۳۸: محیط نرم افزار xgraph

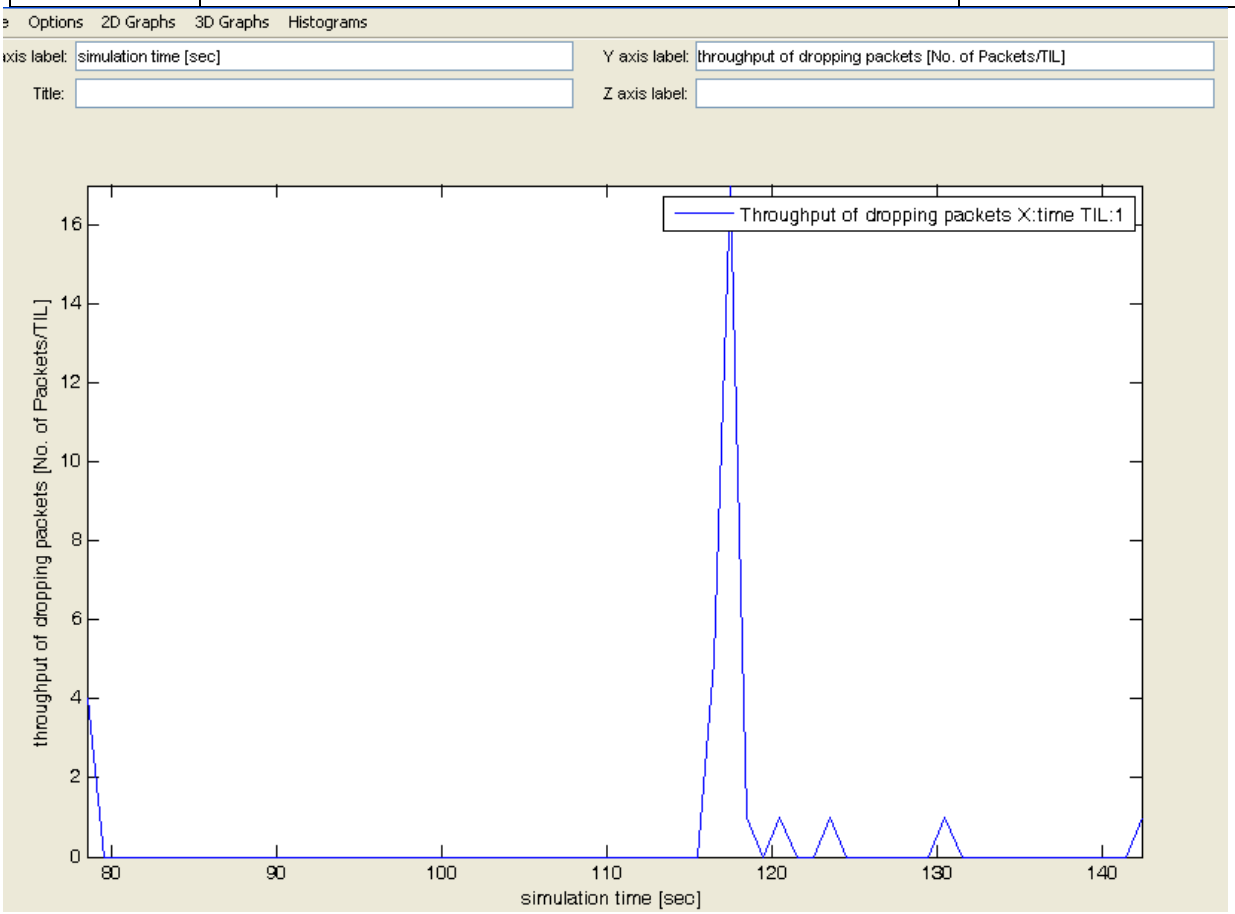
بعد از این مرحله بر روی file->open کلیک کنید.

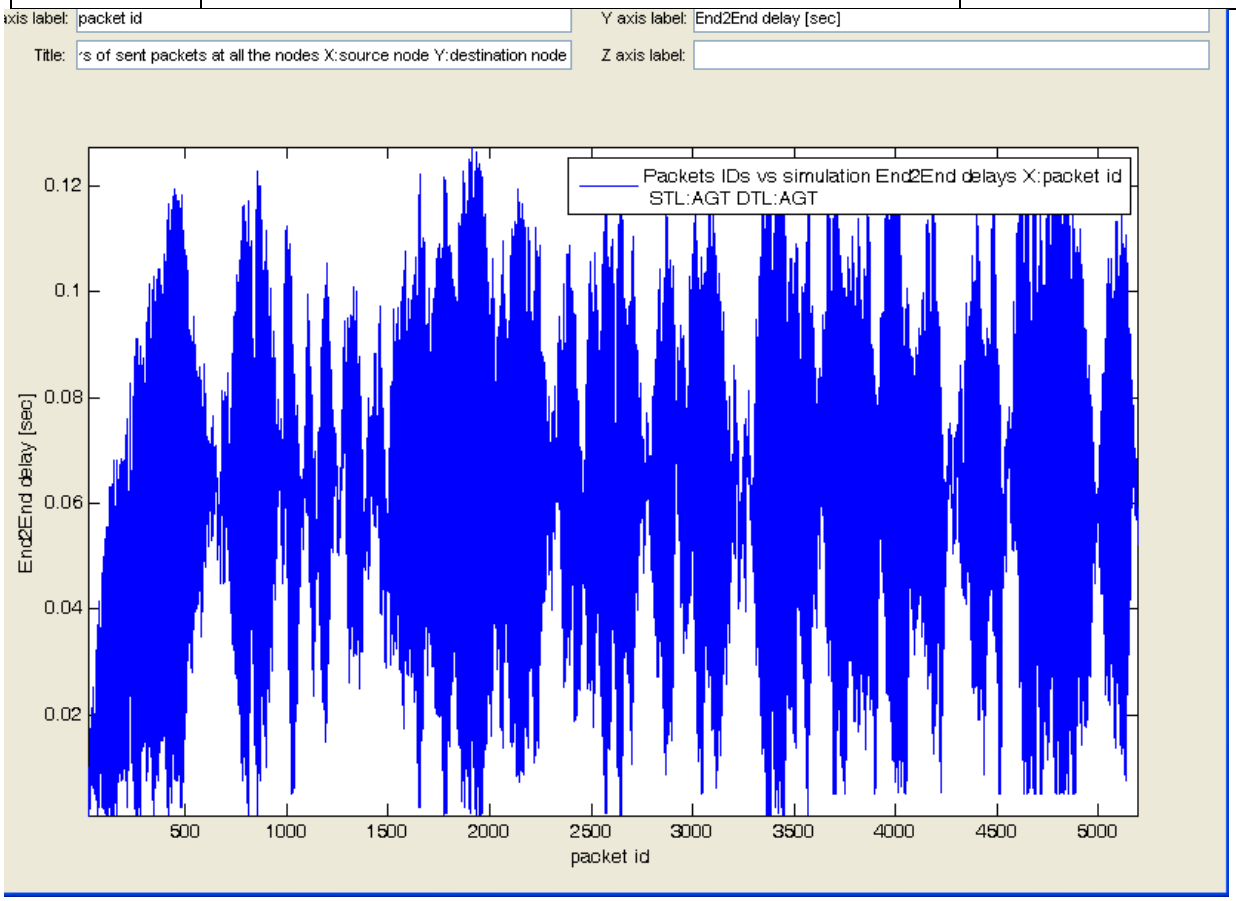


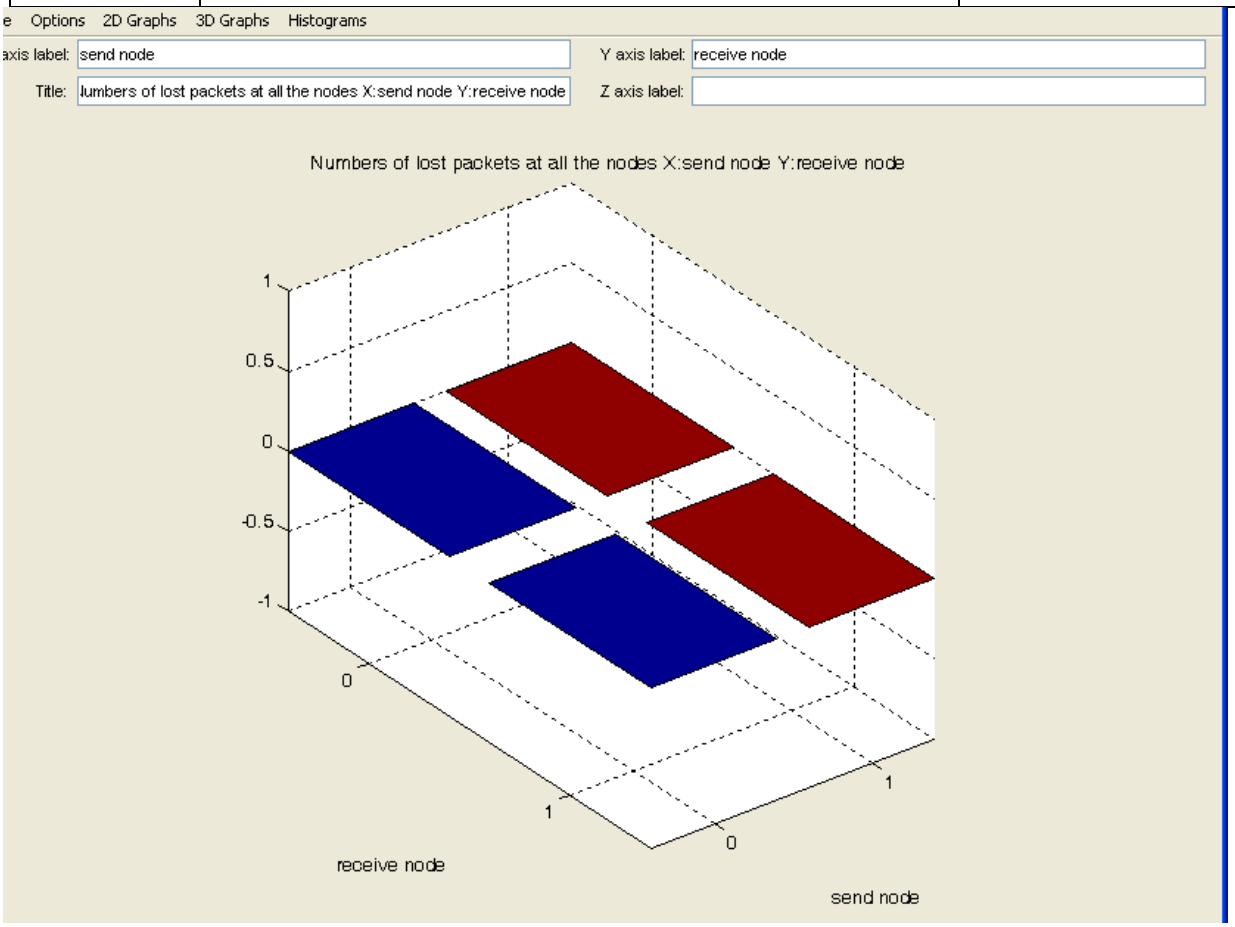


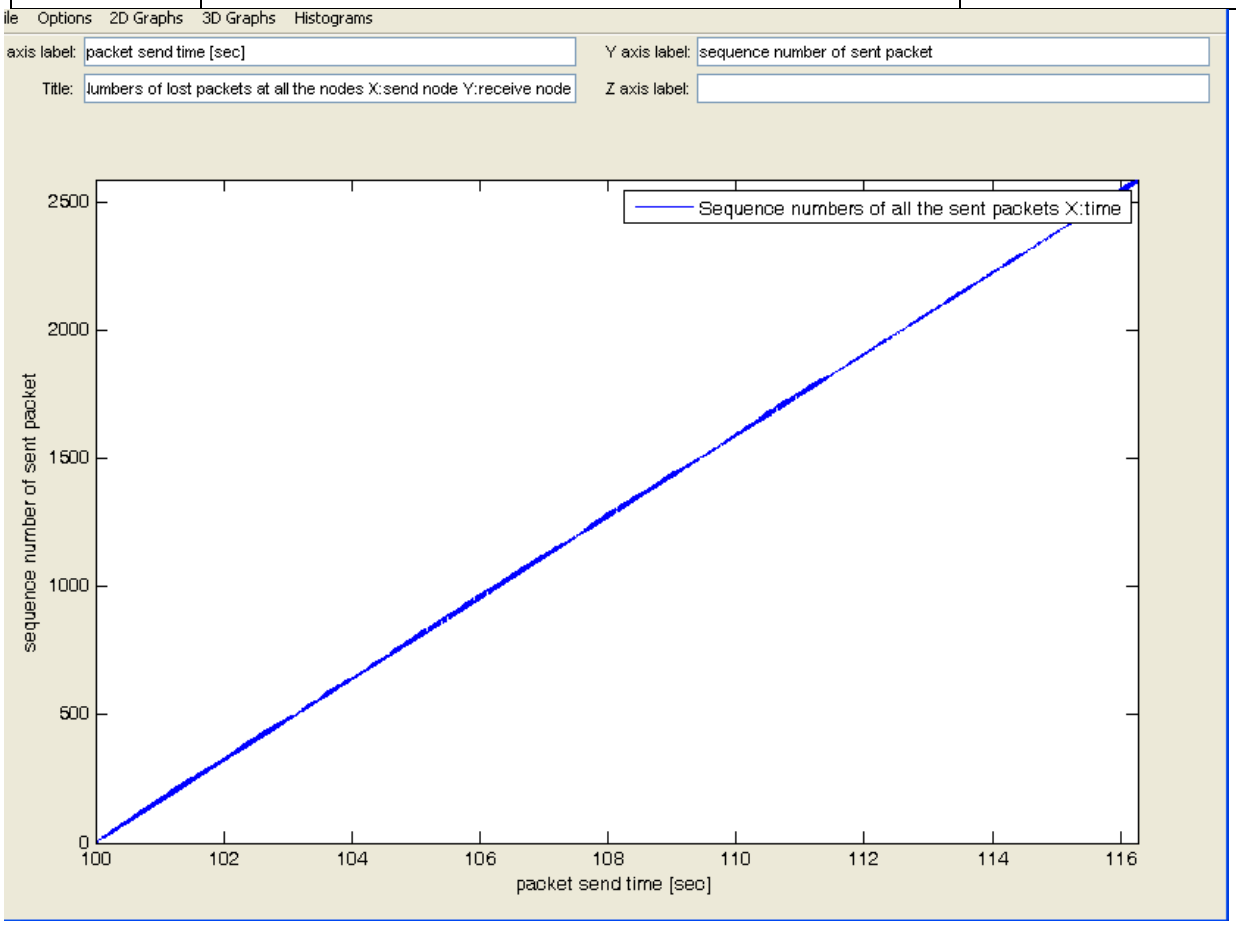
شکل ۳۹: باز کردن فایل Trace

بعد از این مرحله پنجره بزرگتر فعال می‌شود که پارامترهای زیادی را در آن مشاهده میکنید که برای خروجی گرفتن می‌توانید استفاده کنید


Options 2D Graphs 3D Graphs Histograms
axis label: simulation time [sec]
Y axis label: throughput of dropping packets [No. of Packets/TIL]
Title:
Z axis label:
Throughput of dropping packets X:time TIL:1
throughput of dropping packets [No. of Packets/TIL]
simulation time [sec]
0
2
4
6
8
10
12
14
16
80
90
100
110
120
130
140


axis label: packet id
Y axis label: End2End delay [sec]
Title: rs of sent packets at all the nodes X:source node Y:destination node
Z axis label:
Packets IDs vs simulation End2End delays X:packet id
STL:AGT DTL:AGT
End2End delay [sec]
0.12
0.1
0.08
0.06
0.04
0.02
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
packet id


e Options 2D Graphs 3D Graphs Histograms
axis label: send node
Y axis label: receive node
Title: lumbers of lost packets at all the nodes X:send node Y:receive node
Z axis label:
Numbers of lost packets at all the nodes X:send node Y:receive node
1
0.5
0
-0.5
-1
0
1
receive node
0
1
send node


ile Options 2D Graphs 3D Graphs Histograms
axis label: packet send time [sec]
Y axis label: sequence number of sent packet
Title: lumbers of lost packets at all the nodes X:send node Y:receive node
Z axis label:
Sequence numbers of all the sent packets X:time
sequence number of sent packet
packet send time [sec]
0
500
1000
1500
2000
2500
100
102
104
106
108
110
112
114
116

۵-۳- نمونه شبیه‌سازی شده ۲

در این بخش یک نمونه دیگر از برنامه tcl برای شبکه‌های vanet ارائه شده است این برنامه یک پروتکل broadcast را برای vanet شبیه‌سازی کرده است.

```
==================================================================#
#
  #Wireless simulation, a simulation of a broadcast protocol
  #for vehicular ad hoc networks.
  #
  #Nathan Balon
  #University of Michigan - Dearborn
  #CIS 695
#
 ==================================================================#
==================================================================#
 #Options
==================================================================#

set opt(chan)        Channel/WirelessChannel      ;# channel type
set opt(prop)        Propagation/TwoRayGround     ;# radio-propagation model
#set opt(prop)       Propagation/Shadowing
set opt(ant)         Antenna/OmniAntenna          ;# antenna type
set opt(ll)          LL                           ;# link layer
set opt(ifq)         Queue/DropTail/PriQueue          ;# interface queue
set opt(ifqlen)      50                           ;# max packet in ifq
set opt(netif)       Phy/WirelessPhy                  ;# network interface type
set opt(mac)         Mac/802_11                       ;# mac type
set opt(rp)          DumbAgent                    ;# routing protocol
set opt(nn)          50                               ;# number of mobile nodes
set opt(pkt_size)    250                          ;# size of  broadcast message
set opt(cp)          traffic.tcl                      ;# connection pattern
set opt(tr)          trace.tr                     ;# trace file
set opt(nam_tr)      ""                               ;# name trace file
set opt(seed)                                     #٠.٠seed the random number generator
set opt(stop)        65.0                         ;# time to end simulation
set opt(x)           1000                             ;# x size for topology
set opt(y)           1000                         ;# y size for topology
set opt(lm)          ON                               ;# log movement
set opt(at)          ON                           ;# agent trace
set opt(rt)          OFF                              ;# routing trace
set opt(mact)        ON                           ;# mac trace
set opt(movt)        ON                               ;# movement trace
set opt(cw)          15
```

```
Mac/802_11 set CWMax_        1023
Mac/802_11 set SlotTime_     0.000013          ;# 20us
Mac/802_11 set SIFS_         0.000032          ;# 10us
Mac/802_11 set PreambleLength_      32                ;# 144 bit
Mac/802_11 set PLCPHeaderLength_    40                ;# 48 bits
Mac/802_11 set PLCPDataRate_  6.0e6                   ;# 1Mbps
Mac/802_11 set dataRate_     6.0e6
Mac/802_11 set basicRate_    6.0e6

Mac/802_11 set RTSThreshold_        3000        ;# bytes
Mac/802_11 set ShortRetryLimit_     7           ;# retransmittions
Mac/802_11 set LongRetryLimit_      4           ;# retransmissions

Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 2.5118864e-13     ;# -96 dBm
Phy/WirelessPhy set RXThresh_ 1.0e-12           ;# -90 dBm
Phy/WirelessPhy set bandwidth_ 6.0e6
Phy/WirelessPhy set Pt_ 0.0003754
Phy/WirelessPhy set freq_ 5.9e+9
Phy/WirelessPhy set L_ 1.0

#Propagation/Shadowing set pathlossExp_ 2.7
#Propagation/Shadowing set std_db_     4.0
#Propagation/Shadowing set seed_       0
#Propagation/Shadowing set dist0_      1.0

 #Unity gain, omni-directional antennas
 #Set up the antennas to be centered in the node and 1.5 meters above it
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 4.0
Antenna/OmniAntenna set Gr_ 4.0

==========================================================================#

set BROADCAST_ADDR -1                   ;# broadcast address

set MESSAGE_PORT 42                 ;# periodic message port
set WARNING_PORT 33                 ;# warning message
set EMER_PORT 21                            ;# port to listen for emergency messages

Class Agent/MessagePassing/PeriodicBroadcast -superclass Agent/MessagePassing

Agent/MessagePassing/PeriodicBroadcast instproc recv {source sport size data} {
 #    This empty function is needed so receive works.
```

```
}

Agent/MessagePassing/PeriodicBroadcast instproc send_message} {}
$    self instvar node_
   global ns_ MESSAGE_PORT BROADCAST_ADDR opt

   #puts "[$node_ node-addr] sending message"
 #    send the broadcast message
$    self sendto $opt(pkt_size) 0 $BROADCAST_ADDR $MESSAGE_PORT
}

 #perform clean up at the end of the program
proc finish} {}
   global ns_ tracefd namtracefd opt
$    ns_ flush-trace
   close $tracefd
   if {$opt(nam_tr)"" =! }}
      close $namtracefd
   }
$    ns_ halt
   exit 0
}

 #set the options from the command line arguments
proc getopt {argc argv} {
   global opt
   lappend optlist cp nn seed sc stop tr x y bc_size nam_tr cw

   for {set i 0} {$i < $argc} {incr i} {
      set arg [lindex $argv $i[
      if {[string range $arg 0 0] != "-"} continue
      set name [string range $arg 1 end[
      set opt($name) [lindex $argv [expr $i+1]]
   }
}
 #log the movement of a node every 0.1 seconds
proc log-movement} {}
      global logtimer ns_ ns

      set ns $ns_
      source ../tcl/mobility/timer.tcl
      Class LogTimer -superclass Timer
      LogTimer instproc timeout} {}
            global opt node;_
```

```
            for {set i 0} {$i < $opt(nn)} {incr i} {
$                   node_($i) log-movement
            }
$             self sched 0.1
}
      set logtimer [new LogTimer[
$       logtimer sched 0.1
}
#get command line arguments
getopt $argc $argv

Mac/802_11 set CWMin_ $opt(cw (


puts "x: $opt(x), y: $opt(y)"
puts "CW: $opt(cw)"

if {$opt(seed) > 0} {
   puts "Seeding Random number generator with $opt(seed)\n"
   ns-random $opt(seed(
}

 #create a new simulator
set ns_ [new Simulator]

 #set up the traces
set tracefd [open $opt(tr) w]
$ns_ trace-all $tracefd



 #set the topology
set topo [new Topography]
$topo load_flatgrid $opt(x) $opt(y)

set god_ [ create-god $opt(nn)]
set chan_ [new Channel/WirelessChannel]

 #Set up the nam trace if desired.
if {$opt(nam_tr)"" =! }}
   set namtracefd [open $opt(nam_tr) w]
$    ns_ namtrace-all-wireless  $namtracefd $opt(x) $opt(y)
}

 #configure the nodes of the simulation
$ns_ node-config -adhocRouting $opt(rp)\
```

```
-        llType $opt(ll)\
-        macType $opt(mac)\
-        ifqType $opt(ifq)\
-        ifqLen $opt(ifqlen)\
-        antType $opt(ant)\
-        propInstance [new $opt(prop)]\
-        phyType $opt(netif)\
-        topoInstance $topo\
-        channel $chan\ _
-        agentTrace $opt(at)\
-        routerTrace $opt(rt)\
-        macTrace $opt(mact)\
-        movementTrace $opt(movt)
set y 0
set x 0
 #Create the mobile nodes for the simulation.
for {set i 0} {$i < $opt(nn)} {incr i} {
   set node_($i) [$ns_ node[
$   ns_ initial_node_pos $node_($i) 10
}

 #Source the mobility scenario file.
 source $opt(sc)

   #Attach a new Agent/MessagePassing/PeriodicBroadcast to each node on port
$MESSAGE_PORT
for {set i 0} {$i < $opt(nn)} {incr i} {
   set bc_agent($i) [new Agent/MessagePassing/PeriodicBroadcast[
$   node_($i) attach  $bc_agent($i) $MESSAGE_PORT
}

 #log movement
if { $opt(lm) == "ON "}}
   puts "Logging movement"...
   log-movement
}
 #Source the file that contains the broadcast traffic.
source $opt(cp)

$ns_ at $opt(stop) "finish"
```

```
#Add to the trace simulation parameters .
puts $tracefd "M 0.0 nn $opt(nn) x $opt(x) y $opt(y) rp $opt(rp)"
#puts $tracefd "M 0.0 cp $opt(cp) seed $opt(seed)"
puts $tracefd "M 0.0 prop $opt(prop) ant $opt(ant)"
puts "Starting Simulation"...
$ns_ run
```

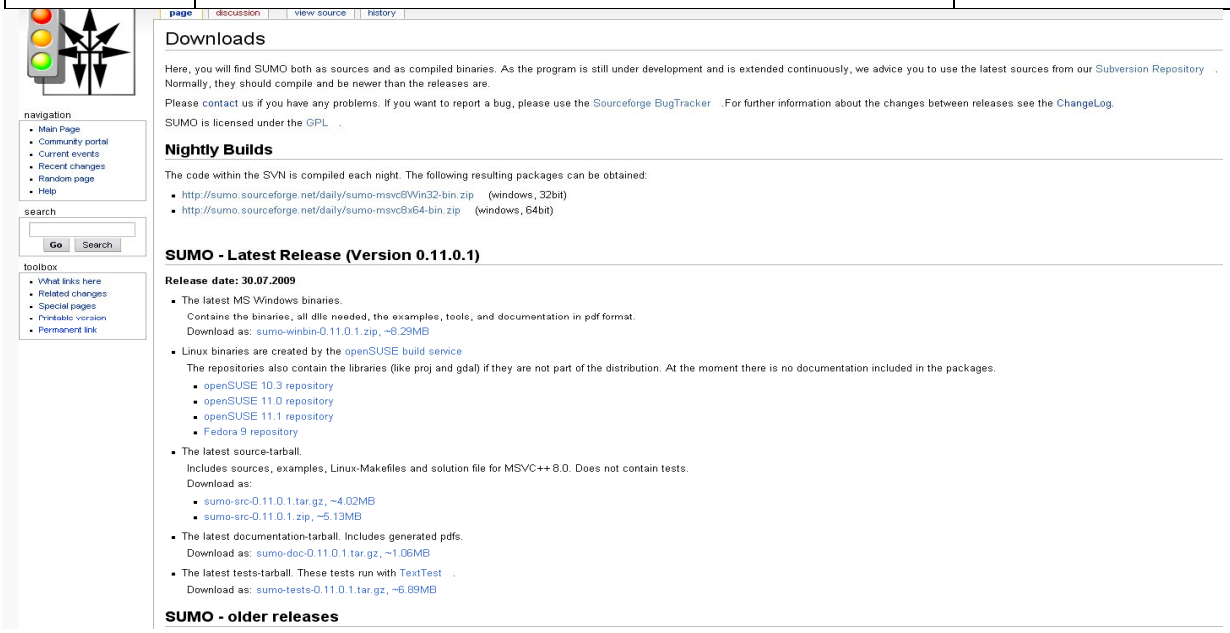
۵-٤- نمونه شبیه‌سازی شده ۳

همانطور که در قسمت قبل اشاره شد. اضافه کردن موبیلتی به گره‌ها در ns2 بسار پیچیده است و نمی‌شود به سادگی آن را در محیط پیاده‌سازی کرد. برای این بهتر است از شبیه‌ساز ns2 به همراه شبیه‌ساز SUMO استفاده کرد. نرم‌افزار SUMO برای شبیه‌سازی مدلهای حرکتی سیار شهری است. این نرم‌افزار می‌تواند جنبه‌های مختلف یک شبکه را با جزییات پیاده‌سازی کند و نقشها را به مدل‌های حرکتی تبدیل کرد یا اطلاعات را از پایگاه‌های داده‌ای سیستم اطلاعات جغرافیایی خوانده و در ns2 از آنها استفاده کرد.

اگر چه sumo نرم افزار قدرتمندی با قابلیت‌های بسار زیاد در ایجاد مدلهای حرکتی شبکه‌ای سیار شهری است اما فاقد محیط گرافیکی برای تولید چنین محیط‌هایی است. برای کار با آن باید به صورت دستی فایل‌های حاوی مشخصات خودروها و جاده‌ها و فایل‌های پیکربندی را ایجاد کرده و معمولا به صورت xml نوشته و در یک فایل sumi,cfg ذخیره می‌شود و با دستورات متنی آنها را به هم لینک داده یا فایلهای مورد نیاز تولید کرد.

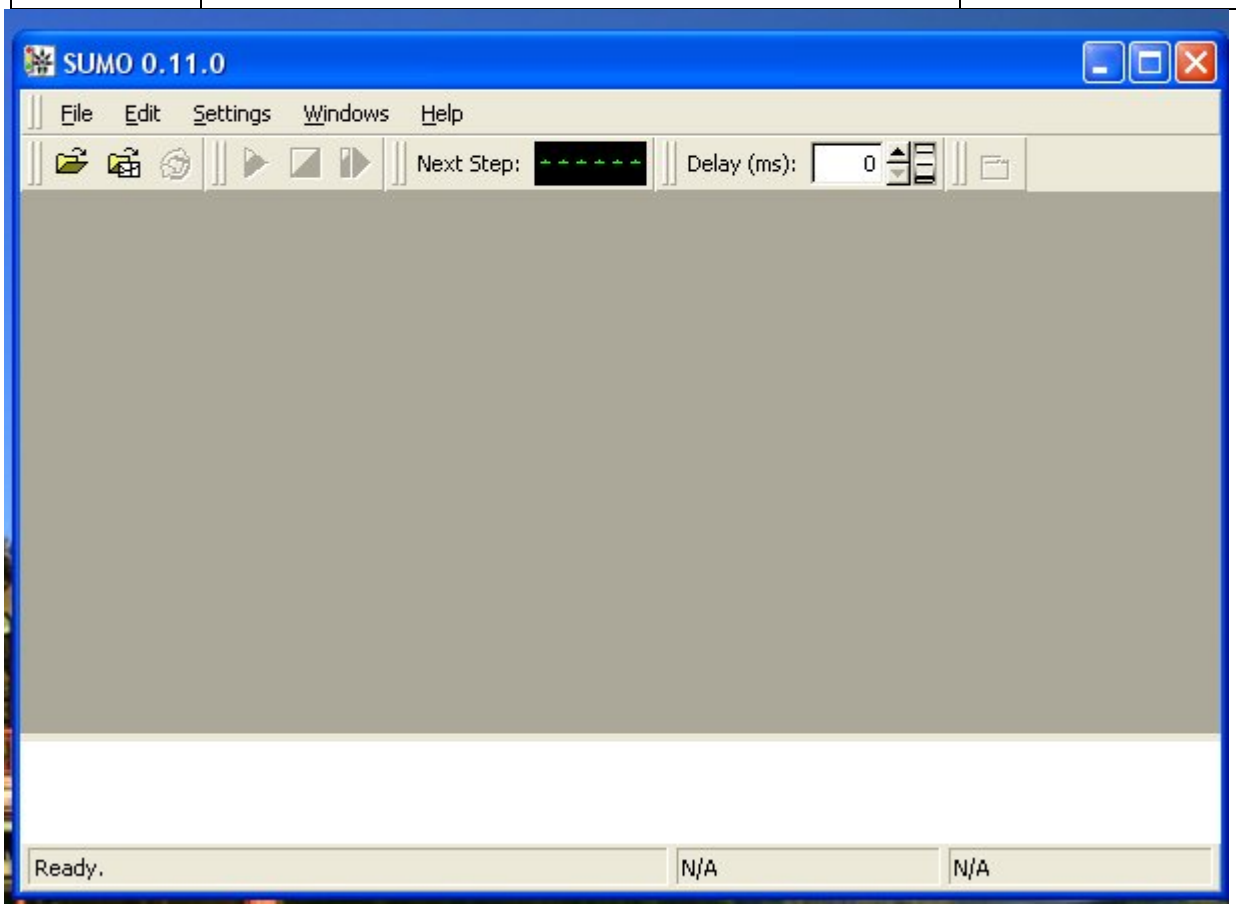
برای دریافت نرم‌افزار به لینک زیر رفته و شبیه‌ساز sumo را دانلود کنید.

http://sourceforge.net/apps/mediawiki/sumo/index.php?title=Downloads



شکل ۴۰: صفحه دانلود نرم‌افزار sumo

لازم به ذکر است مدل جدید دارای محیط گرافیکی می‌باشد.



شکل ۴۱: محیط نرم‌افزار sumo

نصب sumo در محیط ویندوز بسیار ساده است شما می‌توانید فایلهای باینری را از ادرس آمده در بخش قبل که مربوط به سیستم عامل لینوکس می‌باشد دانلود کنید. باید قبل از نصب نرم‌افزار باید نرم‌افزار Microsoft Visual C++ 2005 SP1 را نصب کنید. بعد از نصب نرم‌افزار در ENVIROMENT ویندوز پارامترهای زیر را باید حتما وارد کنید.

```
Path=
C:\Program Files\Microsoft Visual Studio 8\Common7\IDE;C:\Program Files\Microsoft Visual Studio 8\VC\BIN;C:\Program Files\Microsoft Visual Studio 8\Common7\Tools;C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\bin;C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\bin;C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\bin;C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727;C:\Program Files\Microsoft Visual Studio 8\VC\VCPackages;C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin;C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727;C:\Program Files\Microsoft Visual Studio 8\VC\bin;C:\Program Files\Microsoft Visual Studio 8\Common7\IDE;C:\Program Files\Microsoft Visual Studio 8\VC\vcpackages
```

INCLUDE=
C:\Program Files\Microsoft Visual Studio 8\VC\ATLMFC\INCLUDE;C:\Program Files\Microsoft Visual Studio 8\VC\INCLUDE;C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\include;\Program Files\Microsoft Visual Studio 8\SDK\v2.0\include

LIB=
C:\Program Files\Microsoft Visual Studio 8\VC\ATLMFC\LIB;C:\Program Files\Microsoft Visual Studio 8\VC\LIB;C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\lib;:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\lib

LIBPATH=
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727;C:\Program Files\Microsoft Visual Studio 8\VC\ATLMFC\LIB

NetSamplePath=
C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0

DevEnvDir=
C:\Program Files\Microsoft Visual Studio 8\Common7\IDE

FrameworkDir=
C:\WINDOWS\Microsoft.NET\Framework

FrameworkSDKDir=
C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0

FrameworkVersion=
v2.0.50727

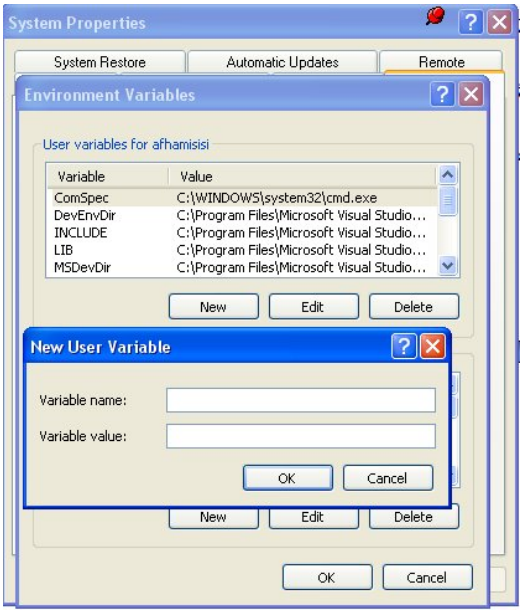
VCBUILD_DEFAULT_CFG=
Debug^|Win32

VCBUILD_DEFAULT_OPTIONS=
/useenv

VCINSTALLDIR=
C:\Program Files\Microsoft Visual Studio 8\VC

VSINSTALLDIR=
C:\Program Files\Microsoft Visual Studio 8

پس از نصب نرم‌افزار Microsoft Visual C++ و تغییر پارامترهای بالا در ویندوز (برای بر روی my computer کلیک راست کرده و گزینه properties را انتخاب می‌کنید و از انجا به منوی advanced رفته و Environment Variables را انتخاب کرده و پارامترها را در شکل () وارد می‌کنید.)



نرم‌افزار sumoدانلود شده را از حالت فشرده خارج می‌شود. به همین سادگی نرم‌افزار نصب می‌شود. در صورت خطا دادن در این قسمت حتما باید به نصب Microsoft Visual C++ و مراحل وارد کردن مشخصات نصب خود توجه کنید.

در این بخش با مالی بسیار ساده با اصول کار شبیه‌ساز sumo اشنا می‌شویم. در این مثال که ساده‌رین حالت ممکن را ایجاد می‌کند. یک خودرو را در حال رانندگی نشان می‌دهد. برای نوشتن برنامه از برنامه‌نویسی استفاده می‌شود. در شبیه‌ساز sumo، شبکه خیابانها از گره‌ها[1] و لبه‌ها[2] تشکیل شده است. لبه‌ها، گره‌ها را به هم متصل می‌نند. بنابراین در یک شبکه که میخواهیم دو خیابان را به متصل شود به سه گره و دو لبه نیاز داریم.

گره‌ها حاوی دو مشخصه طول- عرض جغرافیایی (فاصله تا مبدا به متر) و شناسه (id) برای دسترسی‌های بعدی نیازمند است.

[1] node
[2] Edge

```
<nodes>
<node id="1" x="-500.0" y="0.0" />
<node id="2" x="+500.0" y="0.0" />
<node id="3" x="+501.0" y="0.0" />
</nodes>
```

برنامه xml برای پیکربندی شبکه

برای اینکار می‌توانید از یک ویرایشگر همانند notepad یا PSeditor استفاده کنید. برای دانلود نرم‌افزار PSeditor می‌توانید از لینک زیر استفاده کنید.

http://www.softpedia.com/get/Office-tools/PDF/PSEDITOR.shtml

برنامه را به اسم hello.node.xml ذخیره کنید که node.xml پسوند پیش فرض برای فایل حاوی تنظیمات گره در sumo می‌باشد.

در مرحله بعد، گره‌ها را توسط لبه‌ها به هم متصل می‌کنیم. این کار بسیار ساده می‌باشد از دو مشخصه fromnode و tonode استفاده می‌شود که در آن fromnode گره ابتدای لبه و tonode گره پایانی گره می‌باشد.

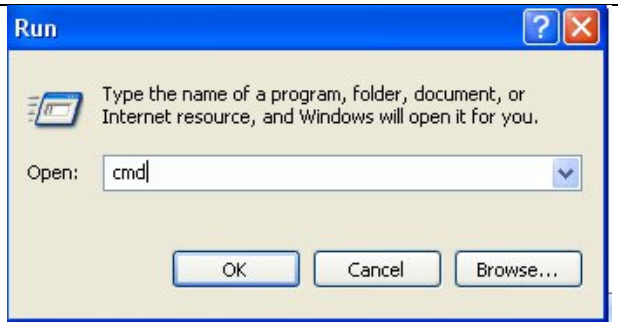
```
<edges>
<edge fromnode="1" id="1to2" tonode="2" />
<edge fromnode="2" id="out" tonode="3" />
</edges>
```

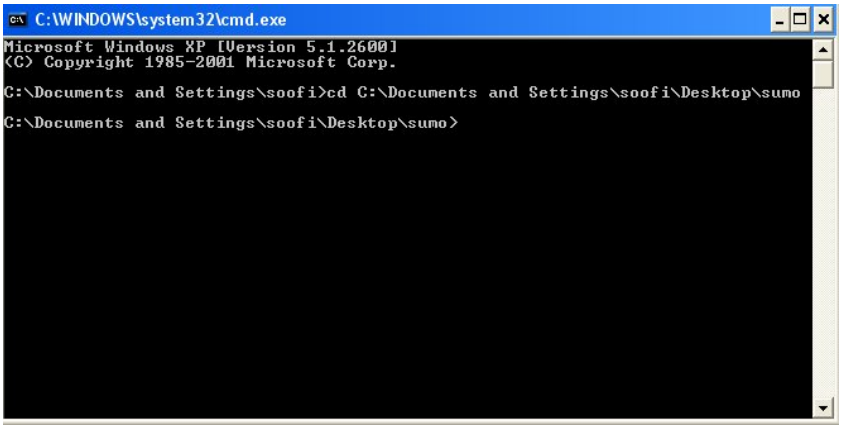
ارتباط میان لبه و گره

مشخصه id برای دسترسی‌های بعدی می‌باشد. داده‌ها را در فایل hello.edge.xml ذخیره کنید.

بعد از ایجاد گره‌ها و لبه‌ها، می‌توان از یکی از ابزار sumo به نام netconvert برای ایجاد کردن شبکه استفاده کنید.

بدین منظور ابتدا به محیط command prompt رفته و به مسیر نصب فایل تغییر مسیر می‌دهید. (طبق شکل ())





پس از این مرحله دستور زیر را می‌نویسید.

```
netconvert –xml-node-files=hello.nod.xml –xml-edge-files=hello.edg.xml –output file= hello.net.xml
```

پس از این دستور فایل hello.net.xml ایجاد می‌کند. بعد از این مرحله مسیرها ایجاد می‌شود. در sumo، خودروها دارای انواع مختلف هستند. زیرا خصوصیات اصلی آنها همانند طول، شتاب، حداکثر سرعت و... متفاوت هستند. علاوه براین خودرو پارامتر به نام sigma نیاز دارد که یک رفتار تصادفی را با توجه به مدل خودرو تولید می‌شود. مقداردهی صفر به این مشخصه باعث تولید یک خودرو قطعی می‌شود. اکنون یک مسیر برای خودرو ایجاد می‌شود که دو لبه را در بر می‌گیرد. علت اینکه از دو لبه استفاده می‌شود این است که یک خودرو هنگامی که به یک لبه می‌رسد ناپدید می‌شود.

```
<routes>
<vtype accel=”1.0” 25dged=”5.0” id=”Car” length=”2.0” maxspeed=”100.0”
sigma=”0.0” />
<route id=”route0”>1to2 out</route>
<vehicle depart=”1” id=”veh0” route=”route0” type=”Car” />
</routes>
```

کدها در فایل hello.ru.xml ذخیره می‌شود.

۵-۴-۱- پیکربندی

در این مرحله تمامی فایل‌ها را در یک فایل پیکربندی با هم پیوند زده می‌شود.

```
<configuration>
<files>
<net-file>hello.net.xml</net-file>
<route-files>hello.rou.xml</route-files>
</files>
<simulation>
<begin>0</begin>
<end>10000</end>
</simulation>
</configuration>
```

پس از این مرحله در فایل hello.sumo.cfg ذخیره می‌کنیم. اکنون می‌توان شبیه‌سازی را از طریق محیط command promp یعنی در محیط متنی اجرا کرد.

```
sumo –c hello.sumo.cfg
```

یا با دستور زیر در محیط گرافیکی sumo اجرا کرد.

Guisim –c hello.sumo.cfg

بعد از این مرحله باید نرم‌افزار move را دانلود کنید برای دانلود نرم‌افزار move از لینک زیر می‌توانید استفاده کنید.

http://lens1.csie.ncku.edu.tw/MOVE/index.htm

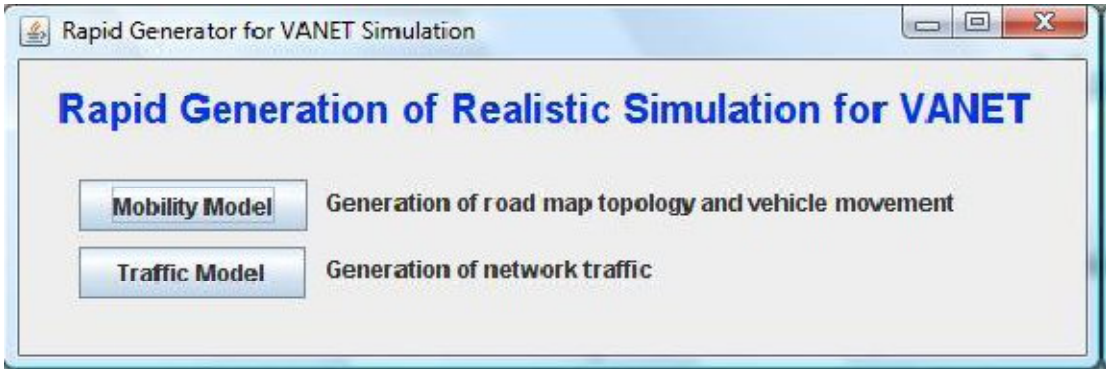
برای اجرا این نرم افزار باید نرم افزارهای مربوط به جاوا باید حتما دانلود شود. از انجا کشور ما تحریم می‌باشد نمی‌توان آن را از سایت جاوا مستقیما دانلود کرد. سایت www.p30download.com از قسمت نرم‌افزارهای برنامه‌نویسی دانلود کرد. Move با نرم‌افزار جاوا نوشته شده است. این بدان معنا است که بر روی سیستم‌عاملهای مختلف همانند لینوکس و ویندوز قابل پیاده‌سازی است. برای کامپایل کردن move ابتدا باید همانند نرم‌افزار micosoft Visual C++ طباید مسیرهای آن وارد سیستم شود. پس از ان در محیط متنی دستور زیر نوشته شود.

java -jar MOVE.jar

برای اجرای آن دستور زیر را وارد کنید.

java vanetsim

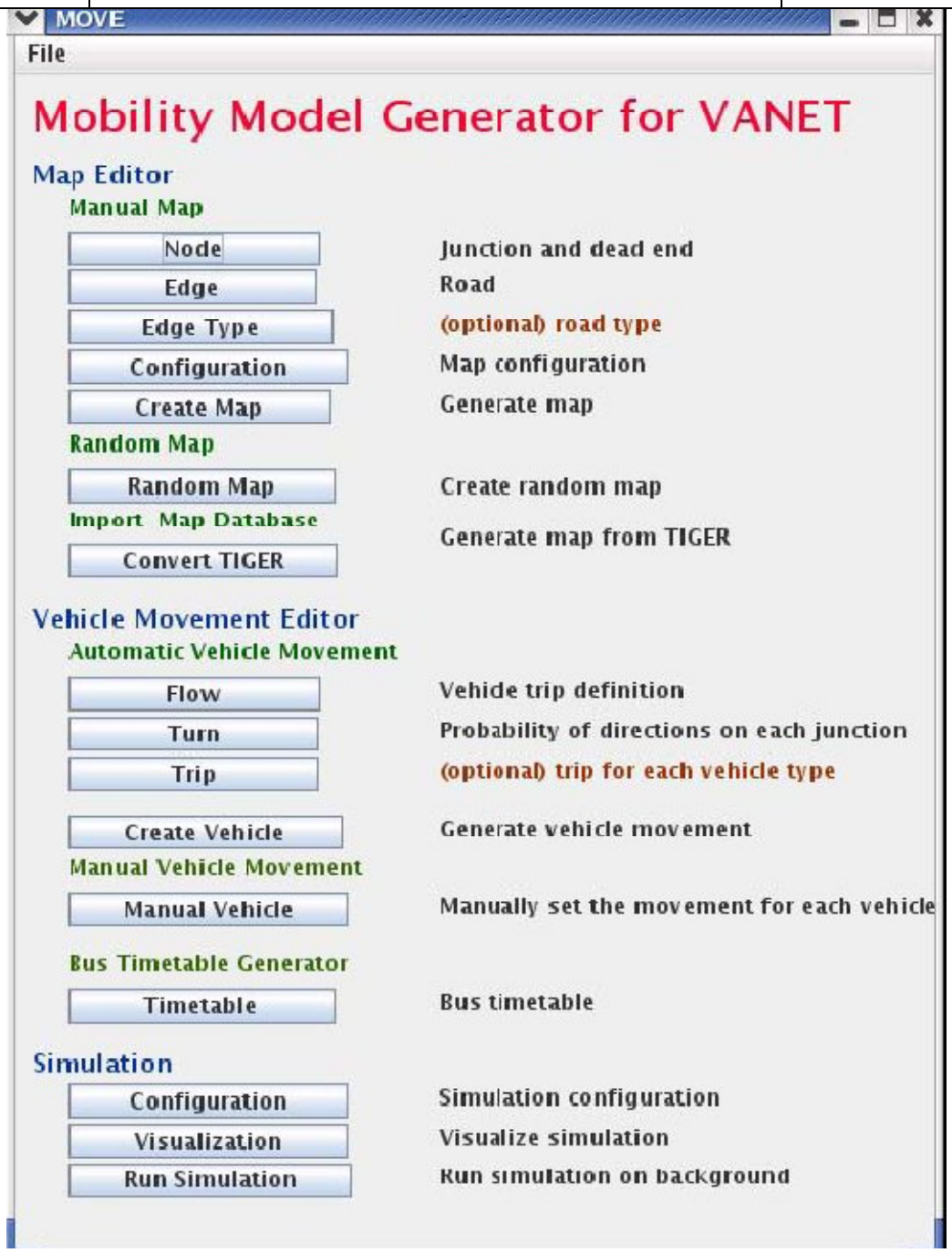
اگر نصب نرم‌افزار بدون مشکل باشد. بعد از اجرای دستور بالا شکل زیر ظاهر می‌شود.



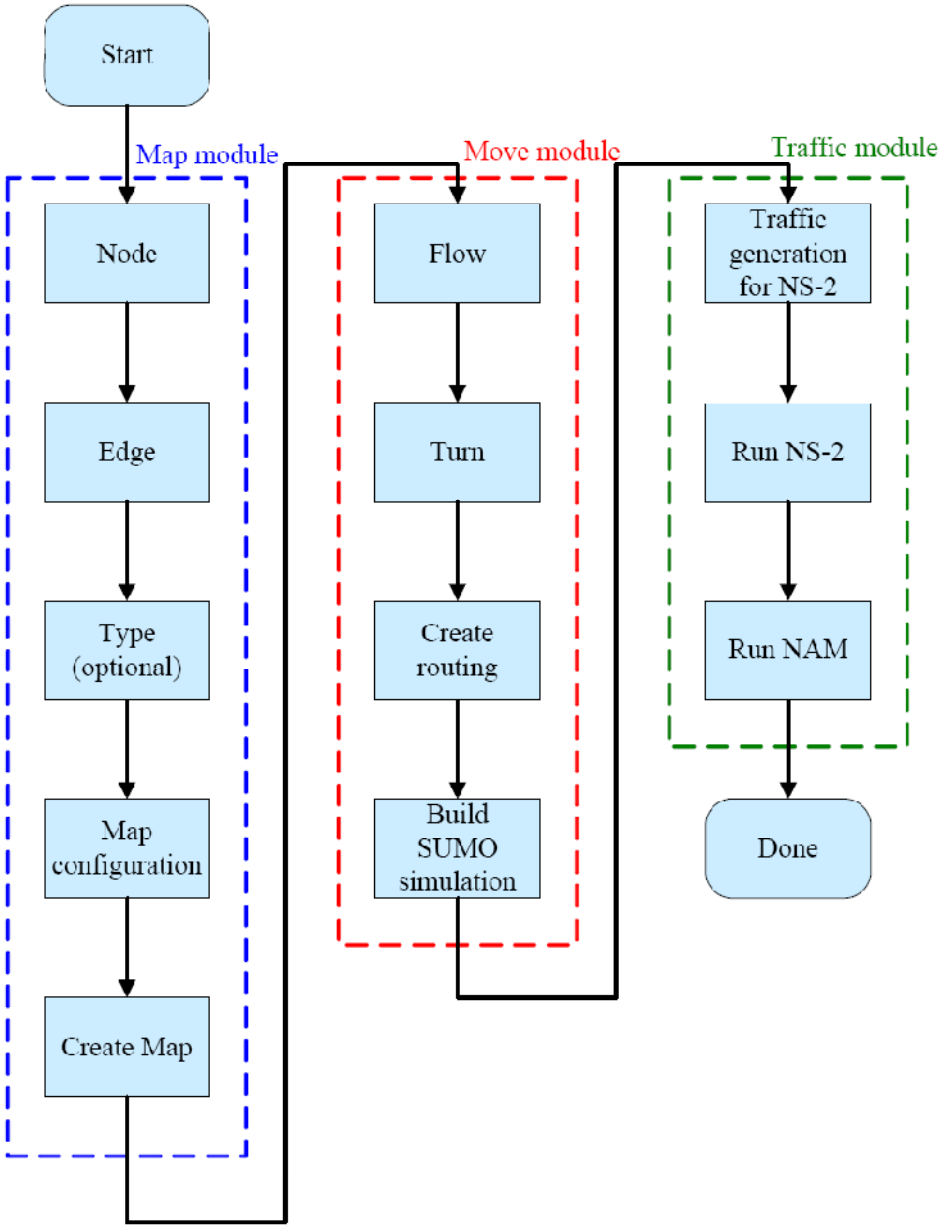
نرم افزار move واسط میان نرم‌افزار میان ns2 و sumo است. در ادامه به توضیح ان اشاره می‌شود.

۵-۴-۲- مراحل شبیه‌سازی

برای شبیه‌سازی VANET ، ابتدا باید مدل حرکتی را با استفاده از SUMO تولید می‌کنیم و سپس این کار را با واسط گرافیکی MOVE انجام داده می‌شود. شکل زیر مراحل انجام کار را با استفاده از ماژول‌های مختلف MOVE انجام می‌شود.



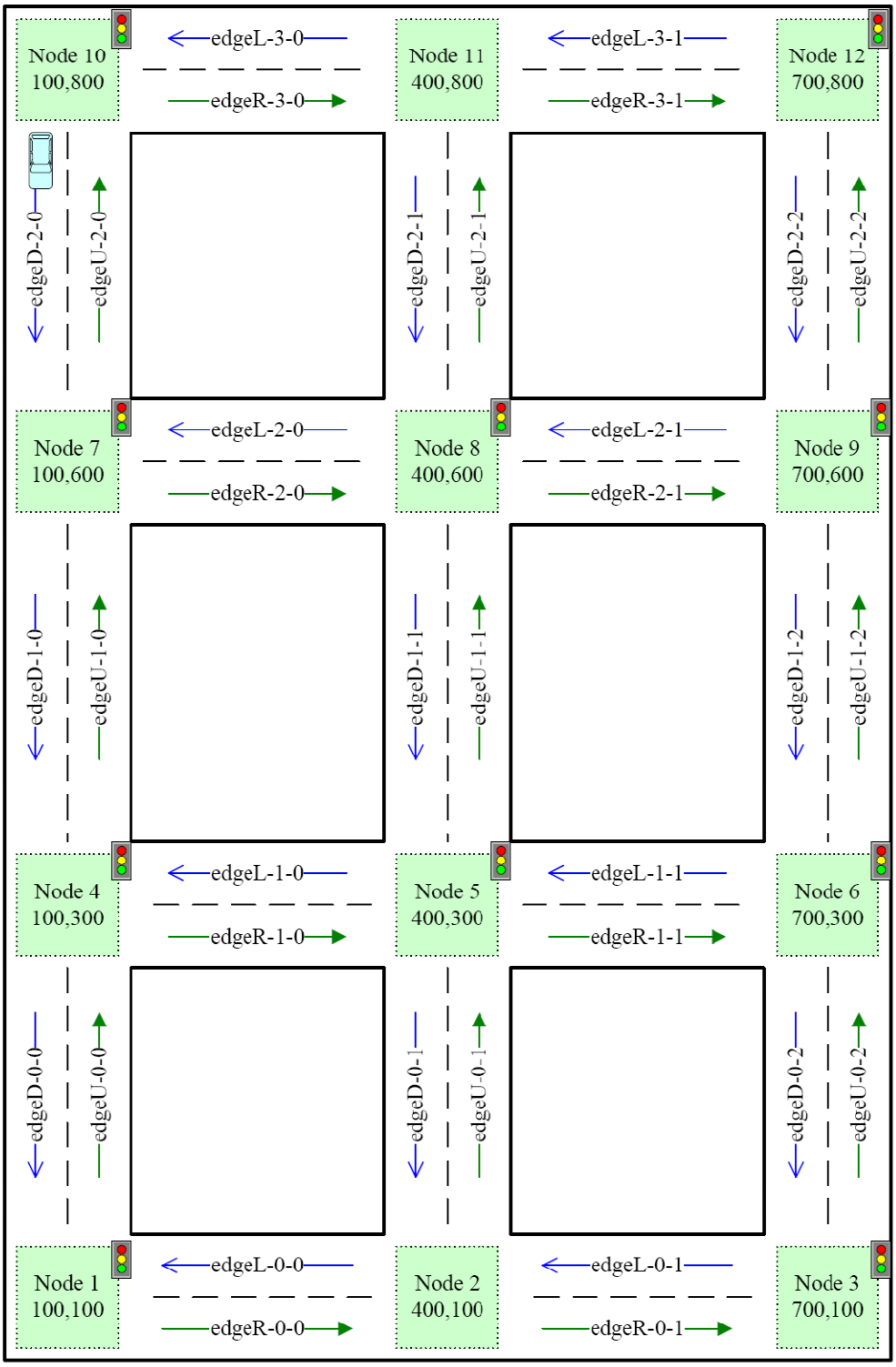
شکل بعد ماژولهای مختلف نرم‌افزار MOVE را نشان می‌دهد.



شکل ۴۲: ارتباط نرم‌افزار SUMO و NS2 و MOVE

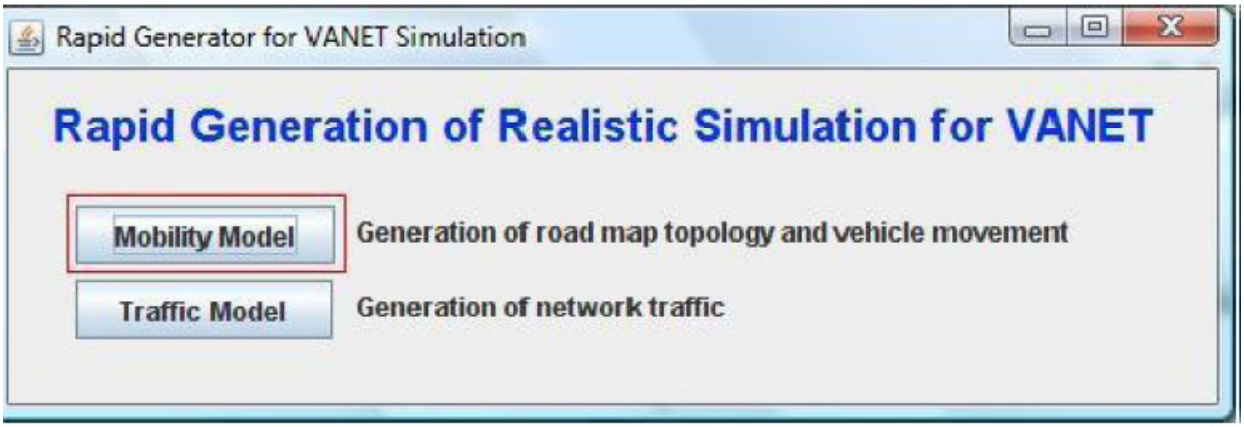
۵-۴-۳- پیاده‌سازی یک مثال

فرض کنید که می‌خواهیم مدل زیر را در شبکه پیاده‌سازی کنیم.



همانطور که در شکل ۴۲ نشان داده شده است. MOVE از دو بخش Mobility Model و Traffic Model تشکیل شده است. Mobility Model برای ایجاد مدل حرکتی مورد نیاز بکار می‌رود که این کار را

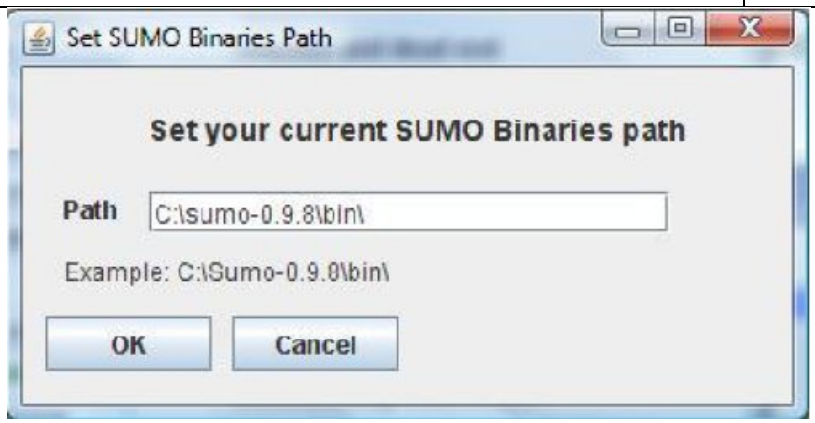
برای ارتباط با نرم‌افزار sumo انجام می‌شود. و منوی دوم Traffic Model مدل ترافیکی برای شبکه های نرم افزاری NS2 تولید و اجرا می‌کند.



در ابتدا Mobility Model انتخاب می‌شود . انتخاب Mobility Model برای کار با نرم افزار sumo استفاده می‌شود. ابتدا باید مسیر SUMO را برای آن مشخص شود. از منوی Set Environment سیستم عامل مورد نظر خود را انتخاب کنید.

شکل ۴۳: انتخاب سیستم عامل سیستم

توجه کنید اگر از سیستم ویندوز استفاده می‌کنید باید حتما مسیر نرم‌افزار sumo را باید انتخاب کنید که در شکل ۴۴ نشان داده شده است.

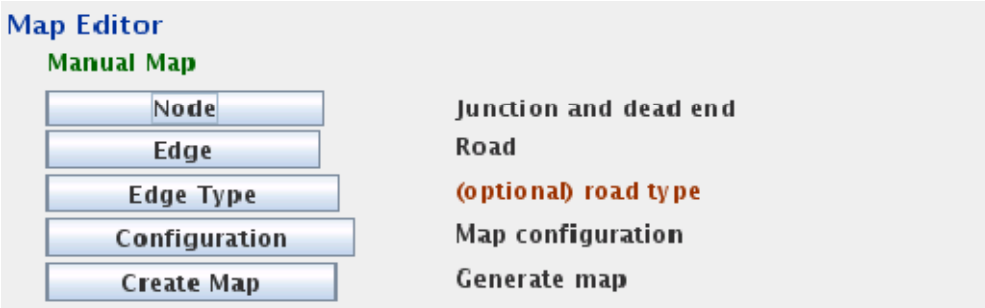


شکل ۴۴: مشخص کردن مسیر نرم‌افزار sumo در محیط نرم افزار move

در محیط لینوکس نیازی به وارد کردن مسیر ندارید. بعد از این مرحله شبیه‌ساز شما آماده است و می‌توانید نقشه شبکه را بکشید. در ادامه به شبیه‌سازی نرم‌افزار توسط شبیه‌سازmove پرداخته می‌شود.

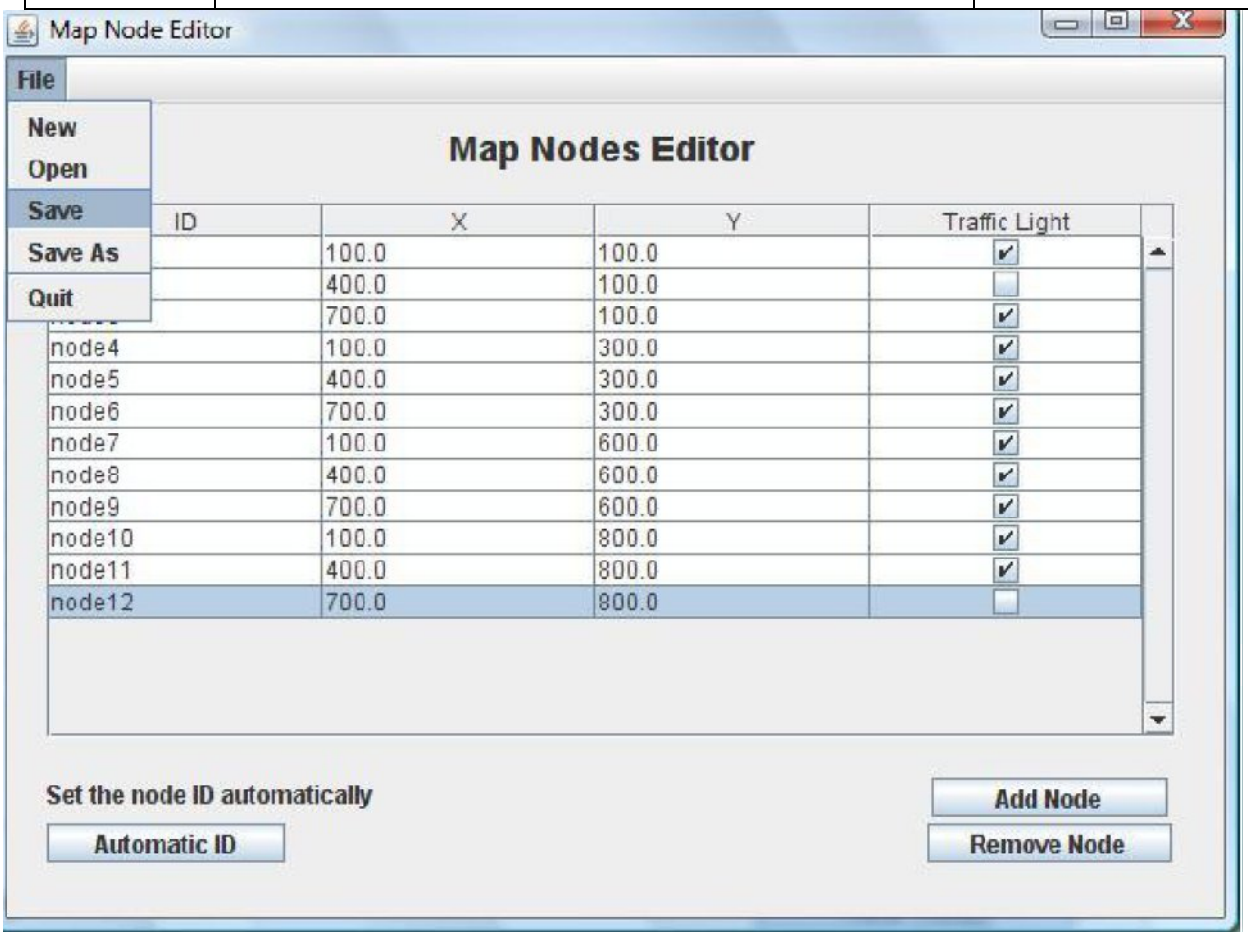
۵-۴-۴- تولید نقشه

۵-٤-٤-۱- ایجاد نقشه به صورت دستی



شکل ۴۵: ایجاد نقشه به صورت دستی

ابتدا بهتر است از منوی اصلی Node انتخاب شود. در این بخش می‌توان گره ها را به صورت دستی ایجاد کرد. بعد از انتخاب Node شکل ۴۵ را مشاهده می‌کنید.



شکل ۴۶: وارد کردن گرهها به صورت دستی

همانطور که در بخش قبل توضیح داده شده است. در نرم‌افزار SUMO هر گره دارای مختصات و شناسه است که در شکل () نشان داده شده است. گزینه Traffic Light نیز مشخص می‌کند که خودروها چه عکس‌العملی در پیچ‌ها داشته باشند. بعد از وارد کردن گره‌ها باید آن را در فایلی با پسوند node.xml. ذخیره کنند. (به عنوان مثال ex_Node.node.xml)

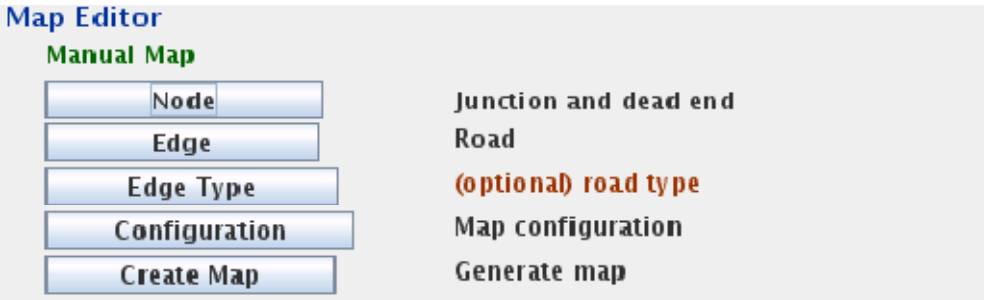
محتویات فایل حاوی گره‌ها در مثال ما در ادامه آمده است.

```
<nodes>
<node id="node1" x="100.0" y="100.0" type="traffic_light" />
<node id="node2" x="400.0" y="100.0" type="priority" />
<node id="node3" x="700.0" y="100.0" type="traffic_light" />
<node id="node4" x="100.0" y="300.0" type="traffic_light" />
<node id="node5" x="400.0" y="300.0" type="traffic_light" />
<node id="node6" x="700.0" y="300.0" type="traffic_light" />
<node id="node7" x="100.0" y="600.0" type="traffic_light" />
```

```
<node id="node8" x="400.0" y="600.0" type="traffic_light" />
<node id="node9" x="700.0" y="600.0" type="traffic_light" />
<node id="node10" x="100.0" y="800.0" type="traffic_light" />
<node id="node11" x="400.0" y="800.0" type="traffic_light" />
<node id="node12" x="700.0" y="800.0" type="priority" />
</nodes>
```

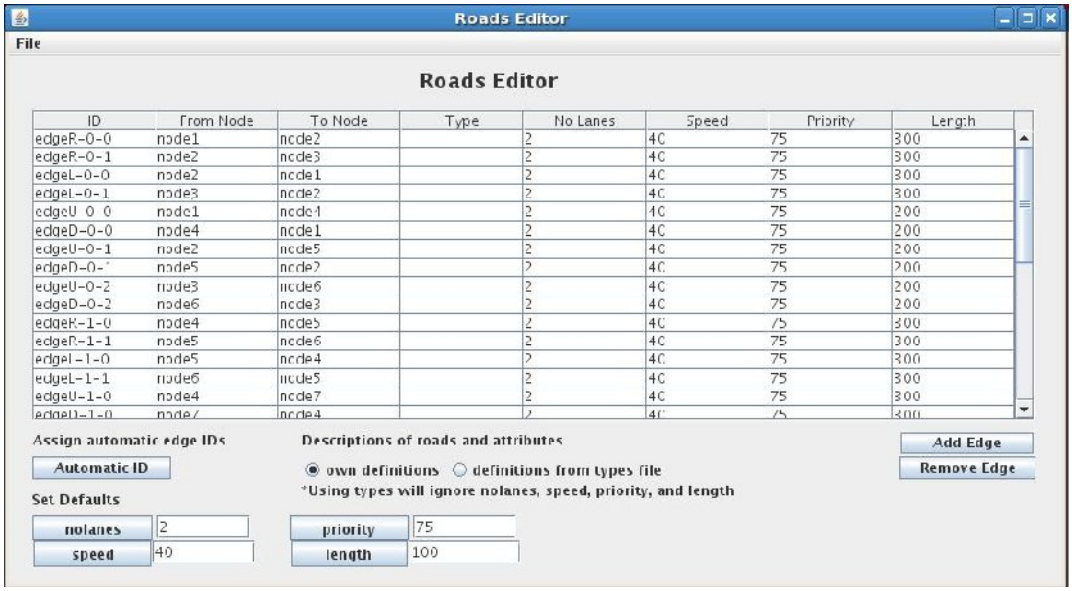
ایجاد دستی گره‌ها

مرحله بعد از تعیین گره‌ها، تعیین لبه‌ها می‌باشد. برای تعیین لبه‌ها می‌توان از منوی اصلی گزینه edge را انتخاب نمود.



شکل ۴۷: انتخاب منوی Edgeاز منوی اصلی

بعد از انتخاب edge از منوی اصلی پنجره زیر ظاهر می‌شود.



شکل ۴۸: پنجره ظاهر شده بعد از انتخاب edg

مشخصات فایل را در فایلی با پسوند edge.xml. (به عنوان مثال ex_EDGE_edg.xml) ذخیره کنید. جزییات مثال در زیر امده است.

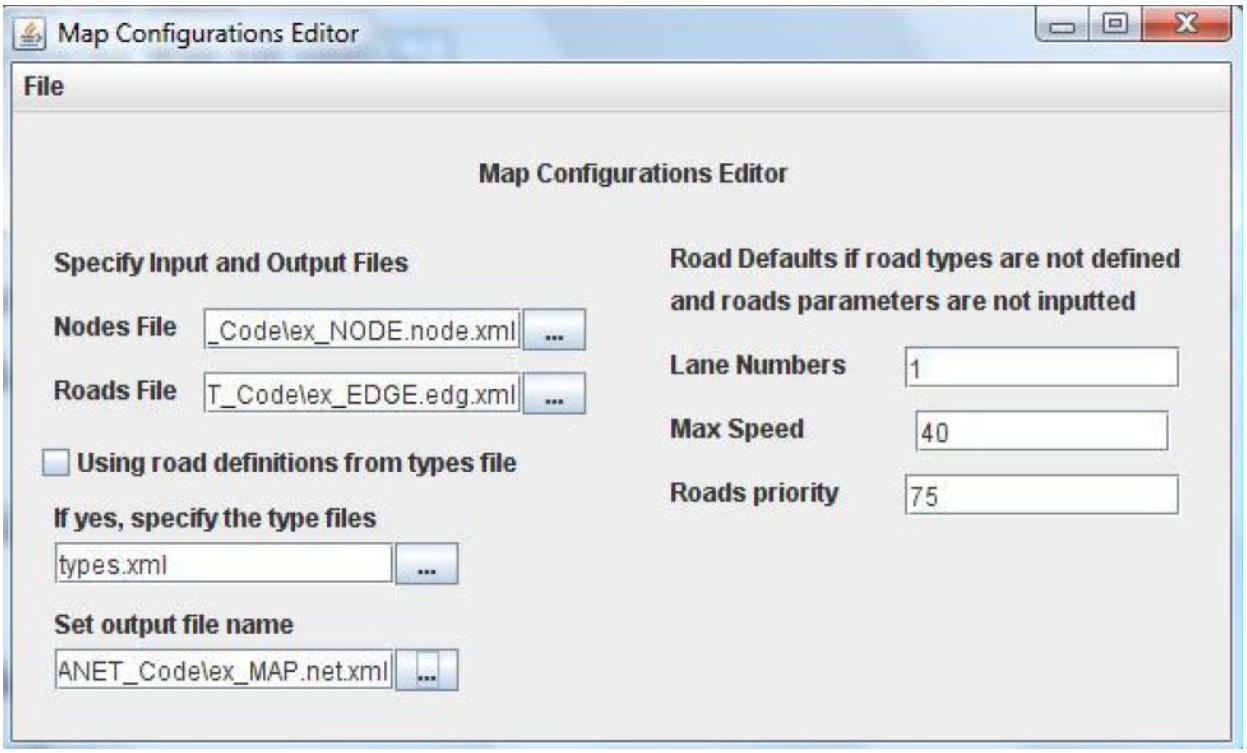
```
<edges>
<edge id="edgeR-0-0" fromnode="node1" tonode="node2" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeR-0-1" fromnode="node2" tonode="node3" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeL-0-0" fromnode="node2" tonode="node1" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeL-0-1" fromnode="node3" tonode="node2" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeU-0-0" fromnode="node1" tonode="node4" priority="75"
nolanes="2" speed="40" length="200"/>
<edge id="edgeD-0-0" fromnode="node4" tonode="node1" priority="75"
nolanes="2" speed="40" length="200"/>
<edge id="edgeU-0-1" fromnode="node2" tonode="node5" priority="75"
nolanes="2" speed="40" length="200"/>
<edge id="edgeD-0-1" fromnode="node5" tonode="node2" priority="75"
nolanes="2" speed="40" length="200"/>
<edge id="edgeU-0-2" fromnode="node3" tonode="node6" priority="75"
nolanes="2" speed="40" length="200"/>
<edge id="edgeD-0-2" fromnode="node6" tonode="node3" priority="75"
nolanes="2" speed="40" length="200"/>
<edge id="edgeR-1-0" fromnode="node4" tonode="node5" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeR-1-1" fromnode="node5" tonode="node6" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeL-1-0" fromnode="node5" tonode="node4" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeL-1-1" fromnode="node6" tonode="node5" priority="75"
nolanes="2" speed="40" length="300"/>
```

```
<edge id="edgeU-1-0" fromnode="node4" tonode="node7" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeD-1-0" fromnode="node7" tonode="node4" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeU-1-1" fromnode="node5" tonode="node8" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeD-1-1" fromnode="node8" tonode="node5" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeU-1-2" fromnode="node6" tonode="node9" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeD-1-2" fromnode="node9" tonode="node6" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeR-2-0" fromnode="node7" tonode="node8" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeR-2-1" fromnode="node8" tonode="node9" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeL-2-0" fromnode="node8" tonode="node9" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeL-2-1" fromnode="node9" tonode="node8" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeU-2-0" fromnode="node7" tonode="node10" priority="75"
nolanes="2" speed="40" length="200"/>
<edge id="edgeD-2-0" fromnode="node10" tonode="node7" priority="75"
nolanes="2" speed="40" length="200"/>
<edge id="edgeU-2-1" fromnode="node8" tonode="node11" priority="75"
nolanes="2" speed="40" length="200"/>
<edge id="edgeD-2-1" fromnode="node11" tonode="node8" priority="75"
nolanes="2" speed="40" length="200"/>
<edge id="edgeU-2-2" fromnode="node9" tonode="node12" priority="75"
nolanes="2" speed="40" length="200"/>
<edge id="edgeD-2-2" fromnode="node12" tonode="node9" priority="75"
```

```
nolanes="2" speed="40" length="200"/>
<edge id="edgeR-3-0" fromnode="node10" tonode="node11" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeR-3-1" fromnode="node11" tonode="node10" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeL-3-0" fromnode="node11" tonode="node12" priority="75"
nolanes="2" speed="40" length="300"/>
<edge id="edgeL-3-1" fromnode="node12" tonode="node11" priority="75"
nolanes="2" speed="40" length="300"/>
</edges>
```

جزییات لبه‌های تعیین شده

مرحله بعد، تعیین پیکربندی گره‌ها در نرم افزار SUMO است. مرحله بعد پیکربندی شبکه است باید بر روی گزینه CONFIGURATION کلیک کنید.



شکل(). پنجره باز شده بعد از کلیک کردن بر روی گزینه configuration

در این مرحله پیکربندی‌های لازم را برای تولید نقشه (MAP) انجام می‌دهیم. ادرس دو فایل حاوی گره‌ها و لبه‌ها و همچنین ادرس فایل نقشه (ex_MAP.net.xml) که باید در مرحله بعد تولید شود. سپس آن را با فایل netc.cfg. ذخیره می‌کنیم. (در این مثال ex_MAP.netc.cfg)

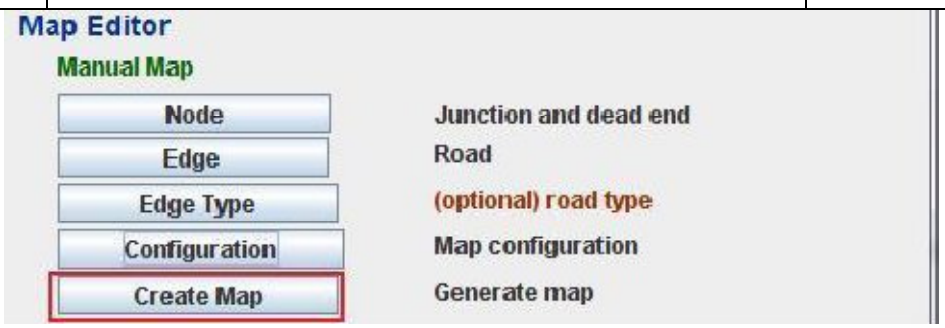
جزییات فایل ex_MAP.netc.cfg مثال در ادامه امده است.

```
<configuration>
<files>
<xml-node-files>D:\Simulation\VANET_Code\ex_NODE.node.xml</xml-node-files>
<xml-edge-files>D:\Simulation\VANET_Code\ex_EDGE.edg.xml</xml-edge-files>
<xml-connection-files />
<type-file />
<output-file>D:\Simulation\VANET_Code\ex_MAP.net.xml</output-file>
</files>
<defaults>
<type>Unknown</type>
<lanenumber>1</lanenumber>
<speed>40</speed>
<priority>75</priority>
<capacity-norm />
</defaults>
<reports>
<verbose />
<print-options />
</reports>
<process>
<remove-geometry>x</remove-geometry>
</process>
</configuration>
```

ex_MAP.netc.cfgخروجی فایل

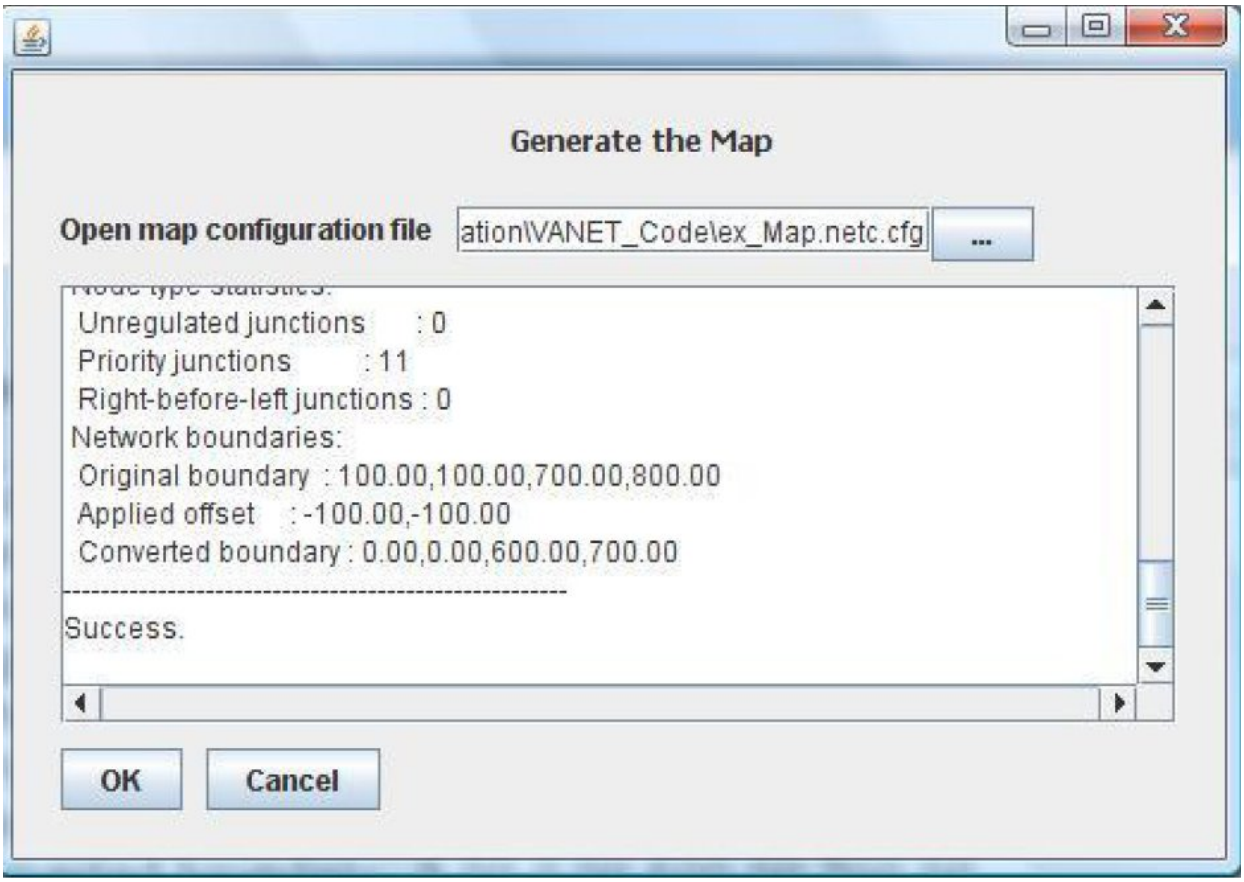
۵-٤-٤-۲- تولید نقشه

در این مرحله نوبت به ایجاد نقشه ای که در مرحله قبل پیکربندی کردیم انجام می‌دهیم. بدین منظور بر روی create Map کلیک می‌کنیم.



شکل ۴۹: تولید نقشه

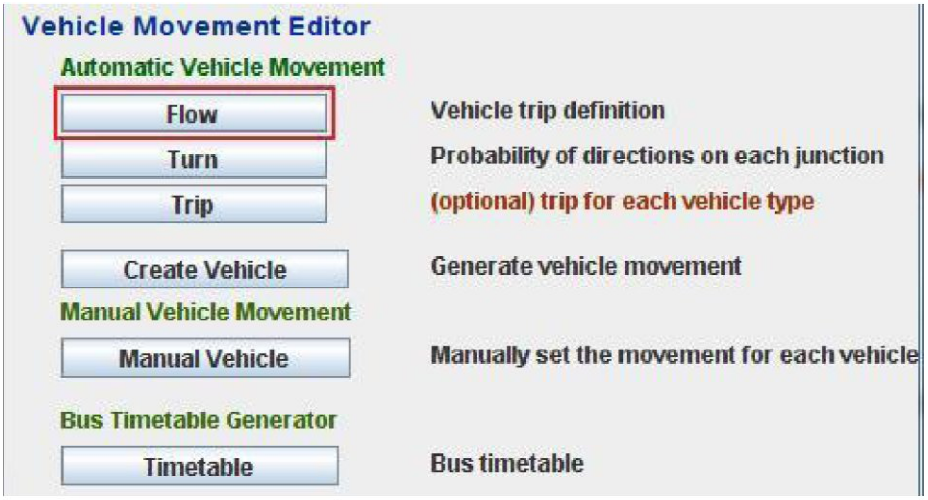
مسیر فایل ex_MAP.netc.cfg را که در مرحله قبل ایجاد شده است را وارد می‌کنیم و بر روی دگمه ok کلیک می‌شود. فایلی با پسوند net.xml. (ex_MAP.net.xml) تولید خواهد شد که فایل نقشه مورد نظر است.



۵-۴-۵- تولید حرکت خودروها

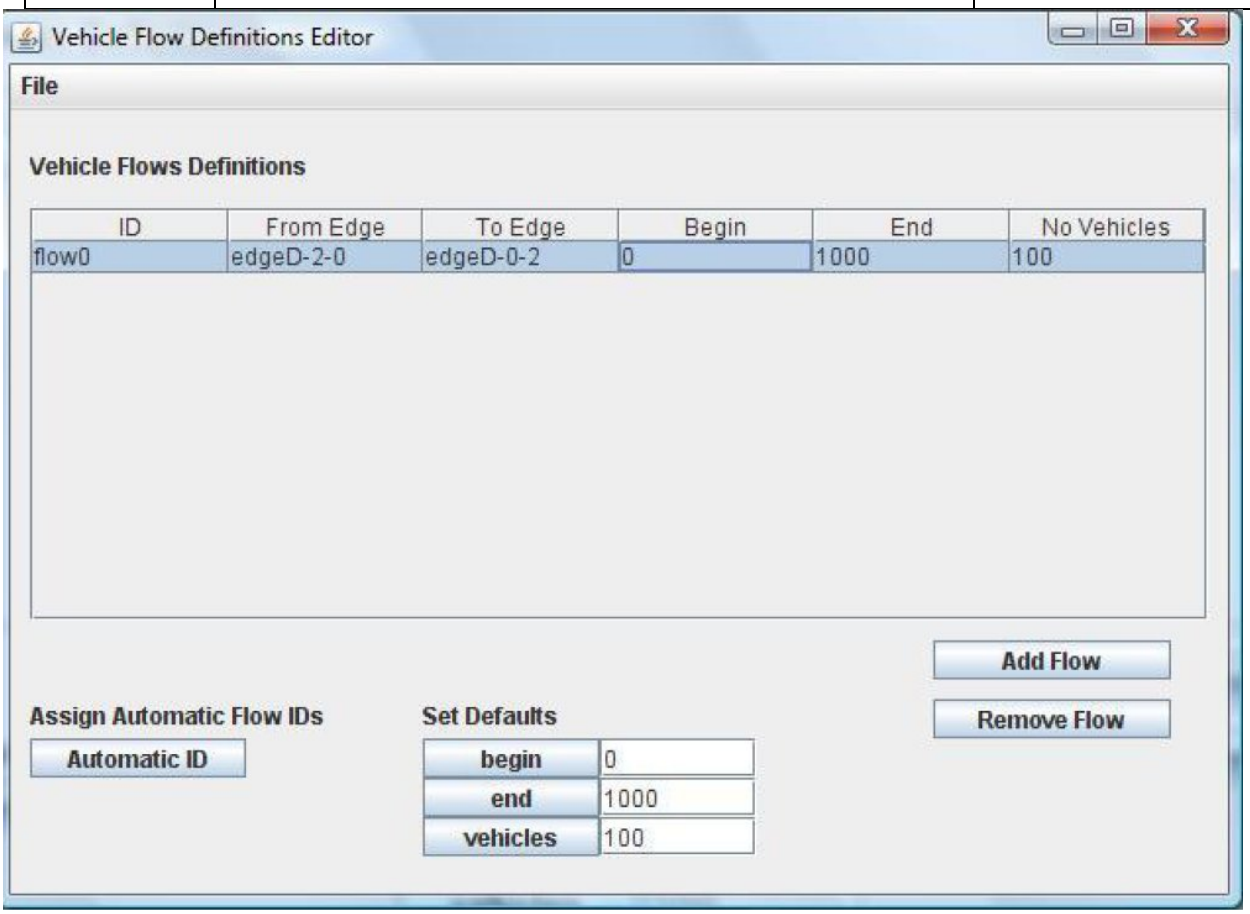
۵-۴-۶- تعریف جریان

برای تعیین جریان حرکت خودروها بر روی Flow کلیک کنید.



شکل ۵۰: پنجره ظاهر شده برای تعریف جریان

بعد از کلیک بر روی flow ، پنجره زیر ظاهر می‌شود.



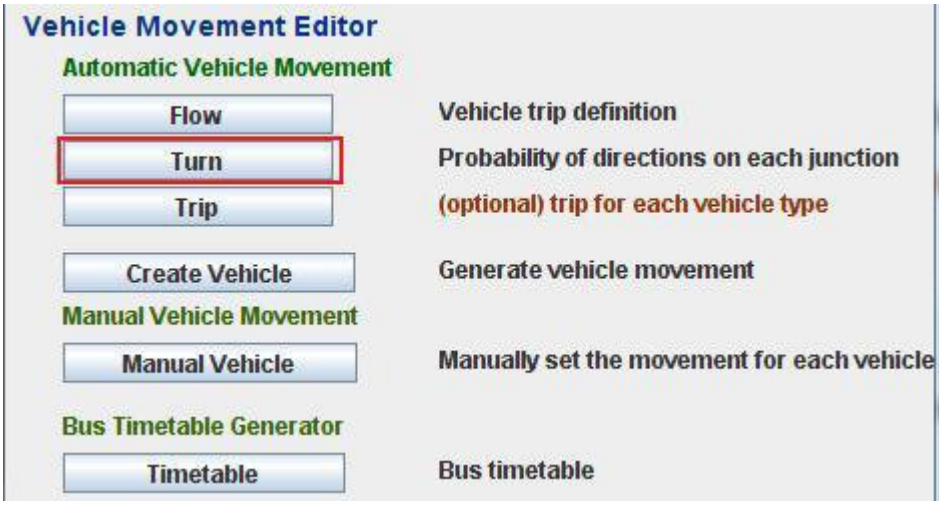
شکل ۵۱: پنجره ظاهر شده پس از کلیک کردن بر روی flow

در این مثال حرکت گروهی خودروها از لبه D-2-0 به لبه D-0-2 است. در تعریف جهت حرکت گره‌ها باید توجه کرد که اگر گره می‌خواهد یک خودرو را از گره ۱ به گره ۲ و گره ۳ حرکت دهد. باید جریان را از لبه ۱ به ۳ تعریف کرد. اگر حرکت خودرو از لبه ۱ به ۲ تعریف شود. نصف راه را حرکت خواهد کرد. تنظیمات را در فایل ex_FLOW.flow.xml ذخیره می‌شود.

```
<flows>
<flow id="flow0" from="39dged-2-0" to="39dged-0-2" begin="0" end="1000" no="100"
/>
</flows>
```

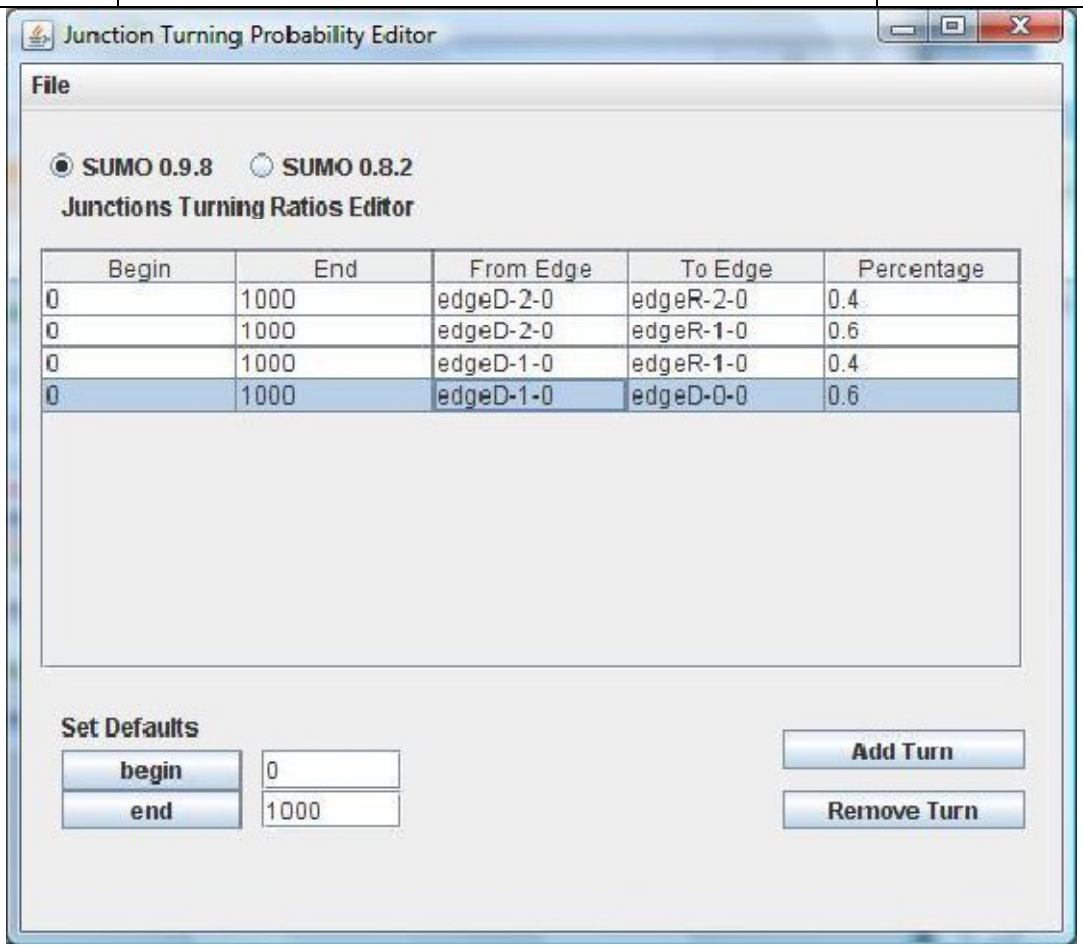
۵-۴-۷- تعیین دور

برای تعیین دور باید از منوی اصلی Turn انتخاب می‌شود.



شکل ۵۲: تعیین دور در منو اصلی

در این مرحله دور و سرعت آن ها را مشخص می کنیم.



شکل ۵۳: پنجره ظاهر شده پس از کلیک بر روی Turn

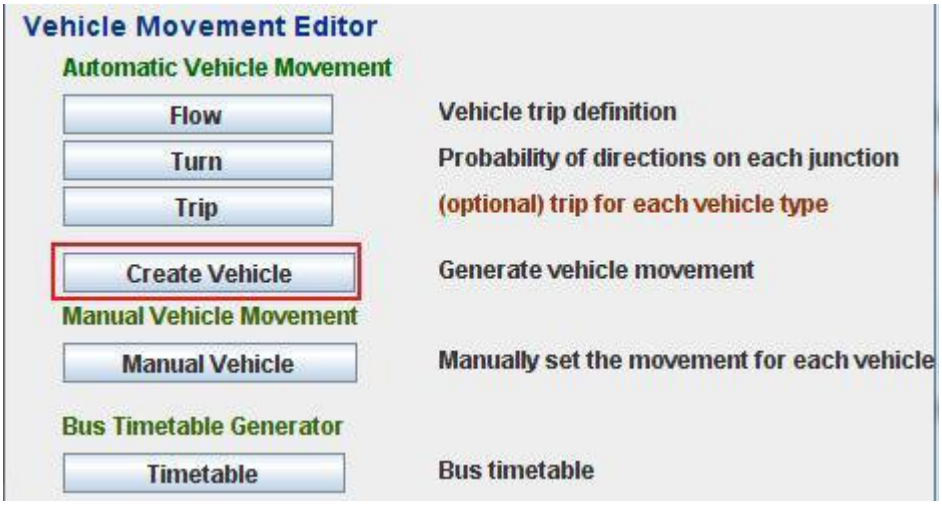
آنرا در فایل ec_TURN.turn.xml ذخیره می‌شود.

```
<turn-defs>
<interval begin="0" end="1000">
<fromedge id="40dged-2-0">
<toedge id="40dged-2-0" probability="0.4" />
<toedge id="40dged-1-0" probability="0.6" />
</fromedge>
<fromedge id="40dged-1-0">
<toedge id="40dged-1-0" probability="0.4" />
<toedge id="40dged-0-0" probability="0" />
</fromedge>
</interval>
</turn-defs>
```

فایل خروجی ec_TURN.turn.xml

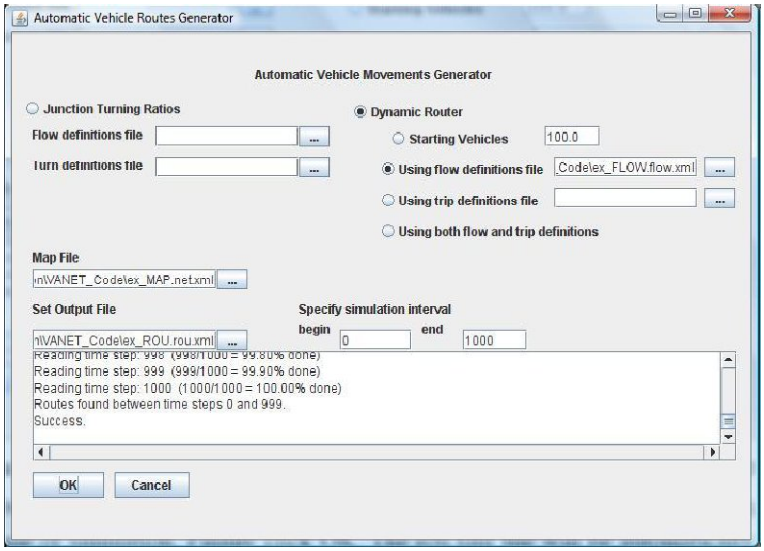
۵-۴-۸- حرکت خودکار خودروها

برای حرکت خودکار خودروها بر روی create vehicle کلیک کنید.



شکل ۵۴: حرکت خودکار خودروها

این مرحله گروهی از خودروها را در آغاز شبیه سازی ایجاد می کند.

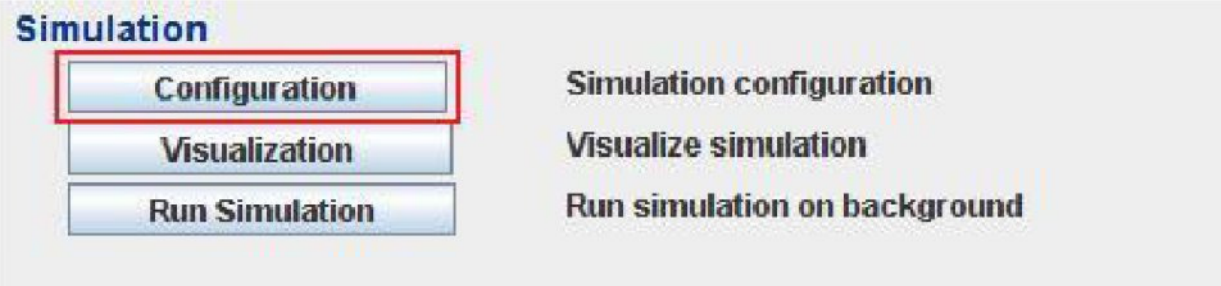


شکل ۵۵: پنجره ظاهر شده پس از کلیک بر روی create vehicle

مسیر فایلهای جریان ex.FLOW.flow.xml و نقشه ex_MAP.net.xml و همچنین مسیر فایل خروجی مسیرها ex_ROU.rou.xml را مشخص می‌کند.

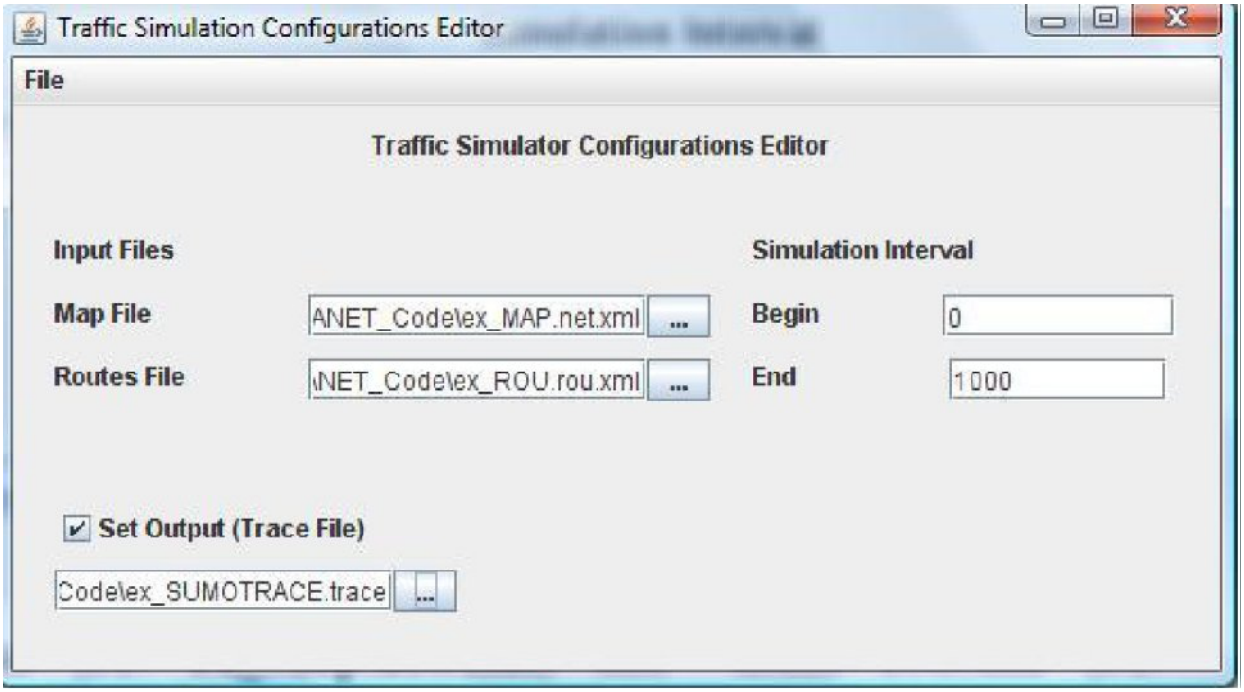
۵-۴-۹- تنظیمات شبیه‌سازی

در مرحله بعد به تنظیمات شبیه‌سازی اشاره می‌شود. بر روی گزینه configuration کلیک کنید.



شکل ۵۶: کلیک بر روی گزینه configuration

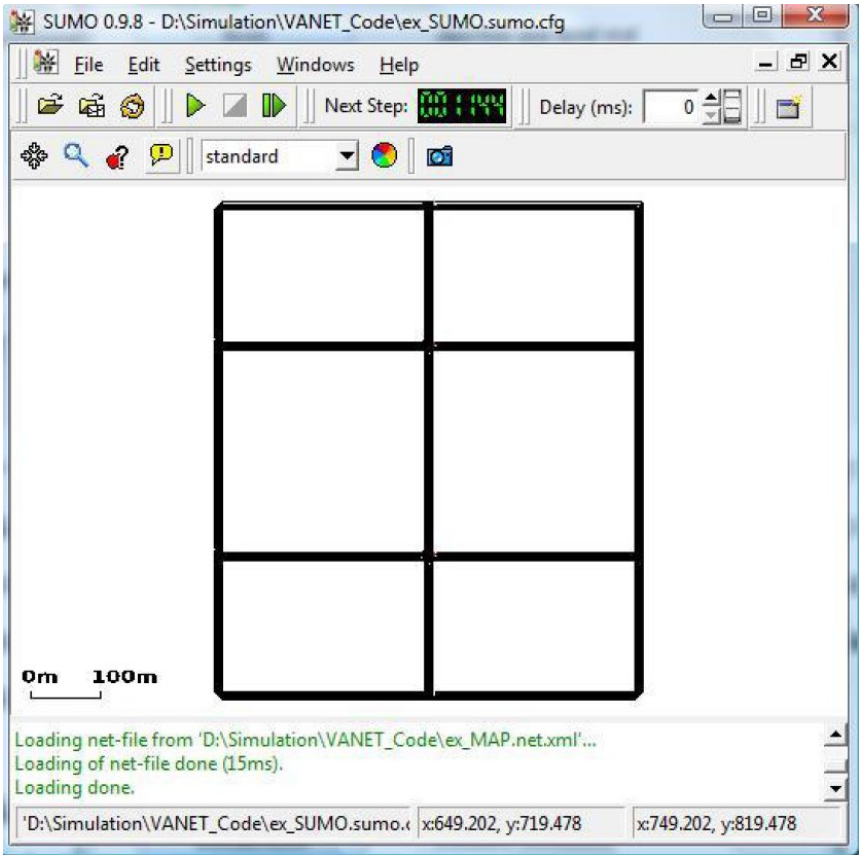
در این مرحله فایل مورد نظر sumo را تولید می‌کند. فایل‌های ex.MAP.net.xml و ex_ROU.rou.xml را وارد کنید. اگر به فایل trace در SUMO نیاز دارید. گزینه ان را انتخاب کنید و مسیر خروجی را به آن بدهید. (ex_SUMOTRACE.tr) . سپس تنظیمات را در فایل ex_SUMO.sumo.cfg ذخیره کنید. پس از کلیک بر روی configuration پنجره زیر را می بینید.



شکل ۵۷: پنجره ظاهر شده پس از کلیک بر روی configuration

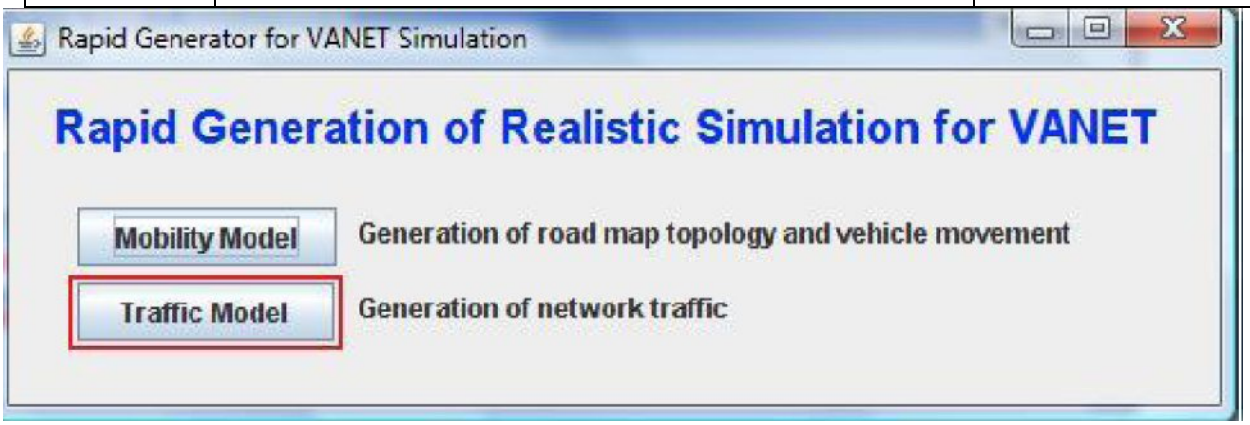
پس از این مرحله Visualization می‌شویم بر روی گزینه Visualization کلیک کنید.

بعد از کلیک بر روی Visualization ، برنامه SUMO اجرا می‌شود و فایل ex.SUMO.sumo.cfg در محیط گرافیکی ایجاد می‌شود.



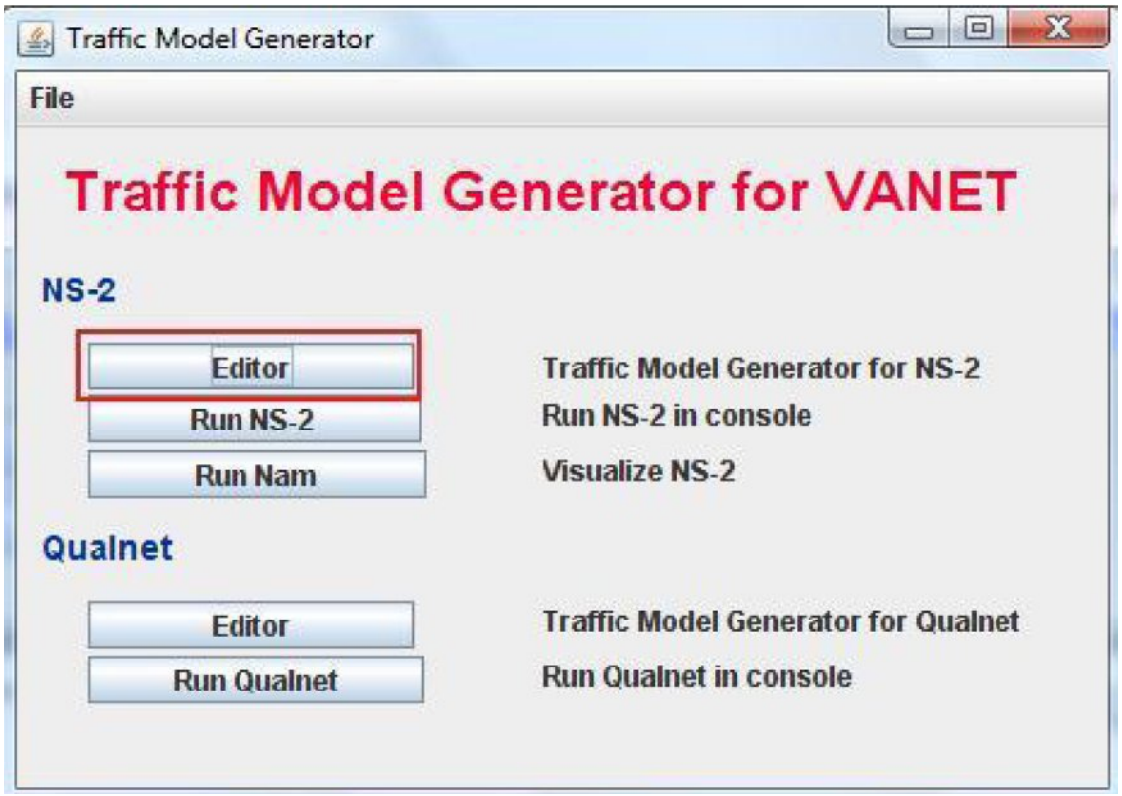
شکل ۵۸: پنجره ظاهر شده پس از کلیک بر روی Visualization

در مرحله اخر مدل ترافیکی تولید می‌شود و گزینه Traffic Model انتخاب می‌شود.



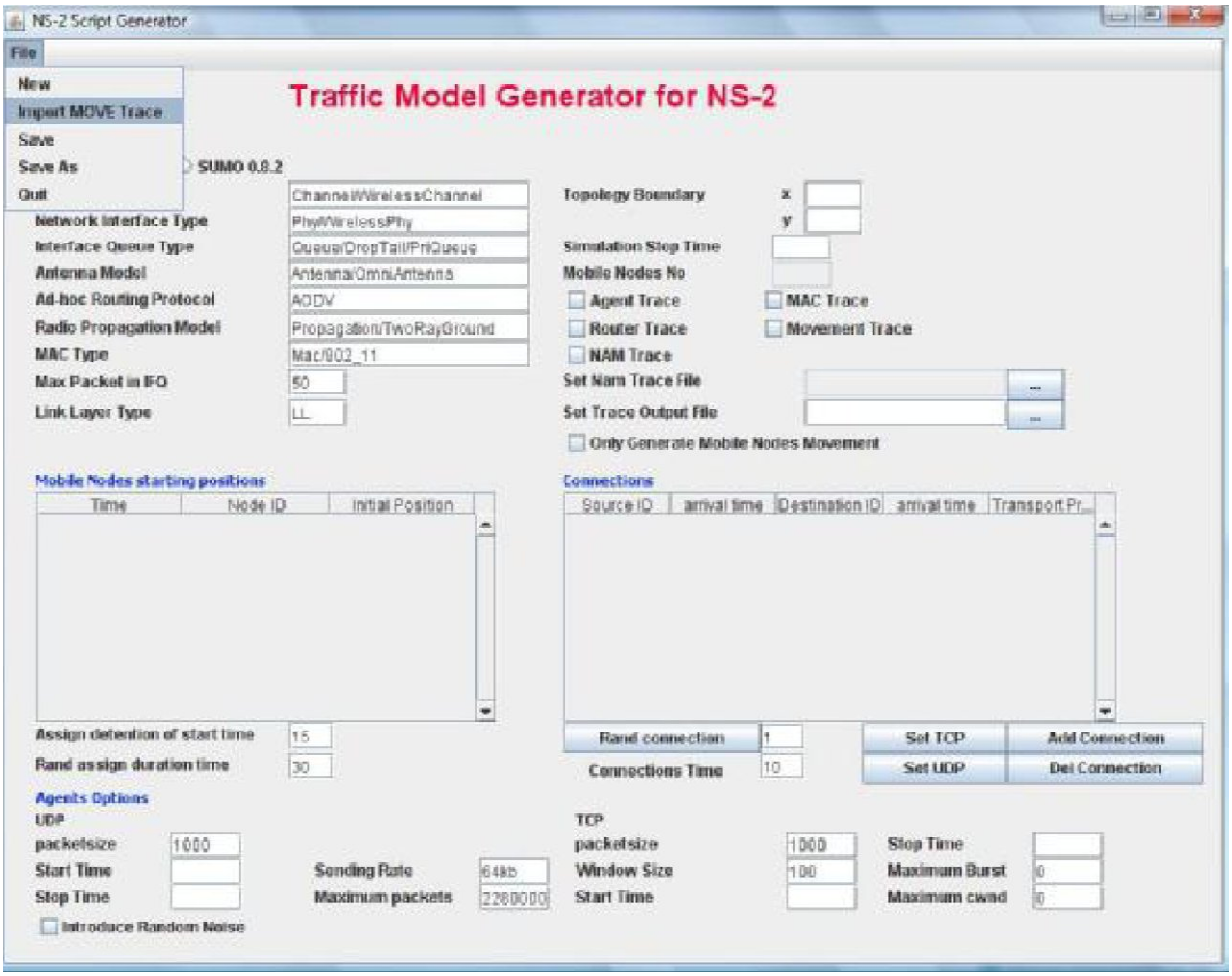
شکل ۵۹: انتخاب برای مدل ترافیک

گزینه traffic model دارای دو بخش اصلی برای نرم افزار NS2 و Qualnet است که فقط به شبیه‌ساز ns2 اشاره می‌شود.



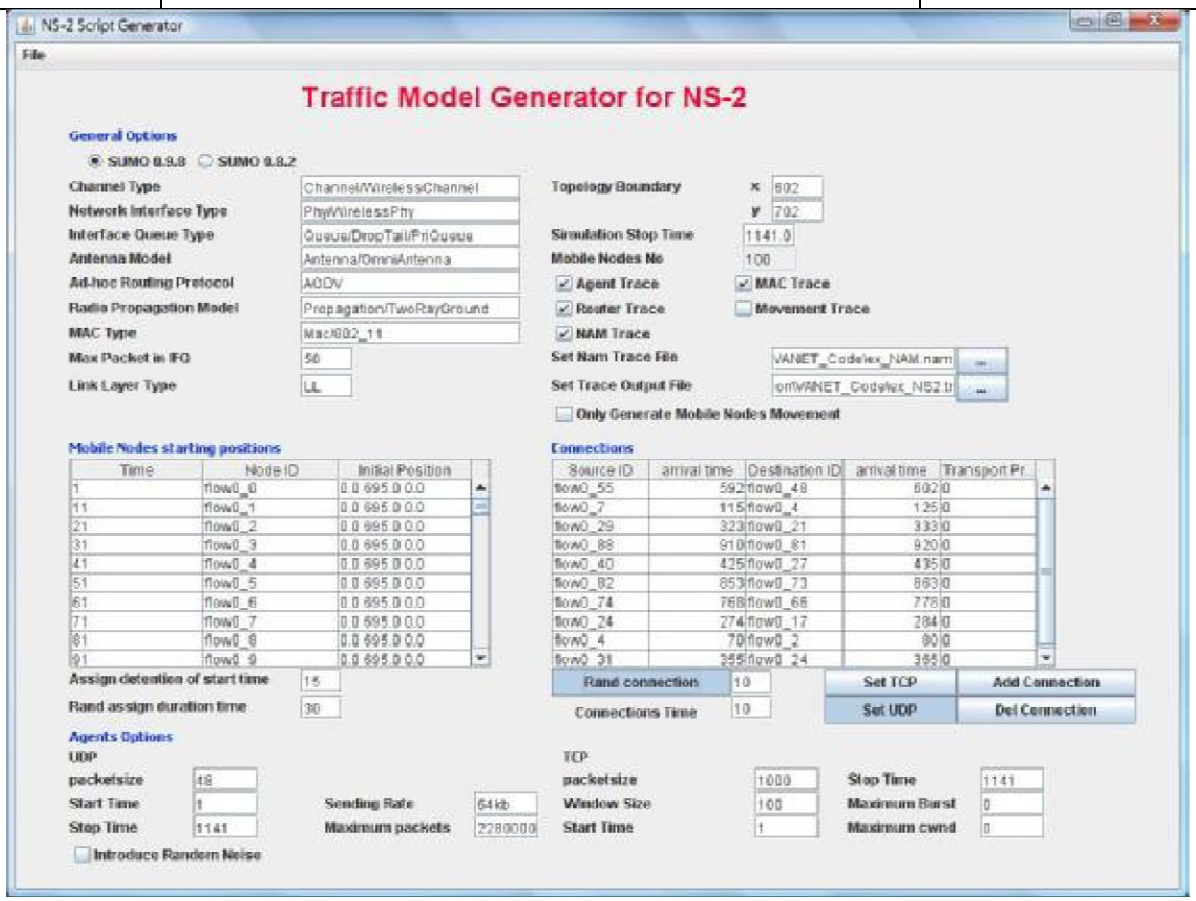
شکل ۶۰: مدل ترافیک نرم افزار ns2

این بخش فایل شبیه‌سازی ترافیکی یک فایل Tcl را برای ns2 ایجاد می‌شود. ابتدا فایل trace (ex_SUMOTRACE.trace) را برای SUMO که در مرحله قبل تولید شده است تولید می‌کنیم و پس از ان فایل نقشه (ex_MAP.net.xml) را وارد می‌کنیم. بسته به حجم فایل و سرعت کامپیوتر باید مدتی صبر کنید تا اطلاعات بار شود.



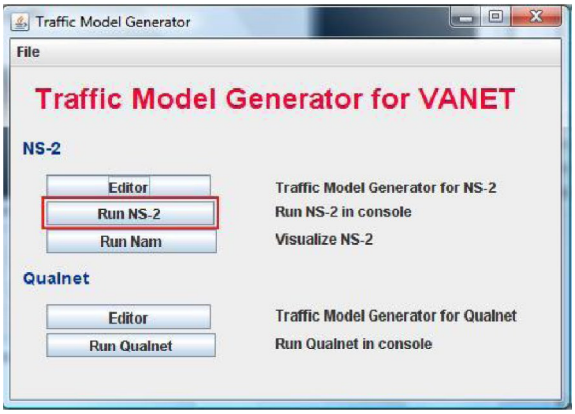
شکل ۶۱: خروجی برای ns2 تولید کردن

بعد از بارگذاری، گزینه مربوط به شبیه‌سازی ns2 را مشخص می‌کنیم. به عنوان مثال فایل trace را برای nam مشخص می‌کنیم و ارتباط بین گره‌ها و نوع آن tcp و udp را مشخص می‌کنیم.



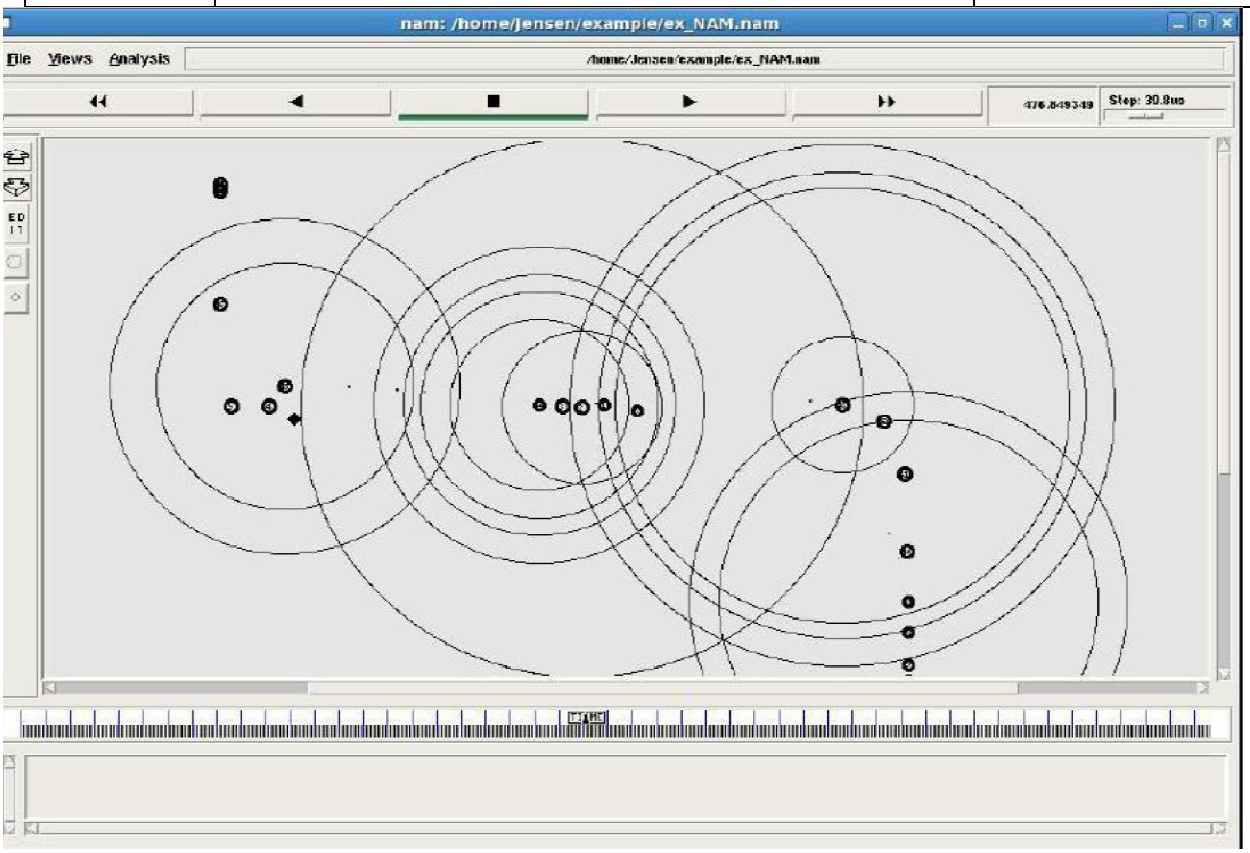
شکل ۶۲: وارد کردن اطلاعات مربوط به شبیه‌ساز ns2

بعد از انجام تنظیمات ان را با پسوند tcl. (ex_NS.tcl) ذخیره کنید. ان را از طریق NS2 اجرا کنید.



شکل ۶۳: وارد کردن اطلاعات در NS2

می‌توانید خروجی ان را در NAM ببینید.



شکل ۶۴: خروجی در شبیه‌ساز NAM

منابع و مراجع

- http://www.isi.edu/nsnam/ns/
- http://nile.wpi.edu/NS/
- http://www.isi.edu/nsnam/nam/
- http://vanet.info/node/13
- Marco Fiore,"Mobility Models in Inter-Vehicle Communications Literature",Technical Report, November 2006.
- Vehicular Ad hoc Networks, Master's Thesis in Computer Science Rainer Baumann, ETH Zurich 2004
- Christoph Sommer and Falko Dressler, The DYMO Routing Protocol in VANET Scenarios, IEEE 2007