www.NoavaranGermi.ir

به نام خدا

گزارش سمینار سیستم عامل پیشرفته

معرفی نرم افزار OMNeT++

استاد : جناب آقای دکتر بابامیر

ارائه شده توسط : حامد قدیریان[۱]

دانشگاه کاشان

[۱] hamed.ghadirian۶۸@yahoo.com

www.NoavaranGermi.ir

۱- معرفی OMNeT++ ۱

-۱-۱ مدل : ۱

۱-۲- انواع ماژول ها : ۲

۱-۳- پیام ها ۲

۱-۴- گیت ها و کانکشن ها(Gates & Connections) ۲

۱-۵- پارامترها ۳

۱-۶- برنامه نویسی الگوریتم ها ۳

۱-۷- سیستم شبیه سازی ۳

۱-۸- نحوه ساخته شدن فایل اجرایی شبیه سازی ۴

۱-۹- اجرای فایل اجرایی و تجزیه و تحلیل آن ۴

۱-۱۰- زبان NED ۴

۱-۱۰-۱- اجزای زبان NED: ۴

۱-۱۰-۲- دستورات import : ۵

۱-۱۰-۳-Identifier : ۵

-۲ مقایسه OMNeT++ و OMNEST(نسخه ی تجاری این نرم افزار) ٦

-۳ نصب نرم افزار OMNeT++ ۹

۳-۱- نحوه نصب در سیستم عامل های ویندوز: ۹

-۳-۲ تنظیم و ساخت OMNeT++ ۹

۳-۳- بررسی درستی نصب ۱۰

۳-۴- اجرای IDE: ۱۱

-٤ بررسی مثال عملی شبیه سازی TicToc ۱۲

۴-۱- شروع به کار ۱۲

-۱-۱-۴ قدم ۱: پیاده سازی اولیه ۱۲

۴-۲- ارتقا و بهبود مدل دو گره ای TicToc ۱۷

-۱-۲-۴ قدم ۲: بهبود گرافیک و افزودن خروجی debug ۱۷

www.NoavaranGermi.ir

۴-۲-۲- قدم ۳: افزودن متغییرهای وضعیت ۱۹
۴-۲-۳- قدم ۴: افزودن پارامترها ۲۱
۴-۲-۴- قدم ۵: استفاده از ارث بری ۲۳
۴-۲-۵- قدم ۶: مدلسازی تاخیر پردازش ۲۴
۴-۲-۶- قدم ۷ : شماره های تصادفی و پارامترها ۲۵
۴-۲-۷- قدم ۸: لغو تایمرها و اتمام زمان ۲۶
۴-۲-۸- قدم ۹: ارسال دوباره ی یک پیام ۲۸
۴-۳- تبدیل به یک شبکه ی واقعی ۲۸
۴-۳-۱- قدم ۱۰ : بیش از دو گره ۲۸
۴-۳-۲- قدم ۱۱: کانال ها و تعریف های نوع داخلی ۳۱
۴-۳-۳- قدم ۱۲: استفاده از اتصالات دو طرفه ۳۲
۴-۳-۴- قدم ۱۳: تعریف کلاس پیام ۳۳
۴-۴- افزودن مجموعه های آماری ۳۶
۴-۴-۱- قدم ۱۴: نمایش شماره پکت های ارسالی/دریافتی ۳۶
۴-۴-۲- قدم ۱۵: افزودن مجموعه های آماری ۳۹
۴-۵- نمایش تصویری نتایج به وسیله ی OMNeT++ IDE ۵۳
۴-۵-۱- نمای ویژوال خروجی بردارها و اعداد اسکالر بدست آمده ۵۳
۵- Directed diffusion در شبکه های حسگر بیسیم ٦۲
۵-۱- مقدمه ۶۲
۵-۲- خانواده پروتکل directed diffusion ۶۳
۵-۳- Two-Phase Pull Diffusion ۶۳
۵-۴- شبیه سازی directed diffusion در OMNeT++ ۶۷
٦- منابع و ماخذ: ٦۹

www.NoavaranGermi.ir

۱- معرفی OMNeT++

این برنامه یک شبیه ساز شبکه است که برای موارد زیر بکار می رود:

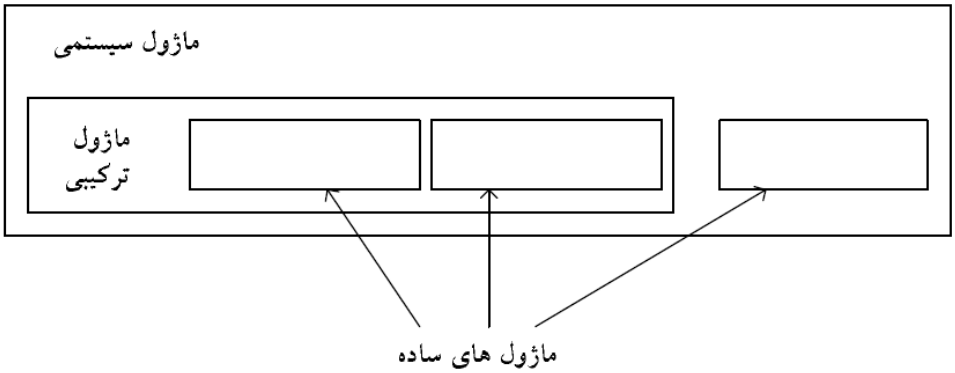
- **مدلسازی ترافیک شبکه**
- **مدلسازی پروتکل**
- **مدلسازی شبکه های صف بندی**
- **مدلسازی میکروپروسسور و سایر سیستم های سخت افزاری**
- **ارزیابی کردن چگونگی اجرای سیستم‌های نرم افزاری پیچیده**

OMNeT++ ابزارهای سودمندی برای توصیف ساختار یک سیستم واقعی در اختیار کاربر میگذارد. بعضی از مفاهیم اصلی آن عبارتند از:

- **ماژول‌ها می توانند تو در تو باشند.**
- **ماژول‌ها به وسیله پیام ها ارتباط برقرار می کنند.**
- **ماژول‌ها پارامترهای قابل انعطاف دارند.**
- **زبانی برای توصیف توپولوژی ها**

۱-۱- مدل :

یک مدل در OMNeT++ شامل یک سری ماژول می باشد که به صورت گرافیکی نشان داده می شوند و هر ماژول می تواند خود شامل ماژول دیگری نیز باشد. تعداد ماژولهایی که می‌توانند زیرمجموعه یک ماژول باشند نامحدود است و به وسیله همین امکان ما می توانیم ساختار یک سیستم واقعی را به صورت گرافیکی خیلی بهتر نشان دهیم. شکل زیر نشانگر ماژول ها می باشد :



شکل ۱

www.NoavaranGermi.ir

۱-۲- انواع ماژول ها :

ماژول مرکب : ماژولی است که شامل زیرماژول های دیگر می باشد.

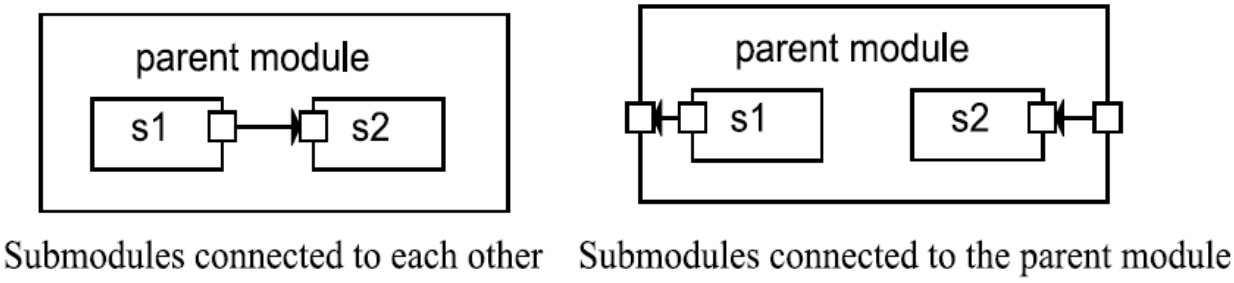
ماژول ساده : همان زیر ماژول ها هستند. در این ماژولها الگوریتم مدل را تعریف می کنیم که با زبان C++ انجام داده می شود و از کتابخانه کلاس های OMNeT++ استفاده می کند.

۱-۳- پیام ها

در یک مدل ماژولها به وسیله پیام ها با هم ارتباط برقرار می کنند. پیام ها می توانند همان فریم ها یا بسته هایی باشند که در شبکه جابجا می شوند. همچنین می توانند داده های پیچیده ای را با خود حمل کنند. زیر ماژول ها یا ماژولهای ساده این پیام ها را یا به طور مستقیم به مقصد میفرستند یا آنها را در طول یک مسیر مشخص به مقصد می رسانند. هر ماژول هنگام شبیه سازی یک زمان دارد که این زمان هنگامی شروع می شود که ماژول یک پیام دریافت می کند. یک پیام می تواند از یک ماژول دیگر باشد یا از خود آن ماژول باشد. پیامهایی که یک ماژول برای خودش میفرستد حکم یک تایمر را دارد.

۱-۴- گیت ها و کانکشن ها(Gates & Connections)

هر گیت یا دروازه یک مدخل ورودی یا خروجی یک ماژول است که پیام ها از گیت ورودی وارد می شوند و از گیت خروجی خارج می شوند. به این ارتباط ها کانکشن گفته می شود :



شکل ۲

www.NoavaranGermi.ir

همانطور که در شکل می بینید هر زیرماژول یا می تواند از طریق گیت خود با یک زیرماژول دیگر ارتباط برقرار کند یا اینکه از طریق گیت خود با ماژول مرکب یا ماژول پدر خود ارتباط برقرار کند.

۱-۵- پارامترها

هر کانکشن می تواند سه پارامتر بگیرد که مدلسازی شبکه را آسانتر می کنند:

- میزان تاخیر
- نرخ خطای هر بیت
- نرخ داده

ماژول ها نیز می توانند پارامتر بگیرند. پارامترها هم می توانند در فایل .ned و هم در فایل پیکربندی omnetpp.ini تعریف شوند. پارامترها برای تعریف رفتارهای یک زیرماژول ساخته می شوند و می توانند مقدارهای string یا numeric یا Boolean بگیرند.

ماژولهای مرکب هم می توانند پارامترهایی مانند تعداد زیر ماژول ، گیت یا کانکشن بگیرند.

۱-۶- برنامه نویسی الگوریتم ها

ماژولهای ساده شامل توابعی به زبان c++ هستند و از اشیایی مثل ماژول یا پیام استفاده می کنند. همه اینها در ماژول با کلاس های c++ نشان داده می شوند. کلاس های زیر بخشی از کتابخانه کلاسهای OMNeT++ هستند:

- ماژولها،گیتها،کانکشن ها و...
- پارامترها
- پیام ها

۱-۷- سیستم شبیه سازی

یک سیستم شبیه سازی در OMNeT++ شامل کامپوننتهای زیر است:

- هسته شبیه سازی : شامل کدهایی است که شبیه سازی را مدیریت می کنند که به زبان C++ نوشته شده است. (فایلی با پسوند a یاlib)
- رابط کاربر

www.NoavaranGermi.ir

۱-۸- نحوه ساخته شدن فایل اجرایی شبیه سازی

وقتی کاربر فایل های مورد نیاز خود را مثل ned. و msg. ساخت همه این فایل ها توسط OMNeT++ به کلاس های C++ تبدیل می شوند و کدهای C++ ساخته شده کامپایل شده و به هسته شبیه سازی و رابط کاربر پیوند می خورند و فایل اجرایی ساخته می شود.

۱-۹- اجرای فایل اجرایی و تجزیه و تحلیل آن

هنگامی که فایل اجرایی ساخته شده را اجرا می کنیم ابتدا فایل omnetpp.ini را می خواند. این فایل شامل تنظیماتی است که کنترل می کند چطور شبیه سازی اجرا شود، پارامترها چه مقداری بگیرند و...

خروجی شبیه سازی را می توان در فایلهای داده ای vector و scalar ذخیره کرد. OMNeT++ ابزارهای جداگانه برای بررسی هرکدام از این فایلهای خروجی دارد. برنامه plove و Scalars همراه OMNeT++ نصب می شوند که به وسیله آنها به ترتیب می توان فایل های vector و scalar را مشاهده کرده و نتایج آن را بررسی کرد.

۱-۱۰- زبان NED

توپولوژی یک مدل با استفاده از زبان NED تعریف می شود.زبان NED تعریف ماژولار یک شبکه را آسان می کند.این بدان معناست که یک تعریف یک شبکه متشکل است از تعریف تعدادی کامپوننت (کانال ها، انواع ماژول ساده و مرکب).کانال ها، ماژول های ساده و مرکب یک شبکه می توانند در تعریف دیگر شبکه ها نیز استفاده شوند.

فایل هایی که حاوی تعریف شبکه می باشند دارای پسوند ned. هستند.

۱-۱۰-۱- اجزای زبان NED:

- دستورات import
- تعریف های channel
- تعریف های ماژول های ساده و مرکب
- تعریف شبکه

www.NoavaranGermi.ir

۱-۱۰-۲- دستورات import :

این دستورات برای وارد کردن تعریفات از یک فایل تعریف شبکه دیگر استفاده می شوند.بعد از import یک تعریف شبکه می توان از اجزای تعریف شده در آن همانند کانال ها و ماژول های ساده و مرکب استفاده کرد.

برای مثال :

```
import "ethernet";    // imports ethernet.ned
```

۱-۱۰-۳- Identifier:

برای نامگذاری ماژولها، کانالها، گیت ها، پارامترها و ... استفاده می شود. ترکیبی از حروف کوچک و بزرگ و اعداد می باشد که کاراکتر اول آن حتماً باید از حروف باشد. اگر این نام ترکیبی از چند کلمه باشد حرف اول هر کلمه را بهتر است بزرگ در نظر بگیرید.Identifiers به حروف بزرگ و کوچک حساس هستند.

توضیحات(comments) : با علامت // شروع می شود که توسط کامپایلر نادیده گرفته می شود.

در بخش های بعدی در قالب یک مثال به نام TicToc شبیه سازی با OMNeT++ را به شما آموزش دهیم و اگر همه مراحل را عملی انجام دهید درک مفاهیمی که تا بحال توضیح دادیم راحت تر خواهد شد.

۲- مقایسه OMNeT++ و OMNEST(نسخه ی تجاری این نرم افزار)

برنامه OMNeT++ یک برنامه رایگان بوده اما کاربر تنها اجازه دارد از آن برای کارهای آکادمیک و علمی استفاده نماید.نسخه تجاری آن OMNEST نام دارد که دارای برخی امکانات و پشتبانی های بیشتر می باشد.

در زیر مقایسه ای بین این دو نسخه از نرم افزار انجام شده است و در ادامه توضیحی مختصر در مورد هر یک از موارد آورده ایم:

	OMNeT++	OMNEST
License	Academic Public License[1]	Commercial License
Commercial use	not allowed[1]	allowed
Simulation kernel, tools, examples, documentation	yes	yes
Eclipse-based Simulation IDE	yes	yes
Support for all major operating systems[2]	yes	yes
Windows installer	no (distributed as zip)	yes
Pre-compiled (and tested) simulation libraries for Windows	no	yes[3]
Support for Microsoft Visual C++	no	yes
Support for the GCC Compiler[4]	yes	yes
Documentation Generation (example: INET)	yes under Creative Commons[5]	yes
SVG Image Export[6]	no	yes
SystemC Integration[7]	no	yes
HLA Support[8]	no	yes
Support	informal, via the mailing list	guaranteed 48-hour email support available
Service Releases	informal	guaranteed after significant fixes, but at least every 6 months

www.NoavaranGermi.ir

شکل ۳

- در OMNeT++ شما تنها اجازه ی انجام کارهای علمی – پژوهشی را داشته و نمی توانید از آن در کارهای تجاری استفاده نمایید.
- همچنین نسخه رایگان دارای نصاب تحت ویندوز نمی باشد و شما باید خودتان به صورت دستی آنرا تنظیم و نصب نمایید.
- تمام کتابخانه های شبیه سازی تنها در نسخه ی تجاری قبلا کامپایل و تست شده اند.
- نسخه ی رایگان نرم افزار از Microsoft Visual C++ پشتیبانی نمی کند.
- در نسخه رایگان میبایست ایجاد مستندات تحت لیسانس Creative Commons انجام گیرد.این لیسانس برای انتشار راحت تر و امن تر مطالب بین کاربران می باشد و جلوی استفاده ی تجاری از مطالب توسط شرکت ها و کمپانی های تجاری را می گیرد.در واقع با این لیسانس شما به جای آنکه بگویید تمام حقوق رزرو شده است[۱] می گویید بعضی از حقوق رزرو شده است[۲].
- توسط نسخه تجاری نرم افزار می توان از مدل ها و نمودارهای توالی (نموداری که نشان دهنده ی پیام های تبادل شده در مدل شبیه سازی شده می باشد) خروجی فایل SVG گرفت.

SVG[۳] زبانی‌ست از نوع اکس‌ام‌ال که به منظور ایجاد، انتشار، و کار با گرافیک دوبعدی و نیز کاربردهای گرافیکی بر روی اینترنت ایجاد گردیده است. به عنوان استانداردی جدید از سوی کنسرسیوم وب جهانی[۴]، اس‌وی‌جی باعث کوچک‌تر گردیدن، سریع‌تر بودن، و تعاملی‌تر[۵] شدن اسناد دربردارنده گرافیک و انیمیشن می‌گردد.در واقع omnest می تواند خروجی تصویر یا انیمیشن از مدل یا نمودار برنامه تولید کرده تا در جایی دیگر بدون نیاز به نرم افزار omnest مدل خود را بر روی یک مرورگر مانند opera مشاهده کنیم.

این فایلها بسیار کم حجم بوده و با بزرگ نمایی کیفیت خود را از دست نمی دهند.

- در صورتی که ماژولی از یک سیستم که به کمک SystemC ایجاد شده است، دراختیار دارید با omnest می توانید آنرا با ماژول های طراحی شده در omnest مجتمع نمایید

---

[۱] all rights reserved

[۲] some rights reserved

[۳] Scalable Vector Graphics

[۴] W۳C

[۵] interactive

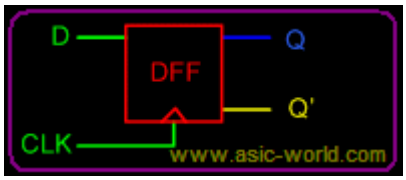
امـــا متاســـفانه ایـــن امکـــان در نســـخه رایگـــان OMNeT++ موجـــود نمـــی باشـــد. SystemC یـک کتابخانـه کـلاس C++ مـی باشـد کـه بـرای پشـتیبانی طراحـی سـطح سیسـتمی، توسـعه یافـت و دانلـود آن رایگـان میباشـد.SystemC زبـانی بـرای تعریـف اجـزای سـخت افـزاری و نـرم افـزار، زبـانی بـرای آسـان سـازی شـبیه سـازی یکپارچـه نـرم افـزار- سـخت افـزار فراهم کرده است.

برای مثال قطعه کد زیر

```
1  // D-FF Code
2  #include "systemc.h"
3
4  SC_MODULE(d_ff) {
5    sc_in<bool> din;
6    sc_in<bool> clock;
7    sc_out<bool> dout;
8
9    void doit() {
10     dout = din;
11   };
12
13   SC_CTOR(d_ff) {
14     SC_METHOD(doit);
15     sensitive_pos << clock;
16   }
17 };
```

شکل ٤

تعریف یک فلیپ فلاپ می باشد:



شکل ٥

- HLA مخفف high-level architecture می باشد که یک معماری همه منظوره برای سیستم های شبیه سازی کامپیوترهای توزیع شده می باشد.با استفاده از HLA، شبیه سازی های کامپیوتری می توانند با دیگر شبیه سازی ها صرفه نظر از پلتفرم محاسباتی شان، باهم تعامل(تبادل داده و همزمان سازی عملیات) داشته باشند.در omnest از HLA پشتیبانی می شود تا بتوان براساس HLA / IEEE ١٥١٦ به دیگر شبیه سازها متصل شد.
- در omnest تضمین شده است که پاسخ گویی از طریق ایمیل ظرف مدت ٤٨ ساعت انجام گیرد.

www.NoavaranGermi.ir

۳- نصب نرم افزار OMNeT++

OMNeT++ در سیستم عامل های زیر قابل نصب می باشد:

- Windows ۷, Vista, XP
- Mac OS X ۱۰,٦ and ۱۰,۷
- Linux(Ubuntu، ۱٦ and ۱٥ Fedora، Red Hat، OpenSUSE، Unix)

۳-۱- نحوه نصب در سیستم عامل های ویندوز:

ابتدا باید از سایت omnetpp.org فایل آرشیو شده ی omnetpp-٤,۲,۲-src-windows.zip که مخصوص ویندوز می باشد را دریافت کنید.این پکیج با داشتن فایل های OMNeT++، کامپایلر C++،یک command-line و تمام کتابخانه ها و برنامه های مورد نیاز برای OMNeT++ تقریبا کامل است.

باید این پکیج را در پوشه ای قرار دهید که دارای هیچ فاصله ای در نام آن کامل آن نباشد.مثلا نمی توان آن را در پوشه Program Files قرار داد.سپس آنرا از حالت آرشیو دربیاورید.

۳-۲- تنظیم و ساخت OMNeT++

بر روی mingwenv.cmd موجود در پوشه omnetpp-٤,۲,۲ دابل کلیک کرده تا کنسول باز شود.حال دستورات زیر را وارد کرده:

```
configure
make
```

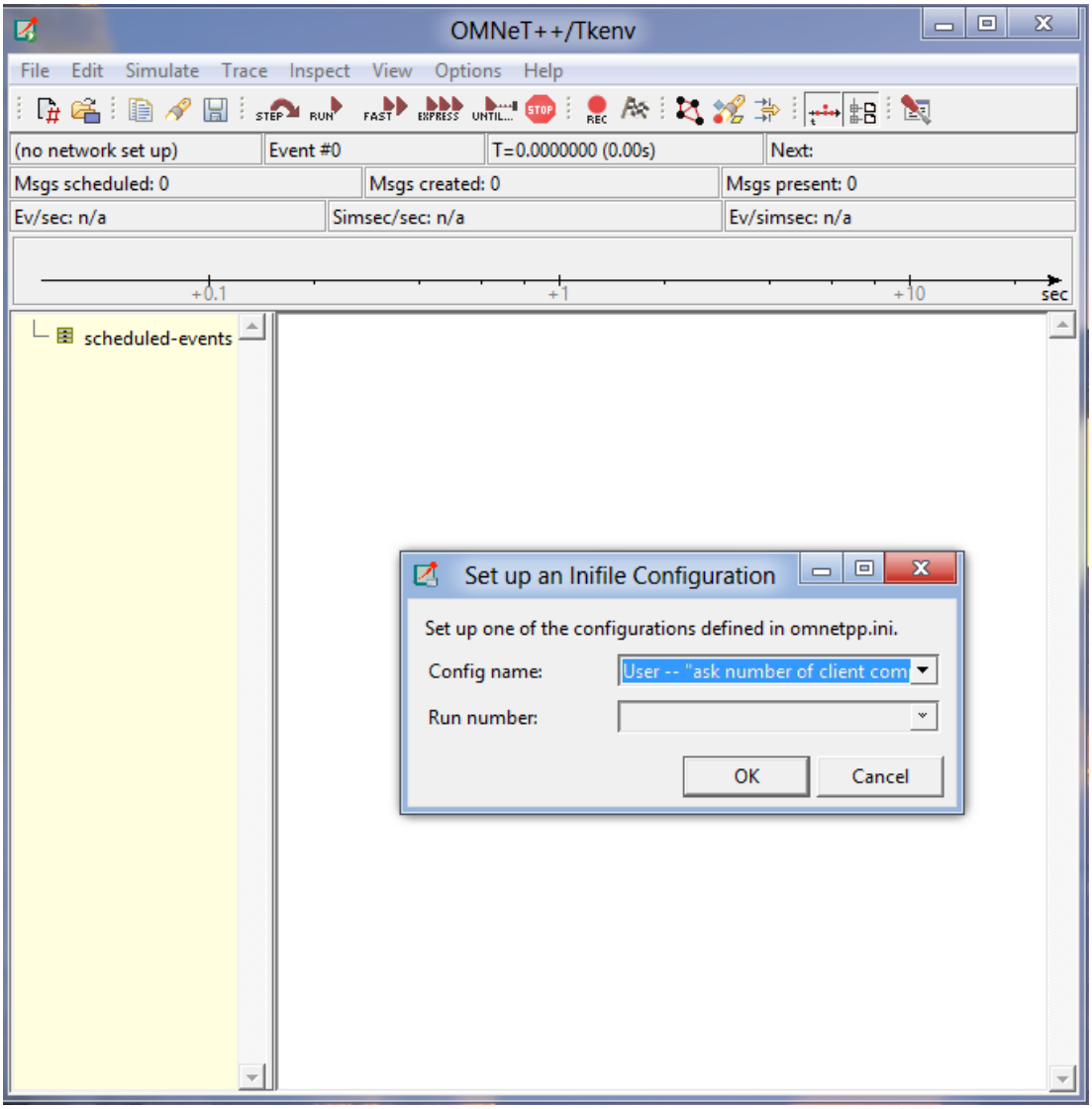
این دستورات باینری های debug و release را ایجاد می کنند.ایجاد این فایل ها نسبتا زمان بر است و چند دقیقه ای به طول می انجامد.

۳-۳- بررسی درستی نصب

برای بررسی اینکه آیا نصب شما درست انجام شده است یا خیر می توانید یکی از برنامه های نمونه موجود در پوشه samples را اجرا نمایید.مثلا با دستورات زیر به پوشه dyna رفته و آنرا اجرا کنید:

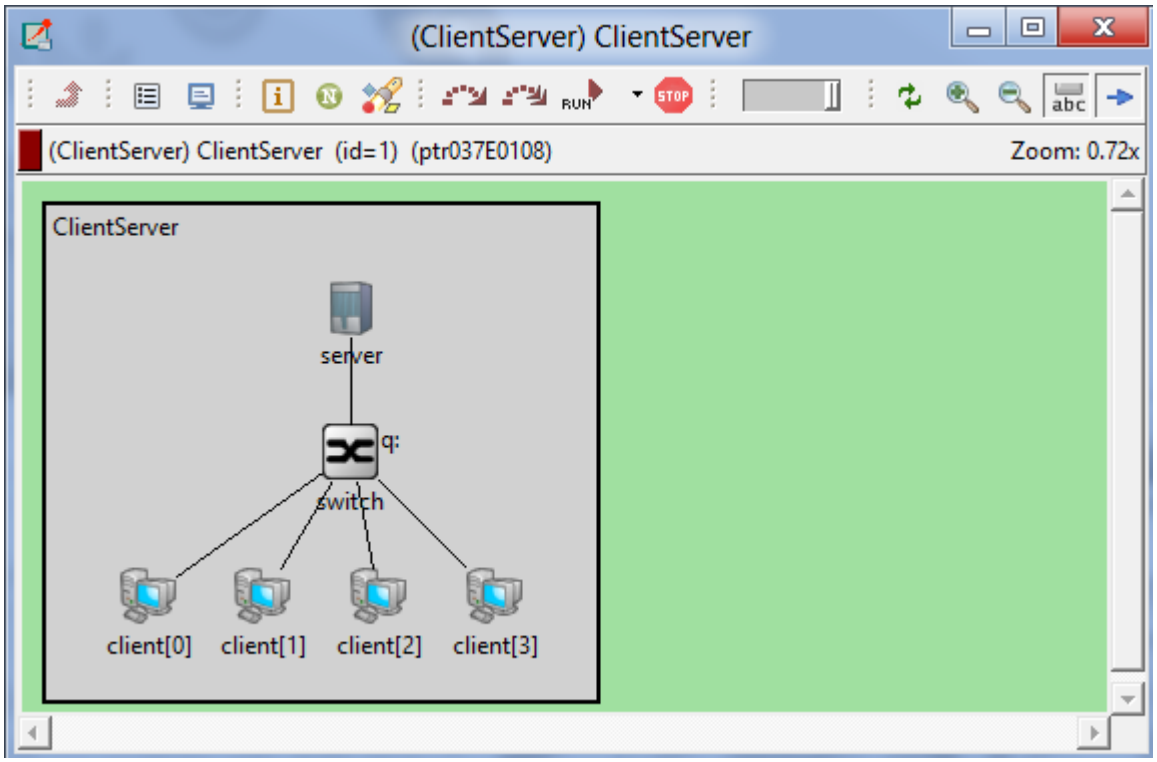
```
cd samples/dyna
dyna
```

با اجرای این دستور باید ابتدا پنجره زیر را از محیط گرافیکی Tkenv مشاهده نمایید:



شکل ٦

سپس با فشردن دکمه OK ، پنجره زیر که نشان دهنده ی خروجی گرافیکی برنامه dyna می باشد را خواهید دید:



شکل ۷

۳-۴- اجرای IDE:

OMNeT++ به همراه IDE شبیه ساز برمبنای اکلیپس عرضه شده است.با دستور زیر می توانید این IDE را اجرا کنید:

Omnetpp

٤- بررسی مثال عملی شبیه سازی TicToc

از آنجاییکه بخش بسیار مهمی از نرم افزار OMNeT++ شبیه سازی ارتباطات در شبکه است،در این جا به نحوه ی شبیه سازی یک شبکه می پردازیم.برای شروع از شبکه ای شروع می کنیم که دارای دو گره است.کاری که این دو گره انجام می دهند بسیار ساده است. یک گره packet ساخته و آن را به گره دیگر پاس می دهد و گره دیگر نیز آن را دریافت کرده و دوباره به گره اول بازپس می فرستد و همین طور این کار را تا بی نهایت انجام می دهند.نام این دو گره را "tic" و "toc" می گذاریم.

۴-۱- شروع به کار

٤-۱-۱- قدم ۱: پیاده سازی اولیه

در اینجا مراحل پیاده سازی این شبیه سازی را از ابتدا توضیح می دهم:

۱. یک پوشه به نام tictoc بسازید و با وارد کردن دستور cd در mingwenv.cmd وارد این پوشه شوید.

۲. با ساختن یک فایل NED توپولوژی شبکه ی خود را تعریف کنید.فایل توپولوژی یک فایل متنی ساده است که گره های شبکه و ارتباط میان آنها را مشخص می کند.این فایل را می توانید با هر ویرایشگر متنی مانند notepad بسازید.ما در اینجا اسم این فایل را tictoc۱.ned میگذاریم:

```
simple Txc1
{
    gates:
        input in;
        output out;
}

//
// Two instances (tic and toc) of Txc1 connected both ways.
// Tic and toc will pass messages to one another.
//
network Tictoc1
{
    submodules:
        tic: Txc1;
        toc: Txc1;
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

شکل ۸

بهتر است این فایل را از پایین به بالا بخوانید:

- Tictoc۱ شبکه ای است که از دو زیر ماژول به نام tic و toc ساخته شده است.tic و toc هر دو نمونه هایی از یک ماژول یکسان به نام Txc۱ هستند.ما گیت خروجی tic را که out نام دارد به گیت ورودی toc به نام in متصل نموده ایم و به صورت برعکس نیز همین کار را انجام دادیم.تاخیر انتشار در هر میسر را نیز ۱۰۰ میلی ثانیه در نظر گرفتیم.
- Txc۱ یک ماژول ساده است و یک گیت خروجی به نام out و یک گیت ورودی به نام in دارد.

۳. حال باید نحوه ی عملکرد (functionality) ماژول ساده Txc۱ را پیاده سازی کنیم.این کار را با نوشتن فایل txc۱.cc با زبان c++ انجام می دهیم:

```
#include <string.h>
#include <omnetpp.h>


class Txc1 : public cSimpleModule
{
  protected:
    // The following redefined virtual function holds the algorithm.
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

// The module class needs to be registered with OMNeT++
Define_Module(Txc1);

void Txc1::initialize()
{
    // Initialize is called at the beginning of the simulation.
    // To bootstrap the tic-toc-tic-toc process, one of the modules needs
    // to send the first message. Let this be `tic'.

    // Am I Tic or Toc?
    if (strcmp("tic", getName()) == 0)
    {
        // create and send first message on gate "out". "tictocMsg" is an
        // arbitrary string which will be the name of the message object.
        cMessage *msg = new cMessage("tictocMsg");
        send(msg, "out");
    }
}

void Txc1::handleMessage(cMessage *msg)
{
    // The handleMessage() method is called whenever a message arrives
    // at the module. Here, we just send it to the other module, through
    // gate `out'. Because both `tic' and `toc' does the same, the message
    // will bounce between the two.
    send(msg, "out");
}
```

شکل ۹

ماژول ساده Txc۱ با کلاس Txc۱ به زبان C++ ارائه شده است که زیر کلاسی از cSimpleModule است و از آن ارث بری می کند و با ماکرو Define_Module() در OMNeT++ ثبت شده است.ما دو متد از cSimpleModule به نام های initialize() و handleMessage() را دوباره تعریف کرده ایم.آنها توسط هسته شبیه سازی فراخوانی می شوند: اولی تنها یک مرتبه و دومی هر زمان که پیامی به ماژول برسد.

در initialize() شئ از نوع cMessage ساختیم و آنرا از طریق گیت out ارسال کردیم.از آنجاییکه این گیت به گیت ورودی ماژول دیگری متصل است، هسته شبیه ساز پیام را به ماژول دیگر متصل به سمت دیگر گیت از طریق آرگمانی در handleMessage() بعد از گذشت ۱۰۰ میلی ثانیه تاخیر که در فایل NED تعیین شده بود، می رساند.و ماژول دیگر تنها آن را دوباره با ۱۰۰ ثانیه تاخیر باز پس می فرستد.

پیام ها(پکت ها، فریم ها، job و ...) و رویدادها(timers, timeout) به وسیله ی شئ ای از cMessage و یا یکی از زیرکلاس های آن در OMNeT++ نمایش داده می شوند.بعد از ارسال و یا زمانبندی آنها،توسط هسته شبیه سازی در لیست "scheduled events" و یا "future events" قرار می گیرند تا زمانی که زمان شان به پایان برسد و سپس از طریق handleMessage() به ماژول مقصد تحویل داده می شوند.

٤. حال Makefile را با کمک دستور زیر می سازیم که به ما کمک می کند تا برنامه خود را compile و link کرده و فایل اجرایی tictoc را بسازیم:

```
$ opp_makemake
```

٥. با دستور زیر اولین شبیه سازی خود را کامپایل و لینک می کنیم و فایل اجرایی را می سازیم:

```
$ make
```

٦. اگر همین الان فایل اجرایی را اجرا کنید،پیغام می دهد که نمی تواند فایل omnetpp.ini را پیدا کند.در شبیه سازی همزمان می توان چندین شبکه تعریف کرد و این فایل به برنامه شبیه ساز می گوید که شما کدام شبکه را می خواهید شبیه سازی کنید.پس باید فایل به نام omnetpp.ini بسازید و در آن دستور زیر را بنویسید:

```
[General]
network = Tictoc1
```

اما در مثال های بعدی برای راحتی کار،ما از فایل omnetpp.ini زیر استفاده می کنیم:

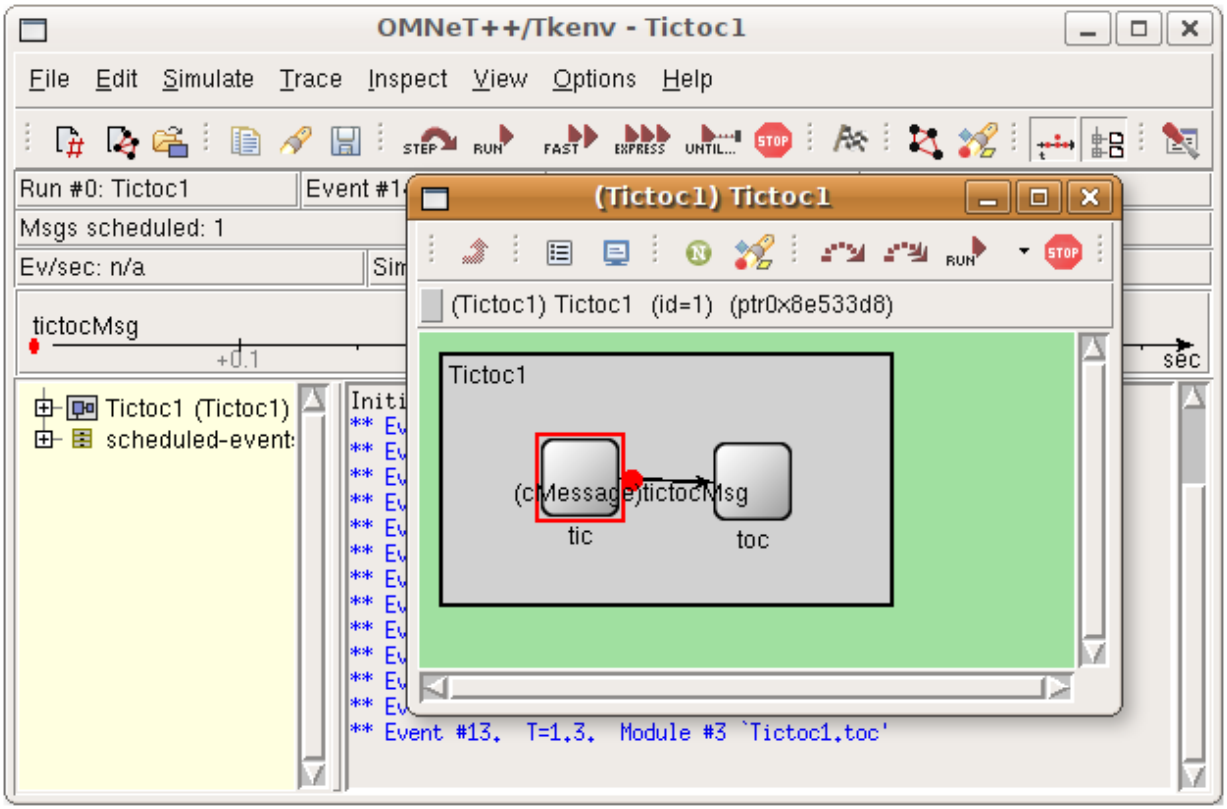
```
[General]
# nothing here
[Config Tictoc1]
network = Tictoc1

[Config Tictoc2]
network = Tictoc2

[Config Tictoc3]
network = Tictoc3

[Config Tictoc4]
network = Tictoc4
Tictoc4.toc.limit = 5

[Config Tictoc5]
network = Tictoc5
**.limit = 5

[Config Tictoc6]
network = Tictoc6

[Config Tictoc7]
network = Tictoc7
# argument to exponential() is the mean; truncnormal() returns values from
# the normal distribution truncated to nonnegative values
Tictoc7.tic.delayTime = exponential(3s)
Tictoc7.toc.delayTime = truncnormal(3s,1s)

[Config Tictoc8]
network = Tictoc8

[Config Tictoc9]
network = Tictoc9

[Config Tictoc10]
network = Tictoc10

[Config Tictoc11]
network = Tictoc11

[Config Tictoc12]
network = Tictoc12
```

```
[Config Tictoc13]
network = Tictoc13

[Config Tictoc14]
network = Tictoc14

[Config Tictoc15]
network = Tictoc15
record-eventlog = true
```

شکل ۱۰

www.NoavaranGermi.ir

۷. بعد از انجام تمام مراحل گفته شده حال می توانید با نوشتن دستور tictoc برنامه را اجرا کنید و پنجره شبیه ساز OMNeT++ را ببینید:



شکل ۱۱

۸. دکمه Run در toolbar شبیه ساز را بزنید تا تبادل پیام شروع شود.

Toolbar پنجره ی اصلی برنامه زمان شبیه سازی را نمایش می دهد. این زمان مجازی است و هیچ ربطی به زمان واقعی یا زمان ساعت دیواری (wall-clock) ندارد.

توجه کنید که در اینجا برای پردازش پیام در نود هیچ زمان شبیه سازی ای را اختصاص نداده ایم.تنها زمان شبیه سازی سپری شده در اینجا مربوط به تاخیر انتقال پیام در اتصالات نودها بود.

۹. می توانید سرعت انیمیشن را از طریق دکمه slider موجود در بالای پنجره کم یا زیاد کنید.

۴-۲- ارتقا و بهبود مدل دو گره ای TicToc

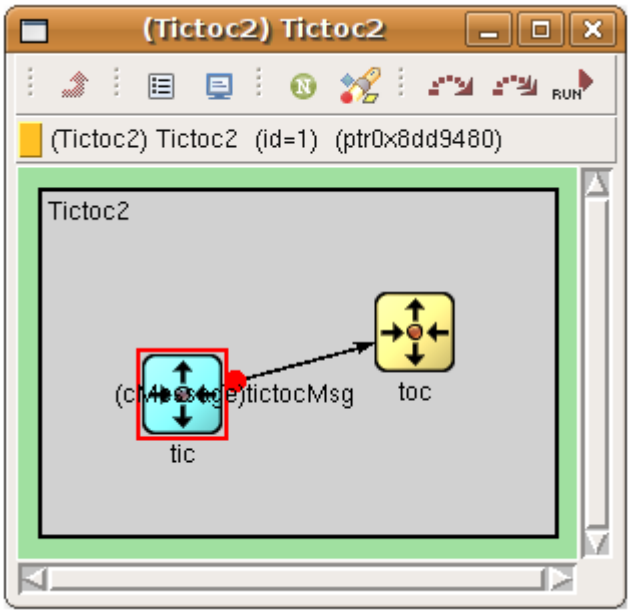
٤-۲-۱- قدم ۲: بهبود گرافیک و افزودن خروجی debug

در این مرحله ظاهر مدل در GUI را کمی بهتر می کنیم.آیکن routing در مسیر images/block/routing.png را به گره ها اختصاص داده و tic را فیروزه ای و toc را زرد می کنیم. این کار از طریق رشته display در فایل NED انجام می شود که تگ i در آن مشخص کننده ی آیکن است.

```
simple Txc2
{
    parameters:
        @display("i=block/routing"); // add a default icon
    gates:
        input in;
        output out;
}

//
// Make the two module look a bit different with colorization effect.
// Use cyan for `tic', and yellow for `toc'.
//
network Tictoc2
{
    submodules:
        tic: Txc2 {
            parameters:
                @display("i=,cyan"); // do not change the icon (first arg of i=) just colorize it
        }
        toc: Txc2 {
            parameters:
                @display("i=,gold"); // here too
        }
    connections:
```

شکل ۱۲

نتیجه ی این تغییرات به شکل زیر خواهد بود:



شکل ۱۳

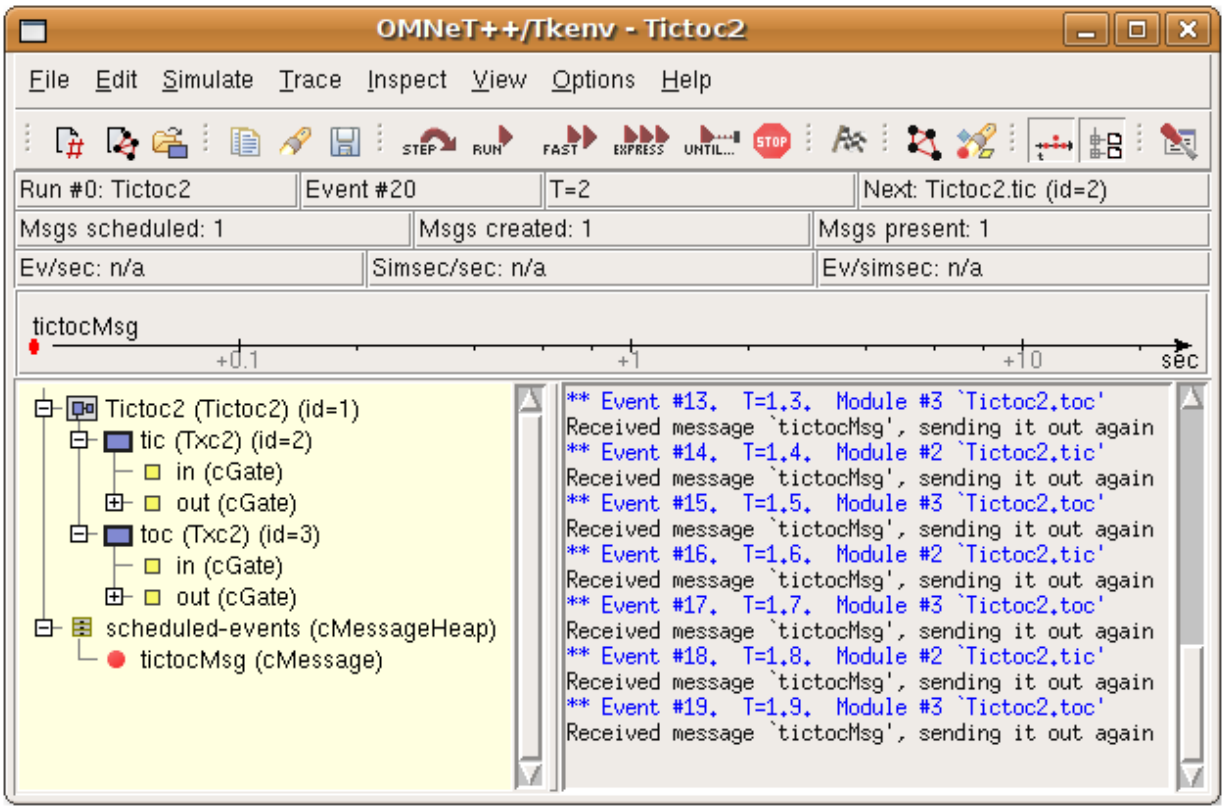
همچنین ما فایل C++ را برای افزودن پیامهای دیباگ به Txc۱، به وسیله ی شئ EV تغییر دادیم:

```
EV << "Sending initial message\n";
```

و

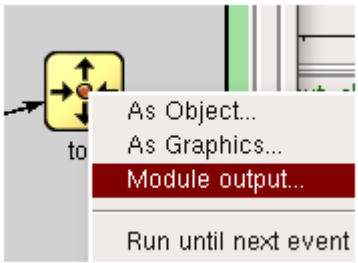
```
EV << "Received message `" << msg->getName() << "', sending it out again\n";
```

اگر شبیه سازی را در OMNeT++ GUI Tkenv اجرا کنید، خروجی زیر را در پنجره متنی خواهید دید:



شکل ١٤

همچنین می توانید پنجره های جداگانه ای برای هر یک از tic یا toc باز کنید تا خروجی مربوط به خودشان را در آن مشاهده نمایید.این کار مناسب زمانی است که مدل بزرگی دارید و تنها علاقمندید که خروجی مربوط به یکی از ماژول های خاص را ببینید.برای این کار بر روی آیکن یکی از گره ها راست کلیک کنید و module output را از منوی باز شده انتخاب نمایید.



شکل ١٥

٤-٢-٢- قدم ٣: افزودن متغییرهای وضعیت

در این مرحله شمارنده ای به ماژول اضافه کرده و پیام را بعد از ده مرتبه رد وبدل شدن بین گره ها حذف می کنیم.

شمارنده را به شکل عضو کلاس اضافه می کنیم:

```
class Txc3 : public cSimpleModule
{
  private:
    int counter;  // Note the counter here

  protected:
```

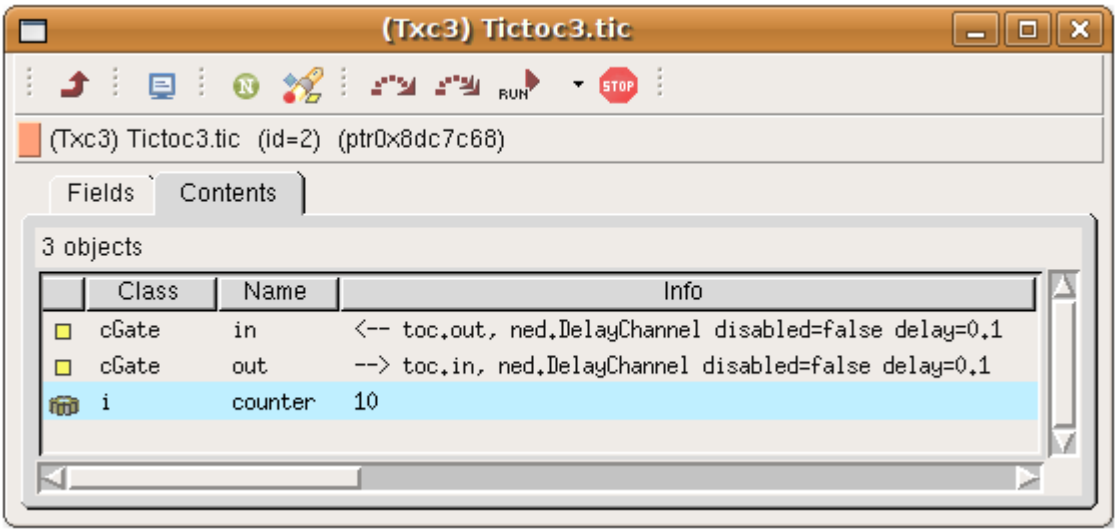
شکل ۱۶

در تابع ()initialize متغیر را برابر ۱۰ قرار داده و در تابع ()handleMessage یعنی در هر بار دریافت پیام آن را کم می کنیم.پس از آنکه به مقدار صفر رسید، شبیه سازی متوقف می شود.

به دستور زیر در کدها دقت کنید:

```
WATCH(counter);
```

این دستور، دیدن مقدار counter در Tkenv را ممکن می سازد.بر روی آیکن tic کلیک کنید، سپس صفحه Contens را از پنجره ی باز شده انتخاب کنید.



شکل ۱۷

با ادامه اجرای شبیه سازی، می توانید کم شدن مقدار شمارنده تا رسیدن آن به مقدار صفر را دنبال کنید.

٤-٢-٣- قدم ٤: افزودن پارامترها

در این مرحله با نحوه افزودن پارامترهای ورودی به شبیه سازی آشنا می شوید:ما عدد ۱۰ را با پارامتری عوض کرده و نیز یک پارامتر بولی اضافه می کنیم تا ماژول تصمیم بگیرد که آیا باید پیغامی را در قسمت کد initialization خود ارسال کند یا خیر!

پارامترهای ماژول باید در فایل NED تعریف شوند.نوع داده می تواند عددی،رشته،بولی ویا حتی xml باشد.

```
simple Txc4
{
    parameters:
        bool sendMsgOnInit = default(false); // whether the module should send out a message on initialization
        int limit = default(2);   // another parameter with a default value
        @display("i=block/routing");
    gates:
```

شکل ۱۸

باید برای خواندن پارامتر و اختصاص آن به شمارنده در initialize() کد C++ را نیز کمی تغییر دهیم:

```
counter = par("limit");
```

از دومین پارامترنیز برای تصمیم اینکه آیا پیام اولیه را بفرستیم یا خیر استفاده می کنیم:

```
if (par("sendMsgOnInit").boolValue() == true)
```

حال میتوانیم پارامترها را در فایل NED یا omnetpp.ini مقداردهی کنیم.که مقداردهی در فایل NED اولویت دارد.میتوانید برای پارامترها با استفاده از دستور default در فایل NED مقادیر پیشفرض تعریف کنید که در این صورت می توانید پارامترها را از طریق omnetpp.ini مقداردهی کنید و یا از همان مقادیر مشخص شده در default موجود در فایل NED استفاده کنید.

در اینجا من یک پارامتر را مقداردهی کردم:

```
network Tictoc4
{
    submodules:
        tic: Txc4 {
            parameters:
                sendMsgOnInit = true;
                @display("i=,cyan");
        }
        toc: Txc4 {
            parameters:
                sendMsgOnInit = false;
                @display("i=,gold");
        }
    connections:
```

شکل ۱۹

و دیگری را در فایل omnetpp.ini مقدار دادم:

```
Tictoc4.toc.limit = 5
```

از آنجایکه omnetpp.ini از wildcard پشتیبانی می کند و مقداردهی پارامترها از طریق فایل های NED بر omnetpp.ini اولویت دارند،می توانیم از این روش استفاده کنیم:

```
Tictoc4.t*c.limit=5
```

یا

```
Tictoc4.*.limit=5
```

یا حتی

```
**.limit=5
```

که همه در اینجا نتیجه ی یکسانی دارند.ماژولی که limit کوچکتری دارد پیام را حذف کرده و در نتیجه به شبیه سازی خاتمه می دهد.

در Tkenv می توانید پارامترهای ماژول را از طریق درخت شی موجود در سمت چپ پنجره اصلی یا در صفحه پارمترهای module inspector که با دابل کلیک کردن بر روی آیکن ماژول باز می شود،مشاهده و بررسی کنید.

٤-٢-٤- قدم ٥: استفاده از ارث بری

اگر با دقت به فایل NED نگاه کنیم خواهیم فهمید که tic و toc تنها در مقدار پارمترها و رشته display خود با یکدیگر متفاوت اند.می توانیم نوع ماژول ساده ی جدیدی از طریق ارث بری بسازیم و بعضی از پارمترها را override کنیم.در اینجا ما دو ماژول ساده Tic و Toc داریم که مشتق شده اند.

ارث بری از یک ماژول ساده به آسانی انجام می شود:

```
simple Txc5
{
    parameters:
        bool sendMsgOnInit = default(false);
        int limit = default(2);
        @display("i=block/routing");
    gates:
        input in;
        output out;
}
```

شکل ۲۰

در ماژول مشتق شده تنها مقدار پارامترها را مشخص می کنیم.

```
simple Tic5 extends Txc5
{
    parameters:
        @display("i=,cyan");
        sendMsgOnInit = true;   // Tic modules should send a message on init
}
```

۲۱ شکل

در ماژول Toc نیز به همین صورت است، تنها مقادیر پارامترها متفاوت اند.

```
simple Toc5 extends Txc5
{
    parameters:
        @display("i=,gold");
        sendMsgOnInit = false;  // Toc modules should NOT send a message on init
}
```

شکل ۲۲

هنگامی که ماژولهای ساده جدید می سازیم، می توانیم از آنها به عنوان submodule در شبکه ی خود استفاده کنیم:

```
network Tictoc5
{
    submodules:
        tic: Tic5;  // the limit parameter is still unbound here. We will get it from the ini file
        toc: Toc5;
    connections:
```

شکل ۲۳

همانطور که می بینید تعریف شبکه بسیار کوتاه تر و ساده تر شد.

٤-۲-٥- قدم ٦: مدلسازی تاخیر پردازش

در مدلهای قبلی، tic و toc بلافاصله بعد از دریافت پیام آنرا باز پس می فرستادند.در اینجا ما tic و toc را به گونه ای تغییر می دهیم که پیام را قبل از ارسال مجدد به اندازه یک ثانیه ی شبیه سازی شده نگاه دارند.در OMNeT++ یک چنین شبیه سازی از طریق ارسال پیام به خود بدست می آید.این پیامها self-messages نامیده می شوند(تنها به دلیل نحوه ی استفاده از آنها وگرنه آنها نیز همانند دیگر پیام ها هستند).

دو متغیر از نوع اشارگر به cMessage به نامهای event و tictocMsg به کلاس می افزاییم.یکی برای ارسال پیام به خود و دیگری برای ارسال پیام به ماژول دیگر.

```
class Txc6 : public cSimpleModule
{
  private:
    cMessage *event; // pointer to the event object which we'll use for timing
    cMessage *tictocMsg; // variable to remember the message until we send it back

  public:
```

شکل ۲٤

Self-messages را به وسیله ی تابع scheduleAt () به خود می فرستیم و از طریق آرگمان های آن می توانیم زمانی که باید پیام تحویل داده شود را تعیین کنیم.

```
scheduleAt(simTime()+1.0, event);
```

حال در handleMessage() باید نوع پیام دریافتی را تشخیص دهیم.

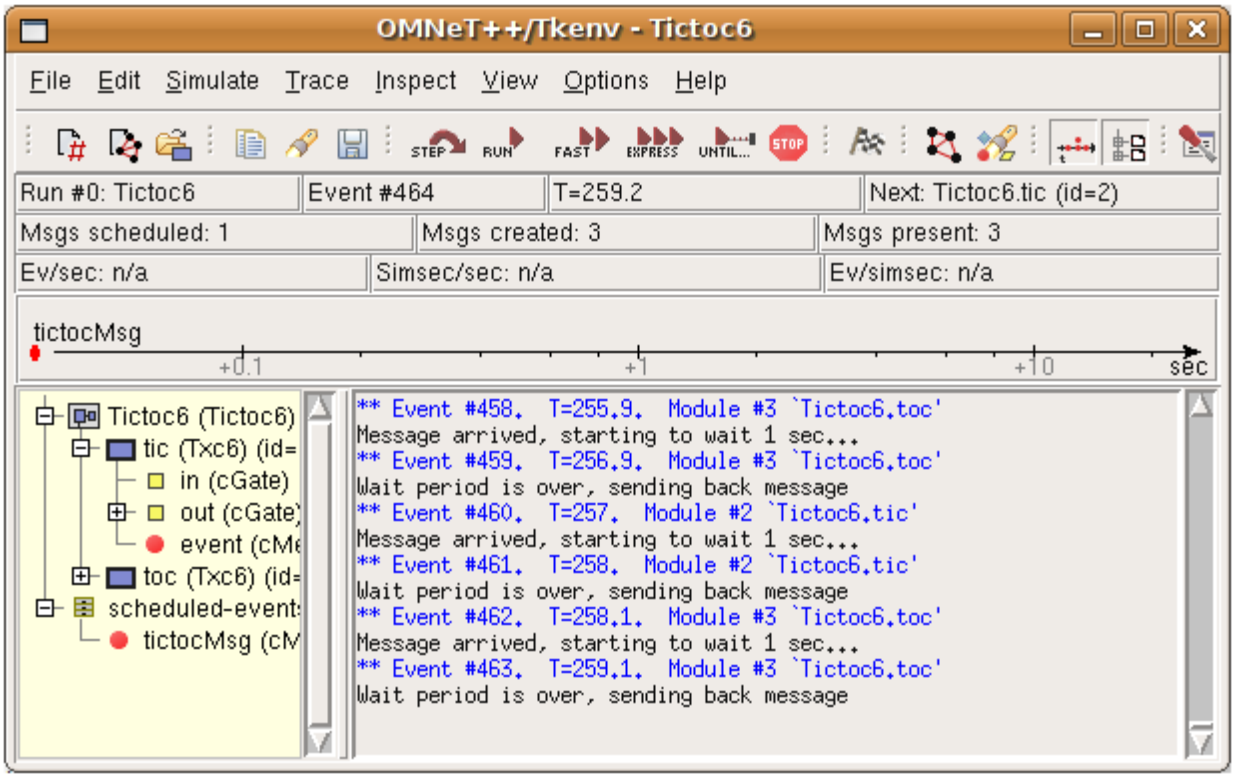
```
if (msg==event)
```

همچنین این کار را می توانیم به این روش انجام دهیم:

```
if (msg->isSelfMessage())
```

در اینجا برای سادگی کار شمارنده را حذف کردیم.

نتیجه اجرای شبیه سازی در زیر نشان داده شده است:



شکل ۲۵

٤-۲-٦- قدم ۷ : شماره های تصادفی و پارامترها

در این مرحله شماره های تصادفی را معرفی می کنیم.زمان تاخیر را از ۱ ثانیه به یک مقدار تصادفی که می تواند از طریق فایل NED و یا omnetpp.ini تنظیم شود تغییر می دهیم. پارامترهای ماژول قادر هستند تا متغییرهای تصادفی بازگردانند؛به هر حال برای استفاده از این ویژگی باید پارامتر را هر بار که از آن استفاده می کنیم در handleMessage() بخوانیم.

```
// The "delayTime" module parameter can be set to values like
// "exponential(5)" (tictoc7.ned, omnetpp.ini), and then here
// we'll get a different delay every time.
simtime_t delay = par("delayTime");

EV << "Message arrived, starting to wait " << delay << " secs...\n";
tictocMsg = msg;
scheduleAt(simTime()+delay, event);
```

شکل ۲٦

همچنین با یک احتمال بسیار کم پکت را از دست می دهیم(آن را به دست خودمان پاک می کنیم).

```
if (uniform(0,1) < 0.1)
{
    EV << "\"Losing\" message\n";
    delete msg;
}
```

شکل ۲۷

پارامترها را در omnetpp.ini مقداردهی می کنیم:

```
Tictoc7.tic.delayTime = exponential(3s)
Tictoc7.toc.delayTime = truncnormal(3s,1s)
```

شکل ۲۸

مهم نیست چند بار شبیه سازی را اجرا می کنید(یا آن را restart می کنید)، می توانید امتحان کنید، در هر بار نتایج دقیقا یکسانی را دریافت خواهید کرد.این بدان خاطر است که OMNeT++ برای تولید اعداد تصادفی از یک الگوریتم قطعی استفاده می کند و آنرا در هر بار با یک seed یکسان مقداردهی می کند.این امر برای شبیه سازی های قابل تکرار بسیار مهم است.در صورت تمایل می توانید شبیه سازی را با seed متفاوت امتحان کنید.برای اینکار تنها کافی است خط زیر را به omnetpp.ini بیافزایید:

```
[General]
seed-0-mt=532569  # or any other 32-bit value
```

شکل ۲۹

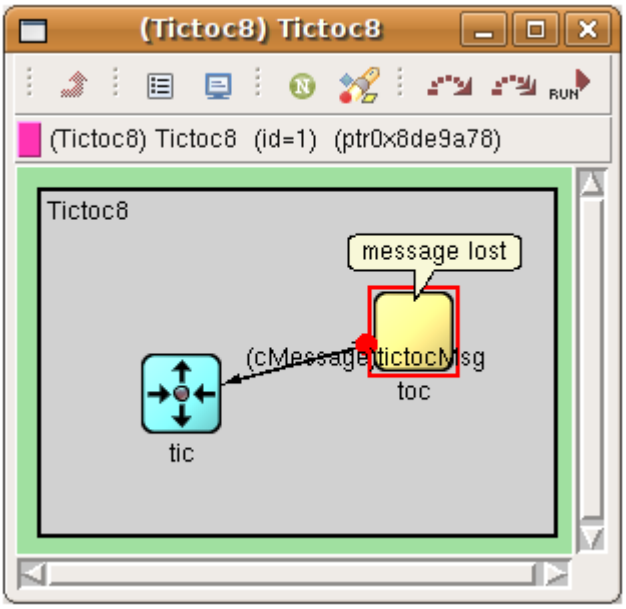
٤-۲-۷- قدم ۸: لغو تایمرها و اتمام زمان

برای اینکه یک گام به مدلسازی پروتکلهای شبکه نزدیکتر شویم، اجازه دهید تا مدل خود را تبدیل به یک مدل از نوع شبیه سازی stop-and-wait کنیم.این دفعه کلاسهای tic و toc را از هم جدا می کنیم.همانند دفعه قبل tic و toc یک پیام را به هم پاس می دهند و toc با یک احتمال کم پیام را گم می کند اما این بار در صورت رخ دادن یک چنین اتفاقی tic شروع به ارسال مجدد پیام می کند.

```
void Toc8::handleMessage(cMessage *msg)
{
    if (uniform(0,1) < 0.1)
    {
        EV << "\"Losing\" message.\n";
        bubble("message lost");  // making animation more informative...
        delete msg;
    }
    else
```

شکل ۳۰

با استفاده از فراخوانی toc،bubble هر زمان که پیام را از دست بدهد آن را اعلام می کند:



شکل ۳۱

خوب، tic هر زمان که پیغامی را ارسال کند، تایمری را تنظیم می کند و زمانی که زمان تایمر به اتمام رسید، فرض می کند که پیام از دست رفته است و پیام دیگری را ارسال می کند.در صورتی که جواب toc رسید تایمر باید لغو شود.همان طور که حدس زده اید این بار نیز تایمر یک self-message است:

```
scheduleAt(simTime()+timeout, timeoutEvent);
```

لغو تایمر:

```
cancelEvent(timeoutEvent);
```

٤-٢-٨- قدم ٩: ارسال دوباره ی یک پیام

در مرحله قبل تنها در صورتی که نیاز به ارسال مجدد یک پکت بود آنرا می ساختیم اما کاری که ما در اینجا انجام می دهیم نگهداری پیام اصلی و تنها ارسال کپی هایی از آن است که دیگر نیاز به ساخت مجدد آن نباشد و زمانی که پیام تایید toc رسید پیام اصلی را پاک می کنیم.برای آسانتر شدن درک مدل در محیط visual، به نام پیام ها شماره پیام را نیز اضافه می کنیم.

برای اجتناب از بزرگ شدن بیش از حد handleMessage() دو تابع جدید به نام های generateNewMessage() و sendCopyOf() می سازیم و آنها را در handleMessage() فراخوانی می کنیم:

```
cMessage *Tic9::generateNewMessage()
{
    // Generate a message with a different name every time.
    char msgname[20];
    sprintf(msgname, "tic-%d", ++seq);
    cMessage *msg = new cMessage(msgname);
    return msg;
}
```

شکل ۳۲

```
void Tic9::sendCopyOf(cMessage *msg)
{
    // Duplicate message and send the copy.
    cMessage *copy = (cMessage *) msg->dup();
    send(copy, "out");
}
```

شکل ۳۳

۴-۳- تبدیل به یک شبکه ی واقعی

٤-٣-١- قدم ١٠ : بیش از دو گره

حالا می خواهیم قدمی بزرگ برداریم: چندین ماژول tic بسازیم و آنها را به یک شبکه متصل کنیم.فعلا کاری که انجام می دهند را ساده در نظر می گیریم: یکی از گره ها پیامی ساخته و ارسال می کند، این پیام بین گره ها به صورت تصادفی پاس داده می شود تا زمانی که پیام به مقصد از قبل تعیین شده برسد.

فایل NED نیاز به کمی تغییر دارد.اول از همه ماژول باید دارای چندین گیت ورودی و خروجی باشد:

```
simple Txc10
{
    parameters:
        @display("i=block/routing");
    gates:
        input in[];  // declare in[] and out[] to be vector gates
        output out[];
}
```

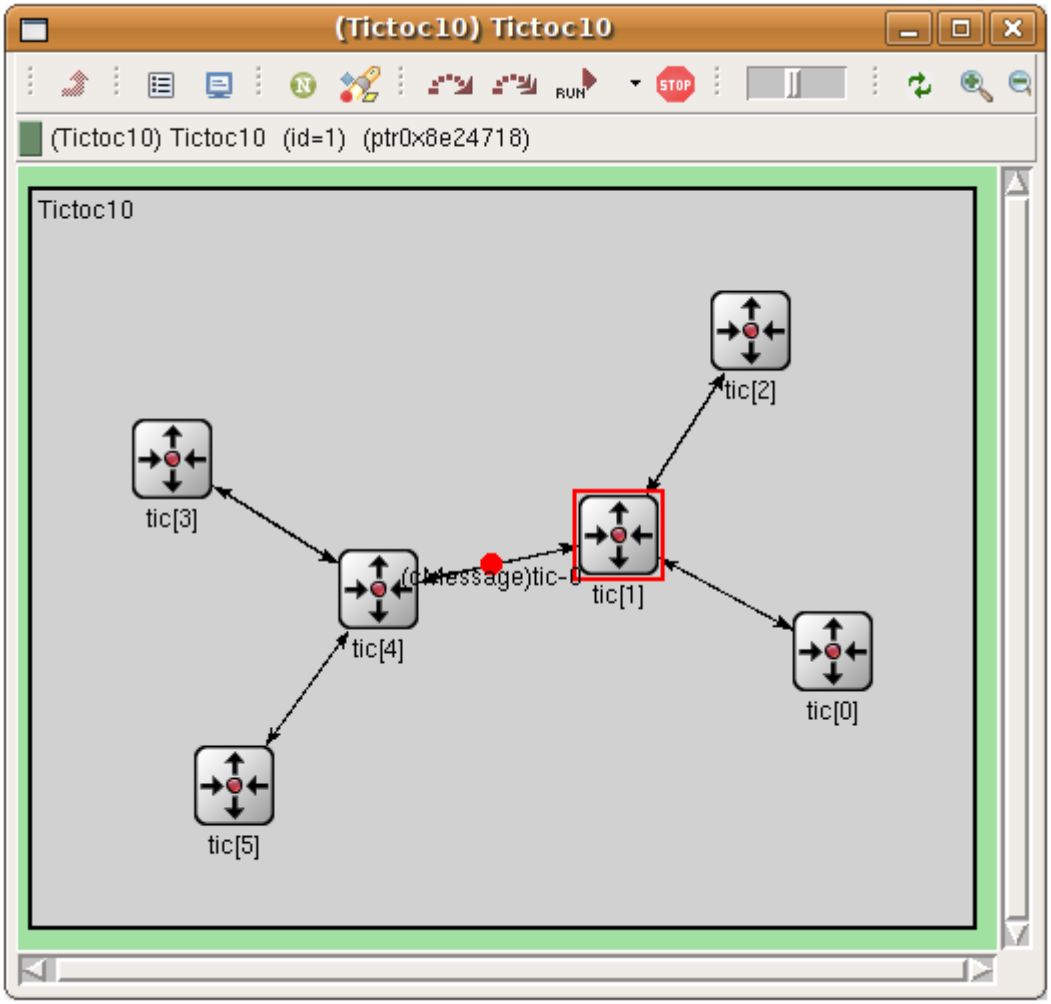
شکل ۳٤

کاراکترهای [] گیت ها را تبدیل به برداری از گیت ها می کند.اندازه بردار(تعداد گیت ها) زمانی تعیین می شود که از Txc برای ساخت شبکه استفاده کنیم:

```
network Tictoc10
{
    submodules:
        tic[6]: Txc10;
    connections:
        tic[0].out++ --> {  delay = 100ms; } --> tic[1].in++;
        tic[0].in++ <-- {  delay = 100ms; } <-- tic[1].out++;

        tic[1].out++ --> {  delay = 100ms; } --> tic[2].in++;
        tic[1].in++ <-- {  delay = 100ms; } <-- tic[2].out++;

        tic[1].out++ --> {  delay = 100ms; } --> tic[4].in++;
        tic[1].in++ <-- {  delay = 100ms; } <-- tic[4].out++;

        tic[3].out++ --> {  delay = 100ms; } --> tic[4].in++;
        tic[3].in++ <-- {  delay = 100ms; } <-- tic[4].out++;

        tic[4].out++ --> {  delay = 100ms; } --> tic[5].in++;
        tic[4].in++ <-- {  delay = 100ms; } <-- tic[5].out++;
}
```

شکل ۳۵

در اینجا ما ٦ ماژول با کمک یک بردار ماژول ایجاد کردیم و آنها را به هم متصل نمودیم.نتیجه این کار این شد:



شکل ۳۶

در این نسخه، tic[۰] پیامی را برای ارسال ایجاد می کند.

قسمت اصلی کد تابع forwardMessage() است که آن را هرزمان که پیامی به گره می رسد در تابع handleMessage() فراخوانی می کنیم.این تابع یک عدد تصادفی از شماره گیت ها تولید کرده و پیام را بر روی همان عدد ارسال می کند.

```
void Txc10::forwardMessage(cMessage *msg)
{
    // In this example, we just pick a random gate to send it on.
    // We draw a random number between 0 and the size of gate `out[]'.
    int n = gateSize("out");
    int k = intuniform(0,n-1);

    EV << "Forwarding message " << msg << " on port out[" << k << "]\n";
    send(msg, "out", k);
}
```

شکل ۳۷

www.NoavaranGermi.ir

زمانی هم که پیام به دست tic[۳] برسد،آن پیام را در handleMessage() خود حذف خواهد کرد.

٤-۳-۲- قدم ۱۱: کانال ها و تعریف های نوع داخلی

تعریف ساختار شبکه ما تا بدین جا بسیار پیچیده و طولانی شده است، این امر مخصوصا در مورد بخش ارتباطات صادق است.باید کمی آن را ساده کنیم.چیزی که در این بخش قابل توجه است وجود پارامتر delay در همه ارتباطات است.می توانیم نوع های جدیدی (شبیه نوع ماژول ساده) با نام channel برای ارتباطات ایجاد کنیم.یک نوع channel تعریف کرده و برای آن یک پارامتر delay تعیین می کنیم؛ حال در تمام شبکه این channel را با ارتباطات موجود تعویض می کنیم:

```
network Tictoc11
{
    types:
        channel Channel extends ned.DelayChannel {
            delay = 100ms;
        }
    submodules:
```

شکل ۳۸

همانطور که متوجه شدید نوع channel جدید را درون شبکه و با افزودن بخش types تعریف کردیم.این نوع تعریف شده تنها درون شبکه قابل رویت است که به آن نوع محلی یا داخلی می گویند.در صورت تمایل می توانید ماژول های ساده را نیز به شکل نوع های داخلی تعریف کنید.

حال بیایید تغییرات به وجود آمده در بخش connections را بررسی کنیم:

```
    connections:
        tic[0].out++ --> Channel --> tic[1].in++;
        tic[0].in++ <-- Channel <-- tic[1].out++;

        tic[1].out++ --> Channel --> tic[2].in++;
        tic[1].in++ <-- Channel <-- tic[2].out++;

        tic[1].out++ --> Channel --> tic[4].in++;
        tic[1].in++ <-- Channel <-- tic[4].out++;

        tic[3].out++ --> Channel --> tic[4].in++;
        tic[3].in++ <-- Channel <-- tic[4].out++;

        tic[4].out++ --> Channel --> tic[5].in++;
        tic[4].in++ <-- Channel <-- tic[5].out++;
}
```

شکل ۳۹

همانطور که مشاهده می کنید تنها نام کانال را درون تعریف ارتباطات مشخص کرده ایم.این کار به ما امکان می دهد تا به راحتی پارامتر تاخیر کل شبکه را تغییر دهیم.

٤-۳-۳- قدم ۱۲: استفاده از اتصالات دو طرفه

اگر بخش connections را کمی بیشتر بررسی کنیم، متوجه خواهیم شد که هر جفت از گره ها از طریق دو اتصال به یکدیگر متصل هستند.می توانیم از این امکان OMNeT++ نیز بهره بریم.برای انجام این کار ابتدا باید گیت های دو طرفه inout را جایگزین گیت های input و output کنیم.

```
simple Txc12
{
    parameters:
        @display("i=block/routing");
    gates:
        inout gate[];  // declare two way connections
}
```

شکل ٤۰

بخش connection به شکل زیر خواهد شد:

```
    connections:
        tic[0].gate++ <--> Channel <--> tic[1].gate++;
        tic[1].gate++ <--> Channel <--> tic[2].gate++;
        tic[1].gate++ <--> Channel <--> tic[4].gate++;
        tic[3].gate++ <--> Channel <--> tic[4].gate++;
        tic[4].gate++ <--> Channel <--> tic[5].gate++;
}
```

شکل ٤۱

با تغییر نام گیت ها باید کمی در کد C++ نیز تغییر ایجاد کنیم:

```
void Txc12::forwardMessage(cMessage *msg)
{
    // In this example, we just pick a random gate to send it on.
    // We draw a random number between 0 and the size of gate `gate[]'.
    int n = gateSize("gate");
    int k = intuniform(0,n-1);

    EV << "Forwarding message " << msg << " on gate[" << k << "]\n";
    // $o and $i suffix is used to identify the input/output part of a two way gate
    send(msg, "gate$o", k);
}
```

شکل ٤٢

٤-٣-٤- قدم ١٣: تعریف کلاس پیام

در این مرحله دیگر آدرس مقصد به صورت دستی مشخص نمی شود بلکه یک مقصد تصادفی تولید کرده و آدرس مقصد را به پیام اضافه می کنیم.

بهترین راه ساختن کلاسی مشتق شده از کلاس cMessage و افزودن مقصد به شکل data member به آن است.با تعیین مشخصات کلاس پیام در فایل msg دیگر نیازی به نوشتن خود کلاس پیام به زبان C++ نیست و OMNeT++ این قابلیت را دارد که کلاس مربوط به پیام را برایمان تولید کند.کلاس تولید شده برای هر یک از فیلدها متدهای getter و setter را دارد.نام فایل تعریف کلاس پیام ما که در زیر نشان داده شده است TicTocMsg١٣ است بنابراین کامپایلر پیام دو فایل به نامهای tictoc١٣_m.h و tictoc١٣_m.cc بریمان تولید خواهد کرد.

```
message TicTocMsg13
{
    int source;
    int destination;
    int hopCount = 0;
}
```

شکل ٤٣

حال باید tictoc١٣_m.h را در کد C++ خود include کنیم و پس از آن می توانیم از TicTocMsg١٣ مانند هر کلاس دیگر استفاده کنیم:

```
#include "tictoc13_m.h"
```

برای مثال در زیر که بخشی از تابع generateMessage() است نحوه ایجاد یک پیام و پر کردن فیلد های آن نشان داده شده است:

```
TicTocMsg13 *msg = new TicTocMsg13(msgname);
msg->setSource(src);
msg->setDestination(dest);
return msg;
```

شکل ٤٤

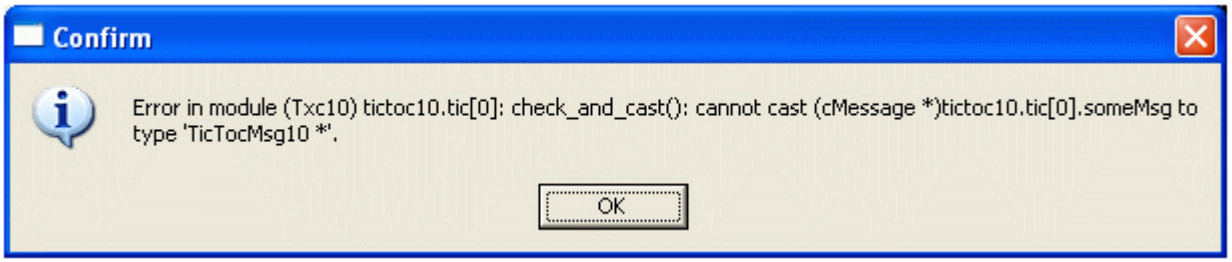
تابع handleMessage() نیز بدین شکل شروع می شود:

```
void Txc13::handleMessage(cMessage *msg)
{
    TicTocMsg13 *ttmsg = check_and_cast<TicTocMsg13 *>(msg);

    if (ttmsg->getDestination()==getIndex())
```

شکل ٤٥

در آرگمان تابع handleMessage() اشاره گری به کلاس cMessage دریافت کرده ایم.در صورتی که msg دریافت شده را به صورت عادی به اشاره گری از TicTocMsg۱۳ تبدیل(cast)کنیم امن نخواهد بود زیرا در صورتی که msg دریافت شده به هر دلیل از نوع TicTocMsg۱۳ نباشد برنامه کرش کرده و یافتن علت خطا بسیار سخت خواهد شد.

C++ راه حل دیگری به نام dynamic_cast معرفی کرده است.در این جا ما از check_and_cast<>() استفاده کرده ایم که توسط OMNeT++ فراهم شده است که سعی می کند اشاره گر را از طریق dynamic_cast تبدیل (cast) کند و در صورتی که با شکست مواجه شود شبیه سازی را با پیغام خطایی مشابه آنچه در زیر آمده است متوقف می کند:

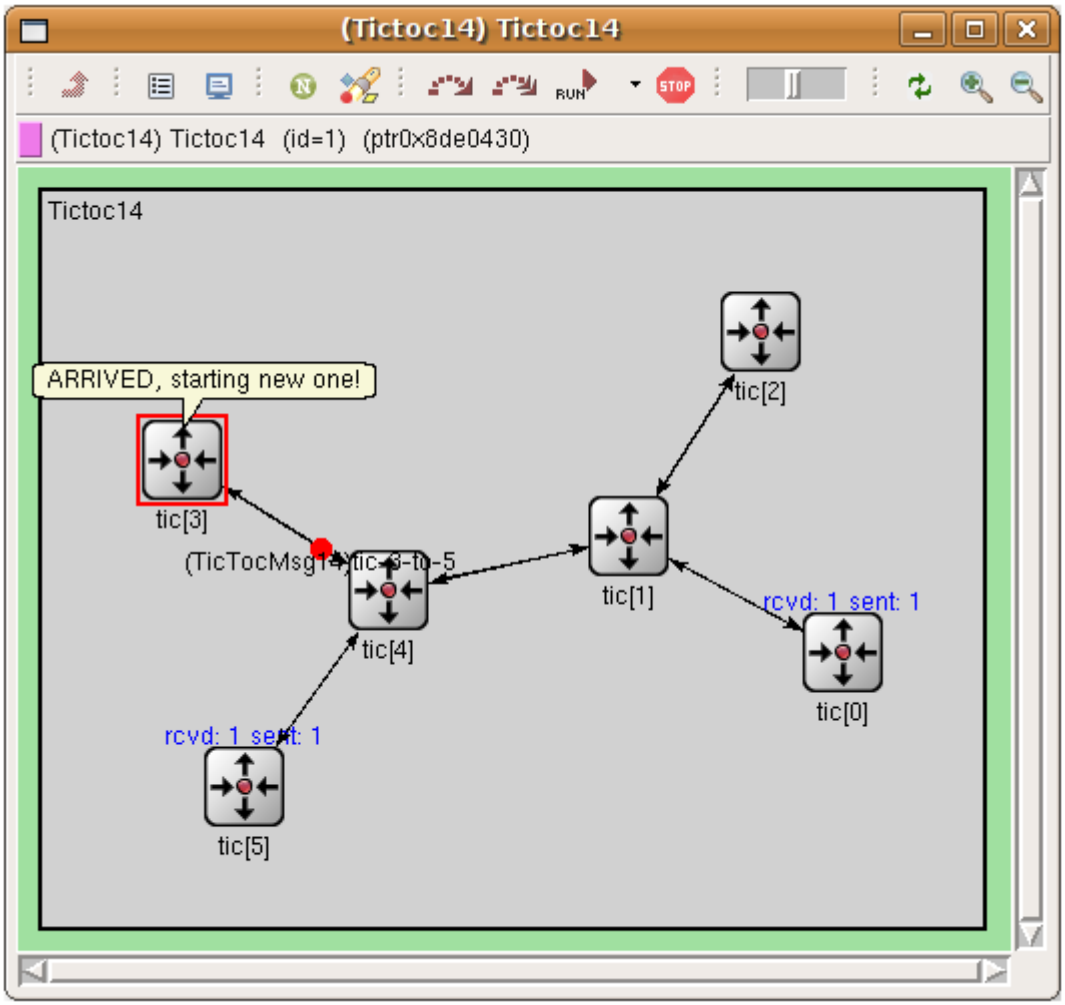


شکل ٤٦

در خط بعد از کد نشان داده شده در بالا،بررسی می کنیم که آیا آدرس مقصد با آدرس گره یکسان است یا خیر.تابع عضو getIndex() اندیس ماژول در بردار زیرماژول را برمی گرداند(فایل NED را به خاطر بیاورید).

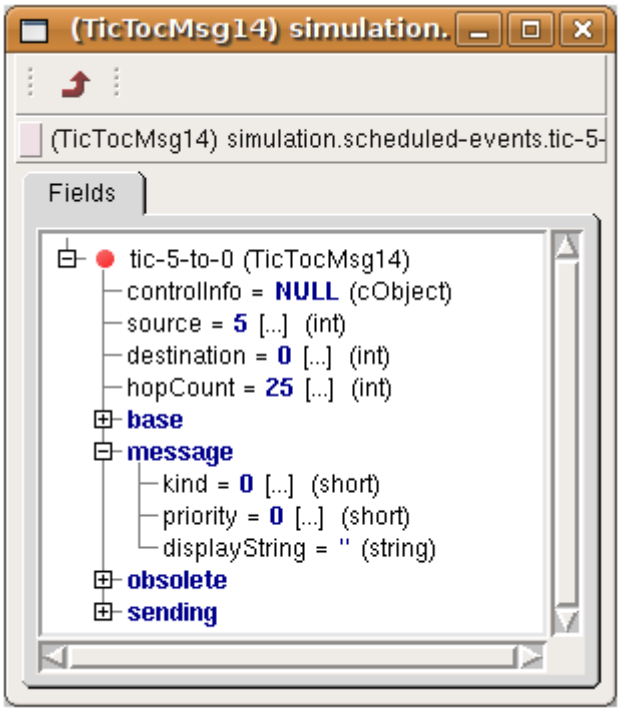
برای اجرای طولانی مدت تر مدل، پس از آنکه پیامی به مقصد خود باز می گردد، گره مقصد پیامی دیگر با یک آدرس مقصد تصادفی تولید کرده و به همین ترتیب این کار ادامه می یابد.

اگر مدل را اجرا نمایید، خروجی زیر را خواهید دید:



شکل ٤٧

می توانید بر روی پیامها دابل کلیک کرده تا یک پنجره inspector برایشان باز شود.این پنجره اطلاعات ارزشمند زیادی در اختیار شما خواهد گذاشت.از جمله این اطلاعات می توان به نام پیام، مبدا و مقصد پیام، تعداد دفعاتی که این پیام بین گره ها جابجا شده است و ... اشاره کرد.



شکل ۴۸

۴-۴- افزودن مجموعه های آماری

٤-٤-١- قدم ١٤: نمایش شماره پکت های ارسالی/دریافتی

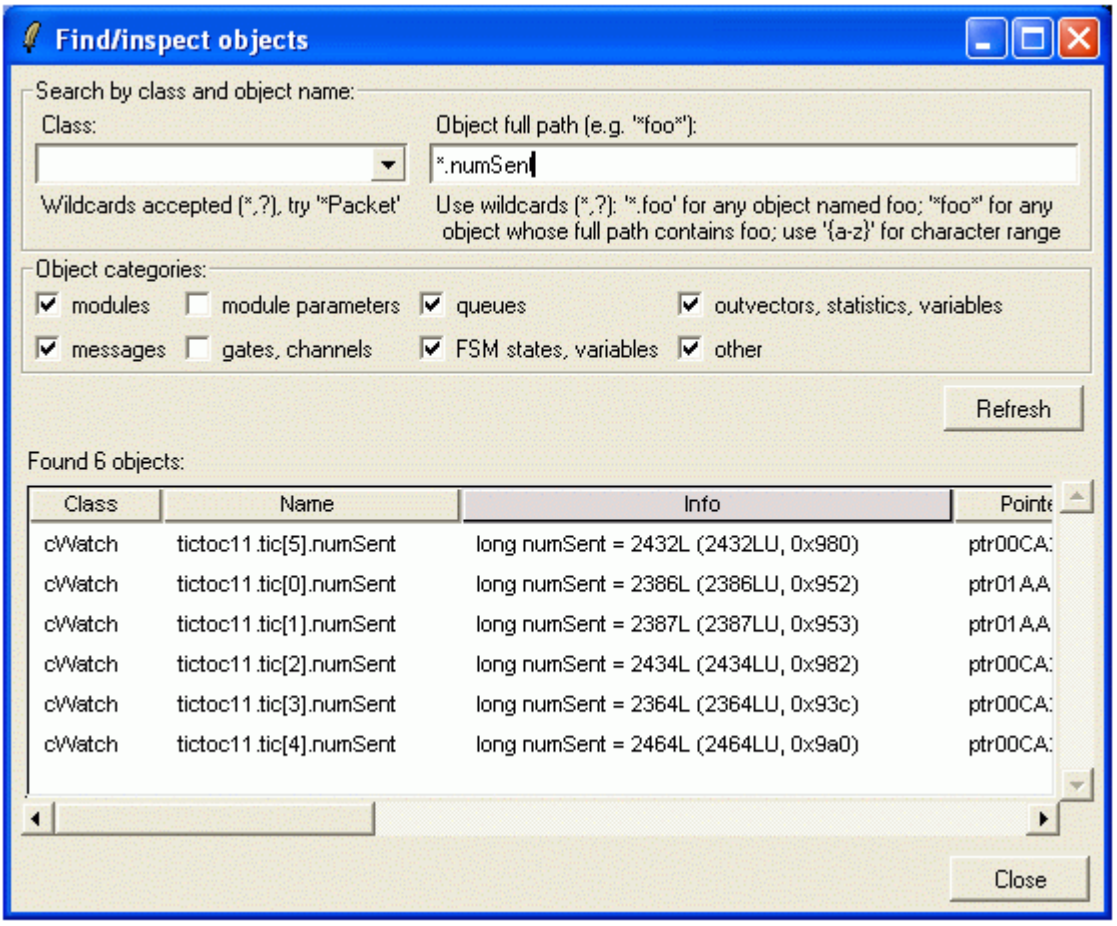
برای دیدن تعداد پیامهایی که هر گره ارسال یا دریافت کرده است، دو شمارنده را به کلاس ماژول اضافه کردیم:

numSent و numReceived.

```
class Txc14 : public cSimpleModule
{
  private:
    long numSent;
    long numReceived;

  protected:
```

شکل ٤٩

این شمارنده ها در متد ()initialize با صفر مقداردهی شده و در WATCH قرار گرفتند.حال می توانیم از پنجره ی محاوره ای Find/inspect objects(از منوی Inspect و یا در toolbar نیز قرار دارد) استفاده کنیم تا تعداد ارسالها و دریافت های گره های مختلف را بدست آوریم.



شکل ۵۰

درست است که در این مدل شبیه سازی شده تعداد ارسالی و دریافتی یکسان است اما در شبیه سازی های واقعی این قابلیت که بتوان مروری سریع بر وضعیت گره های مختلف در مدل داشت بسیار ارزشمند خواهد بود.

همچنین این اطلاعات را می توان در بالای آیکن هر ماژول نیز نشان داد.رشته t در display متن موجود در بالای آیکن را مشخص می کند؛تنها باید این رشته را در زمان اجرا تغییر دهیم.کد زیر این کار را انجام میدهد:

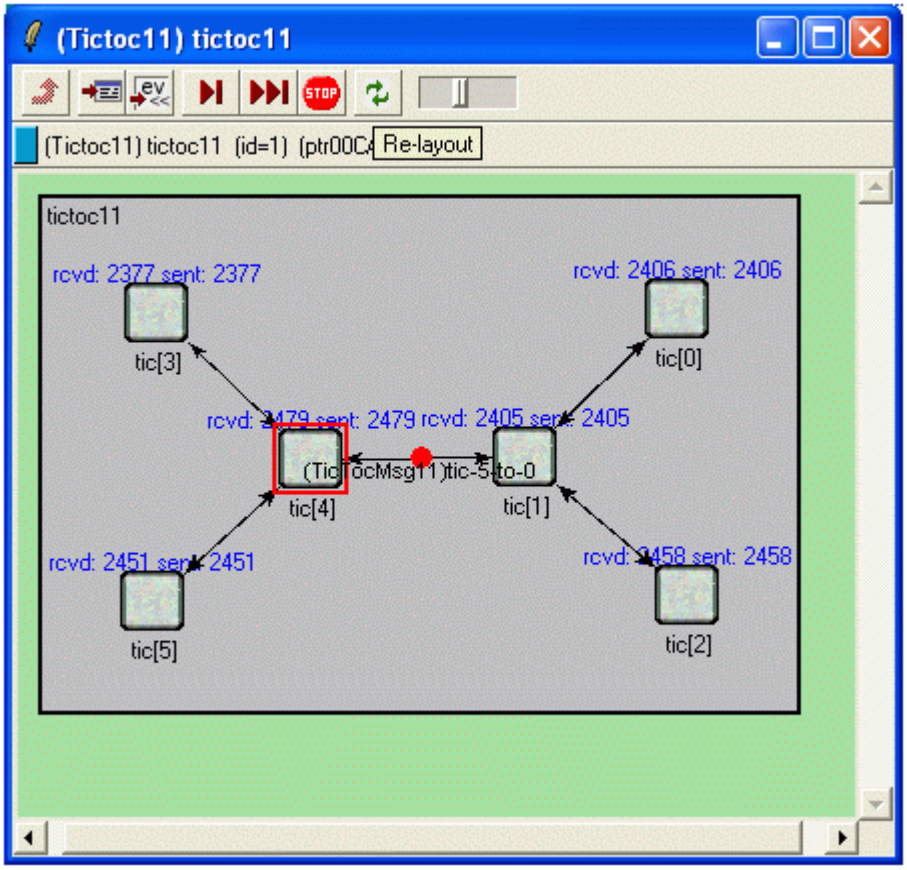
```
if (ev.isGUI())
    updateDisplay();
```

در این شرط بررسی می کنیم که آیا GUI فعال است یا خیر؛در صورت غیر فعال بودن اجرای تابع updateDisplay() با خطا مواجه می شود.مثلا غیرفعال بودن GUI زمانی اتفاق می افتد که در حین اجرای برنامه تنها پنجره ای که نمایش دهنده ی گرافیک مدل است(و نه پنجره ی اصلی Tkenv) را ببندیم.در این صورت برنامه برنامه بسته نمی شود تنها GUI آن بسته شده و غیر فعال می گردد.

```
void Txc14::updateDisplay()
{
    char buf[40];
    sprintf(buf, "rcvd: %ld sent: %ld", numReceived, numSent);
    getDisplayString().setTagArg("t",0,buf);
}
```

شکل ۵۱

و نتیجه شکل زیر خواهد شد:



شکل ۵۲

www.NoavaranGermi.ir

٤-٤-٢- قدم ١٥: افزودن مجموعه های آماری

برای جمع آوری داده ها و تحلیل آماری آنها به وسیله OMNeT++ سه ابزار در اختیار داریم:

١. Event log
٢. Output Vector
٣. Output Scalars

که در ادامه هر یک را مختصرا توضیح می دهیم.

٤-٤-٢-١- Eventlog

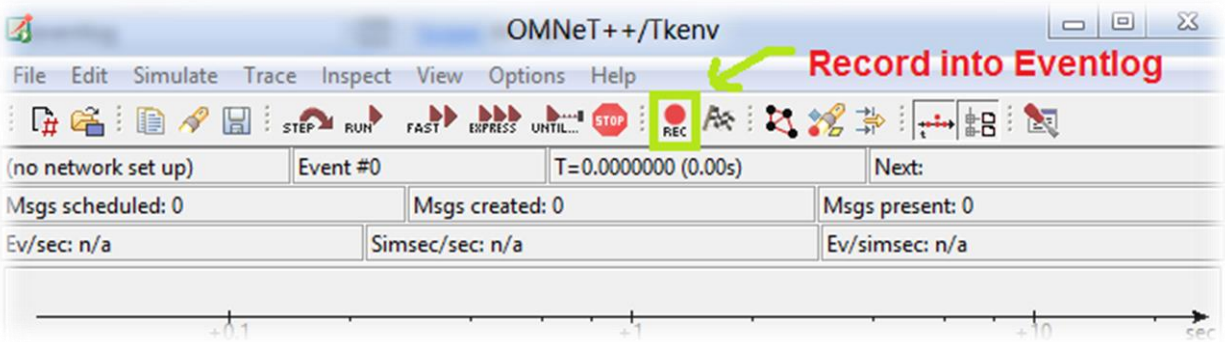
Eventlog فایلی با پسوند elog. است.این فایل با انجام تنظیماتی در فایل ini به صورت خودکار پس از انجام شبیه سازی توسط OMNeT++ تولید شده و در پوشه ای به نام results در پوشه ی برنامه قرار میگیرد.این قابلیت در OMNeT++ ٤,٠ معرفی شد و در نسخه های قبلی وجود ندارد.هدف از ایجاد این قابلیت در OMNeT++ درک آسانتر مدل های شبیه سازی های پیچیده بود زیرا با آن می توان رفتار کل شبکه را زیر نظر گرفت و به نظر خود من می توان آنرا معادل trace در برنامه نویسی دانست!زیرا با استفاده از آن میتوان لحظه به لحظه ی شبیه سازی را از ابتدا تا انتها و برعکس از انتها به ابتدا بررسی کرد و مرحله به مرحله و قدم به قدم با آن پیش رفت.

در Eventlog داده های بدست آمده را می توان فیلتر کرد تا بتوان بر روی یک رخداد خاص مثلا تنها پیام هایی که به یک گره ارسال شده یا از آن به دیگر گره ها ارسال شده است تمرکز کرد.به نوعی می توان به آن دستور داد که تنها داده هایی را که به دنبال آن هستید نمایش دهد.همین امر باعث می شود که بتوان وابستگی های مختلف در بین پیام ها مثلا وابستگی علت و معلولی (causality) که در سیستم عامل بسیار برای ما مهم است را به راحتی بررسی نمود و یا بر اساس زمان شبیه سازی، شماره رخدادها(مثلا ارسال یا دریافت و یا پردازش یک پیام یک رخداد محسوب می شود)، ماژولها و پیام ها، فیلترهایی را در این لاگ اعمال کرد و هر یک را به تنهایی تحلیل نمود.

هسته شبیه ساز OMNeT++ می تواند با تنظیم دستور زیر در فایل omnetpp.ini، لاگ دقیق خودکاری از تبادل پیام ضبط کند:

```
record-eventlog = true
```

همچنین می توان این کار را در Tkenv با فشردن دکمه Rec در toolbar انجام داد:

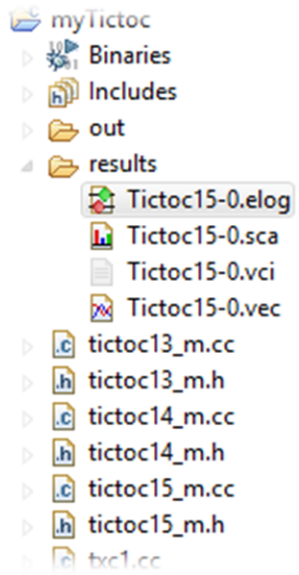


شکل ۵۳

در مورد مثال Tictoc، پس از اتمام شبیه سازی در پوشه results درون پوشه برنامه، فایل لاگی با نام Tictoc۱۵-۰.elog ایجاد می شود. این لاگ را در IDE میتوان به کمک نمودارتوالی[1] نمایش داد.دقت کنید که در صورتی که تبادل پیام در سیستم زیاد باشد این فایل واقعا بزرگ خواهد شد پس تنها در صورتی که بدان نیاز دارید این امکان را فعال کنید.

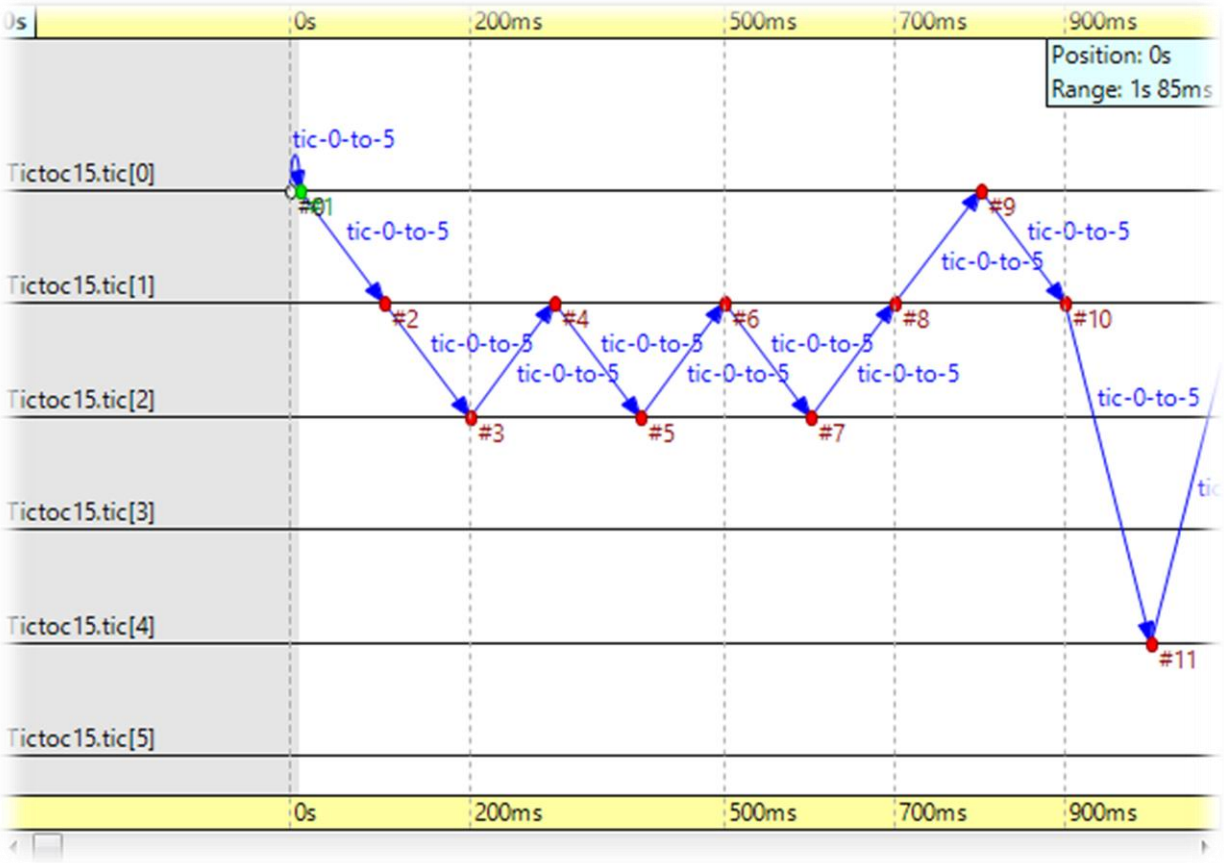
برای مشاهده ی نمودار توالی ابتدا باید در IDE بر روی فایل elog.* دابل کلیک نمایید:

[1] Sequence Chart



شکل ٥٤

تصویر بالا مربوط به مثال Tictoc می باشد.با کلیک بر روی این فایل نمودار توالی نمایش داده می شود:



شکل ٥٥

همانطور که مشاهده می کنید محور افقی این نمودار نشان دهنده ی زمان می باشد و در سمت چپ نمودار و در محور عمودی نام ماژول های موجود در شبکه نشان داده می شود.در بخش اصلی این نمودار به وسیله ی فلشهای آبی تبادل پیام مابین ماژول ها نشان داده می شود.برای مثال در نمودار بالا فلش نشان می دهد که tic[۰] ابتدا پیام tic-۰-to-۵ را برای خود ارسال کرده است و پس از دریافت آن،فلش آبی مستقیم نشان دهنده ی آن است که پیام را به tic[۱] فرستاده است.همچنین در این نمودار شماره ی هر رویداد با علامت # مشخص شده است که رنگ آن وابسته به نوع رویداد تغییر می کند.نمودار به صورت افقی رشد می کند و به دلیل آنکه تصویر کلی آن بسیار بزرگ می باشد در اینجا تنها بخشی از آن نشان داده شده است که محدوده ی نمایش داده شده بوسیله ی کادر سبزرنگ گوشه بالا سمت راست تصویر مشخص می شود.در مثال بالا تصویر از زمان ۰ شروع شده و تا زمان ۱ ثانیه و ۸۵ میلی ثانیه ادامه می یابد.IDE نرم افزار OMNeT++ ابزارهای زیادی را برای کار با این نمودار ارائه کرده است.

٤-٤-٢-٢- راه هایی برای کوچک کردن حجم لاگ و بررسی بخشی از شبیه سازی

برای کم کردن حجم فایل می توان به کرنل دستور داد تا تنها در دوره های زمانی مشخص رویدادها را ذخیره کند.

مثال:

```
eventlog-recording-intervals = ..10.2, 22.2..100, 233.3..
```

به طور پیش فرض رویداد های تمام ماژولها ذخیره میشود.فاکتور دیگری که میتوان برای کوچک کردن اندازه لاگ انجام داد، مشخص کردن ماژول هایی است می خواهید رفتار آنها را ضبط کنید است.با آپشن module-eventlog-recording به کرنل دستور می دهد تا تنها رویدادهایی که برای ماژول مشخص شده رخ می دهد را ذخیره کند.این آپشن تنها بر روی ماژولهای ساده قابل اعمال است.

www.NoavaranGermi.ir

مثال:

```
**.router[10..20].**.module-eventlog-recording = true
**.module-eventlog-recording = false
```

این مثال به کرنل دستور میدهد که تنها رویدادهای روتر هایی که اندیس آنها بین ۱۰ تا ۲۰ است را ذخیره کند و بقیه ضبط رویداد را برای دیگر ماژولها غیر فعال نماید.

٤-٤-٢-٣- Output Vector - Output Scalars

OMNeT++ از طریق output vectors و output scalars امکان ضبط نتیجه ی شبیه سازی را فراهم کرده است.بردارهای خروجی، داده های سری های زمانی می باشند که از ماژولهای ساده و یا کانال ها ضبط شده اند و می توان از آنها برای ضبط تاخیرهای از مبدا به مقصد(end – to – end) یا زمان رفت و برگشت پکت ها،طول صف،وضعیت ماژول،از بین رفتن پکت و هر چیزی که لازم باشد یک تصویر کامل از آن در طول اجرای شبیه سازی بدست آوریم، استفاده کرد.

خروجی های اسکالر خلاصه ی نتایجی هستند که در طول شبیه سازی محاسبه و زمانی که شبیه سازی کامل شد در فایل نوشته می شوند.یک نتیجه ی اسکالر ممکن است یک عدد(صحیح یا حقیقی) باشد یا یک نتیجه آماری متشکل از چندین فیلد مانند شماره،میانگین،انحراف معیار،جمع،حداقل،حداکثر و ... باشد.

هم فایل های خروجی بردار و هم اسکالر، فایلهایی متنی می باشند.مزیت فرمت متنی آن است که بوسیله ی تعداد زیادی از ابزارها و زبان ها قابل دسترسی می باشد.

اشیایی از نوع cOutVector مسئول نوشتن داده های سری های زمانی بر روی یک فایل می باشند که به آنها بردارهای خروجی (Output Vectors) می گویند.از متد ()record برای نوشتن یک مقداربه همراه مهرزمانی(timestamp) استفاده می شود.

در صورتی که چندین شئ cOutVector داشته باشیم، تمام اشیاء در یک فایل وکتور خروجی واحد داده های خود را می نویسند.پسوند این فایل vec. است.به همراه این فایل در پوشه results فایل دیگری به نام vci. نیز تولید می شود که یک فایل کمکی می باشد و

www.NoavaranGermi.ir

درصورت پاک شدن آن دوباره توسط IDE تولید خواهد شد.می توانید تنظیمات وکتورهای خروجی را در omnetpp.ini انجام دهید: مثلا می توانید ضبط داده ها را تنها در یک دوره ی خاص زمانی انجام دهید.

درحالی که وکتورهای خروجی برای ضبط داده های سری های زمانی هستند و حجم بزرگی از داده های زمان شبیه سازی را ذخیره می کنند، خروجی های اسکالر(output scalars) برای ضبط یک مقدار واحد در هر بار شبیه سازی در نظر گرفته شده اند.

با استفاده از خروجی های اسکالر می توانید کارهای زیر را انجام دهید:

- ضبط اطلاعات خلاصه در انتهای اجرای شبیه سازی
- چندین بار یک شبیه سازی را با پارامترها و دانه تصادفی اجرا کرده و وابستگی برخی اندازه ها را با تنظیمات پارامترها بررسی کنید و به نوعی اجراهای مختلف را با هم مقایسه نمایید.

خروجی های اسکالر با متد ()record از کلاس cSimpleModule بر روی یک فایل باپسوند sca. ضبط می گردند. اغلب این متد را درون تابع ()finish قرار می دهند.

٤-٤-٢-٣-١- استفاده از بردارهای خروجی و اسکالرهای خروجی در مثال TicToc

در مدل شبیه سازی قبل اطلاعاتی داشتیم که می توان داده های آماری آنها را جمع آوری کرد.برای مثال، می توان میانگین تعداد یالهایی را که یک پیام برای رسیدن به مقصد باید طی کند، بدست آورد.

ما تعداد گام های(hop) هر پیام را تا زمانی که به مقصد برسد در یک بردار خروجی(دنباله ای از جفت های (زمان،مقدار)) ضبط می کنیم.همچنین مقادیر توابع میانگین،انحراف معیار(standard deviation)، حداقل، حداکثر را برای هر گره در رابطه با پیام های ارسال شده از آن محاسبه کرده و در انتهای شبیه سازی در یک فایل می نویسیم و سپس از ابزارهایی در OMNeT++ IDE برای آنالیز فایلهای خروجی استفاده می کنیم.

برای این کار، یک شئ بردار خروجی(cOutVector – که داده ها را در Tictoc١٥-٠.vec ذخیره خواهد کرد) و یک شئ نمودار(cLongHistogram – که میانگین، حداقل و... را

محاسبه می کند و داده ها را در Tictoc۱۵-۰.sca ذخیره می کند) به کلاس ماژول اضافه می کنیم.

```
class Txc15 : public cSimpleModule
{
  private:
    long numSent;
    long numReceived;
    cLongHistogram hopCountStats;
    cOutVector hopCountVector;

  protected:
```

شکل٥٦

زمانی که یک پیام به گره مقصد می رسد، آمار را به روز می کنیم.کد زیر به handleMessage() اضافه شد:

```
hopCountVector.record(hopcount);
hopCountStats.collect(hopcount);
```

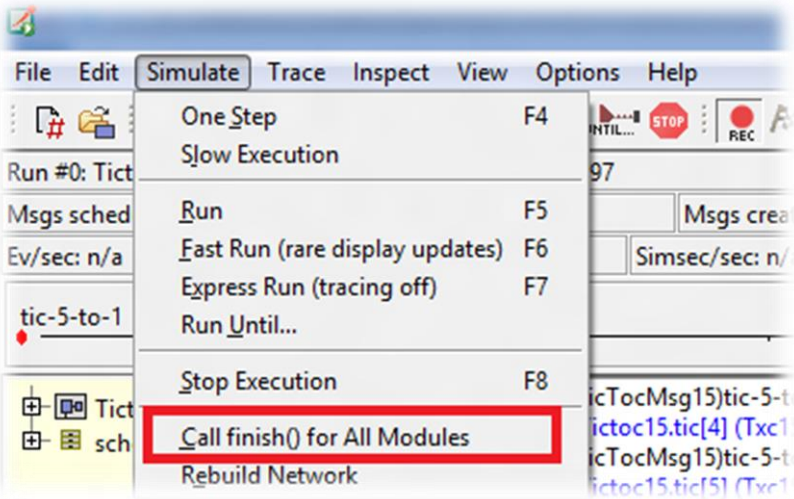
فراخوانی hopCountVector.record() داده ها را در Tictoc۱۵-۰.vec می نویسد.از آنجاییکه اجرای طولانی مدت این شبیه سازی و یا شبیه سازی یک مدل بزرگ باعث حجیم شدن فایل vec خواهد شد، برای مدیریت این شرایط می توان در omnetpp.ini بردار را فعال یا غیرفعال کرد، همچنین می توان بازه هایی از شبیه سازی را برای ضبط داده مشخص کرد و دیگر داده های زائد را دور ریخت.

داده های اسکالر که توسط شئ histogram در این شبیه سازی جمع آوری شده اند باید به طور دستی در متد finish() ضبط گردند.این تابع بعد از اتمام شبیه سازی و درصورت اجرای موفق آن فراخوانی می گردد و در صورتی که شبیه سازی با خطا متوقف شود این تابع فرخوانی نمی شود.فراخوانی تابع recordScalar() در کد زیر باعث نوشته شدن اطلاعات در Tictoc۱۵-۰.sca می گردد.

```
void Txc15::finish()
{
    // This function is called by OMNeT++ at the end of the simulation.
    EV << "Sent:     " << numSent << endl;
    EV << "Received: " << numReceived << endl;
    EV << "Hop count, min:    " << hopCountStats.getMin() << endl;
    EV << "Hop count, max:    " << hopCountStats.getMax() << endl;
    EV << "Hop count, mean:   " << hopCountStats.getMean() << endl;
    EV << "Hop count, stddev: " << hopCountStats.getStddev() << endl;

    recordScalar("#sent", numSent);
    recordScalar("#received", numReceived);

    hopCountStats.recordAs("hop count");
}
```

شکل ۵۷

هنگامی که فکر می کنید به اندازه ی کافی داده جمع آوری شد، می توانید شبیه سازی را متوقف کرده و به آنالیز فایل های بدست آمده بپردازید.برای اجرای تابع finish() و نوشته شدن داده ها درون فایل Tictoc۱۵-۰.sca باید از منوی پنجره ی اصلی Simulate و سپس Call finish() for All Modules را انتخاب کنید.



شکل ۵۸

در مثال Tictoc پس از گذشته ۱۰۰ ثانیه تابع finish() را اجرا و برنامه را متوقف کرده و خروجی زیر را دریافت کردیم:

www.NoavaranGermi.ir

```
Forwarding message (TicTocMsg15)tic-5-to-2 on gate[0]
** Calling finish() methods of modules
Sent:    18
Received: 18
Hop count, min:    2
Hop count, max:    60
Hop count, mean:   11.8333
Hop count, stddev: 13.6952
Sent:    19
Received: 19
Hop count, min:    1
Hop count, max:    18
Hop count, mean:   3.89474
Hop count, stddev: 4.7128
Sent:    11
Received: 11
Hop count, min:    1
Hop count, max:    17
Hop count, mean:   8.63636
Hop count, stddev: 5.46393
Sent:    15
Received: 15
Hop count, min:    1
Hop count, max:    45
Hop count, mean:   17.2
Hop count, stddev: 14.3288
Sent:    18
Received: 18
Hop count, min:    1
Hop count, max:    19
Hop count, mean:   3.5
Hop count, stddev: 4.80502
Sent:    20
Received: 20
Hop count, min:    1
Hop count, max:    78
Hop count, mean:   14.8
Hop count, stddev: 19.8855
```

شکل ۵۹

این اطلاعات پس از اجرای تابع **finish()** برای هر یک از ماژولها نشان داده شده است.

٤-٤-٢-٣-٢- توضیح تابع **finish()**:

در این مثال ٦ ماژول ساده از نوع Txc١٥ تعریف کردیم که هر یک دارای متغییرهای numSent، numReceived هستند که مشخص کننده ی تعداد پیام های دریافتی و ارسالی آنها می باشد(که با هم برابرند).همچنین برای هر ماژول شئ از نوع cLongHistogram به نام hopCountStats وجود دارد.در این تابع ابتدا در خروجی تعداد پیام های دریافتی و ارسالی چاپ شده و سپس با استفاده از توابع شئ hopCountStats آمار گامهای طی شده مربوط به پیام های رسیده به هر ماژول را در خروجی نمایش می دهد:

www.NoavaranGermi.ir

- تابع ()getMin: کمترین تعداد گامهایی که یک پیام دریافتی برای رسیدن به این ماژول طی کرده است را نشان میدهد.
- تابع ()getMax: بیشترین تعداد گامهایی که یک پیام دریافتی برای رسیدن به این ماژول طی کرده است را نشان میدهد.
- تابع ()getMean: متوسط تعداد گامهایی که همه ی پیامهای دریافت شده توسط این ماژول، طی کرده اند را نشان میدهد.
- تابع ()getStddev: انحراف از معیار تعداد گامهای طی شده توسط پیامهای دریافتی این ماژول را نشان میدهد.

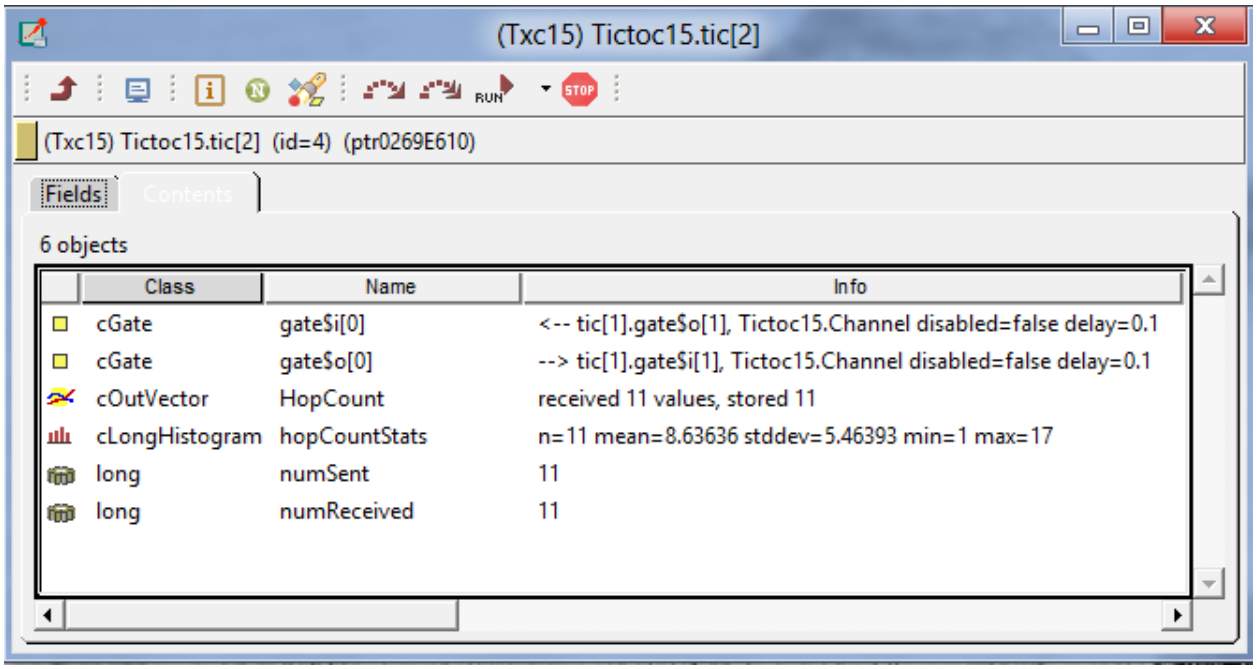
در تابع finish سپس مقدار متغیر های numSent و numReceived به وسیله ی تابع recordScalar ذخیره شده اند و در خط بعد داده های آماری شئ hopCountStats از طریق تابع recordAs با عنوان "hop count" در فایل Tictoc۱۵-۰.sca ذخیره می گردند.از آنجا که فایل اسکالر یک فایل متنی است می توان به راحتی محتویات آنرا مشاهده کرد. برای نمونه داده های نشان داده شده در زیر که بخشی از فایل Tictoc۱۵-۰.sca هستند داده های آماری مرتبط با tic[۲] را پس گذشت زمان ۱۰۰ ثانیه نشان می دهد.این ماژول ۱۱ پیام دریافت و ارسال کرده است.میانگین تعداد گامهایی که همه ی پیامهای دریافت شده توسط این ماژول، طی کرده اند ۸,٦۳٦۳٦۳ میباشد. انحراف از معیار تعداد گامهای طی شده توسط پیامهای دریافتی این ماژول حدودا ٥,٤٦۳۹۳۱ است و مجموع تعداد گامهای ۱۱ پیامی که توسط این ماژول دریافت شده است ۹۵ می باشد. کمترین تعداد گامهایی که یک پیام برای رسیدن به این ماژول طی کرده است ۱ گام می باشد و بیشترین گام طی شده ۱۷ است.

```
scalar Tictoc15.tic[2]  #sent   11
scalar Tictoc15.tic[2]  #received   11
statistic Tictoc15.tic[2]   "hop count"
field count 11
field mean 8.6363636363636
field stddev 5.4639313186153
field sum 95
field sqrsum 1119
field min 1
field max 17
```

شکل ٦۰

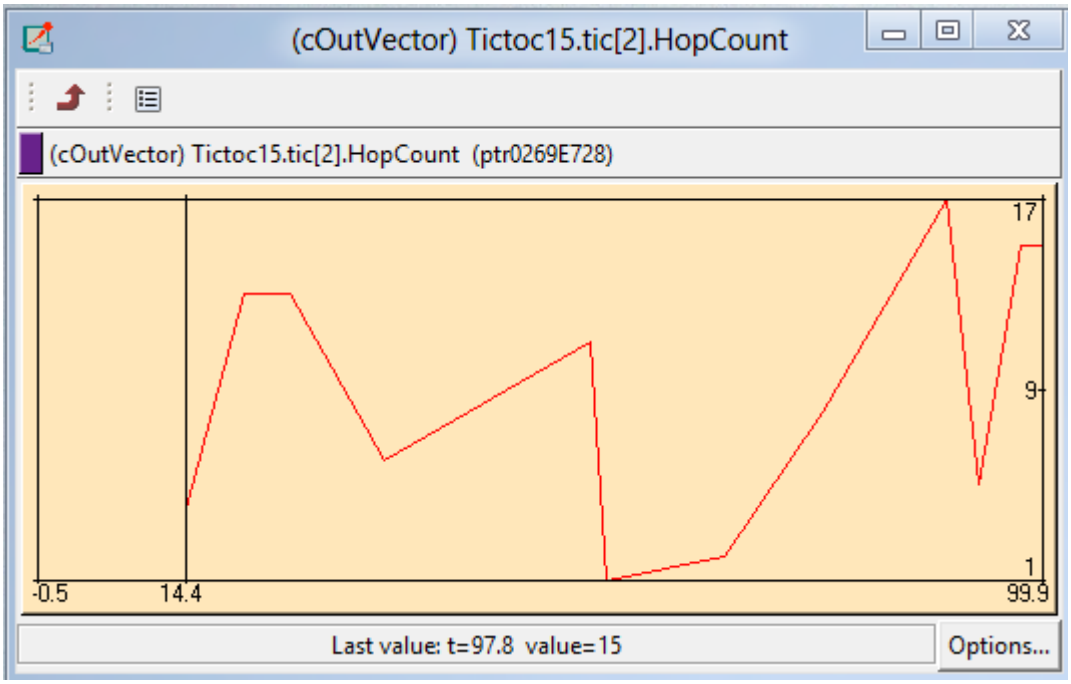
٤-٤-٢-٣-٣- نمایش داده در زمان اجرای شبیه سازی

داده ها را می توانید در زمان شبیه سازی نیز مشاهده کنید.برای این کار باید بر روی ماژول دلخواه خود در زمان شبیه سازی دابل کلیک کرده تا پنجره ی module inspector باز شود.برای مثال ما بر آیکن tic[۲] کلیک کرده و پنجره ی زیر باز می شود:



شکل ٦١

حال می توانید اشیائ hopCountStats و hopCount را پیدا کرده و با دابل کلیک بر روی آنها، inspectors آنها را باز نمایید.در ابتدا این اشیاء مقداری ندارند اما در صورتی که شبیه سازی را با سرعت زیاد اجرا نمایید سریعا داده های زیادی جمع آوری می گردد و نمایی مانند زیر را مشاهده خواهید کرد:



شکل ٦٢

این نمودار که برای tic[۲] رسم شده است در محور افقی خود نشان دهنده ی زمان و در محور عمودی نشان دهنده ی مقدار می باشد که مقدار در این جا همان تعداد گام است.همان گونه که مشاهده می کنید زمان این نمودار از ۰,۵- شروع گشته است(می توانید به resize نمودن پنجره این مقدار را بر روی صفر تنظیم کنید) و تا ۹۹,۹ ادامه پیدا کرده زیرا مدت زمان شبیه سازی ۱۰۰ بود.نمودار که با رنگ قرمز نشان داده شده است دارای نقطه ماکزیمم ۱۷ می باشد.در صورتی که به اطلاعات نمایش داده شده در خروجی که در بالا نشان داده شد دقت نمایید متوجه خواهید شد این میزان همان بیشترین گامهای طی شده یک پیام برای رسیدن به ماژول tic[۲] می باشد.همچنین تعداد خط های این نمودار که نقاط نمودار را به یکدیگر متصل می کند ۱۱ عدد می باشد که دوباره با کمی توجه، متوجه خواهید شد این نیز همان تعداد کل پیام های دریافتی توسط tic[۲] می باشد.

در ادامه اطلاعاتی که در فایل Tictoc۱۵-۰.sca دیده می شود(آن را به صورت متنی باز نمایید)،مجموعه داده هایی با عنوان attr type int است:

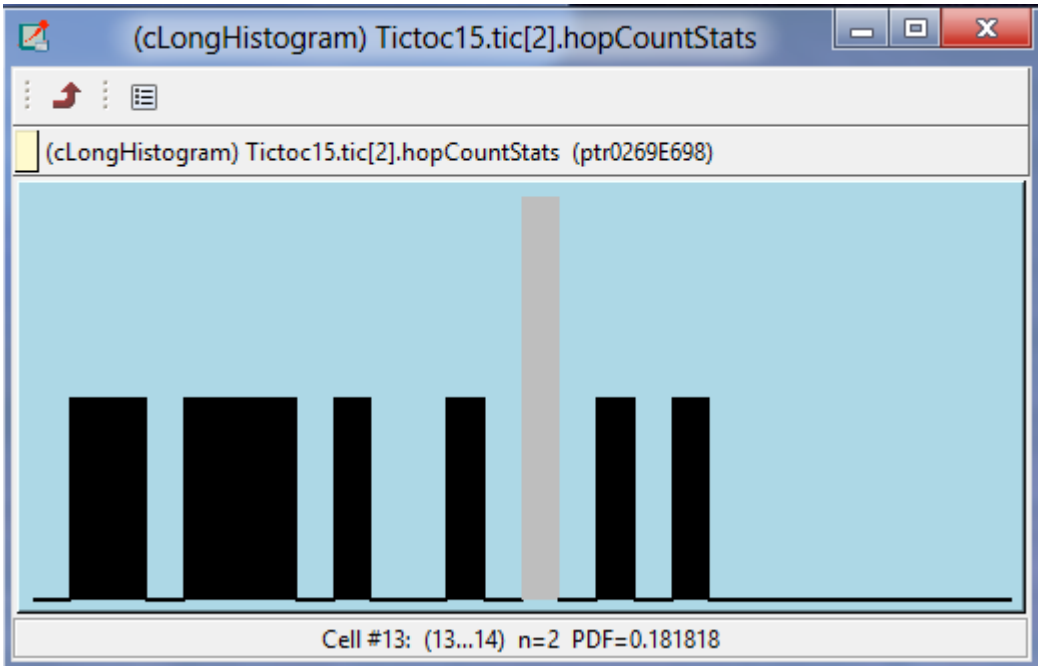
www.NoavaranGermi.ir

```
attr type  int
bin -INF    0
bin 0   0
bin 1   1
bin 2   1
bin 3   0
bin 4   1
bin 5   1
bin 6   1
bin 7   0
bin 8   1
bin 9   0
bin 10  0
bin 11  1
bin 12  0
bin 13  2
bin 14  0
bin 15  1
bin 16  0
bin 17  1
bin 18  0
bin 19  0
bin 20  0
bin 21  0
bin 22  0
bin 23  0
bin 24  0
bin 25  0
bin 26  0
```

در مثال Tictoc این داده ها نمایش دهنده ی فراوانی تعداد گامهای طی شده توسط پیام ها برای رسیدن به مقصد می باشند.برای مثال اطلاعات تصویر بالا که مربوط به ماژول tic[۲] است نشان می دهد دو پیام با طی کردن ۱۳ گام به این ماژول رسیده اند.همچنین در صورتی که تمام اعداد ستون سوم را با هم جمع بزنید برابر خواهد بود با تعداد کل پیام های دریافتی توسط این ماژول(یعنی ۱۱).

در زمان اجرای شبیه سازی و همچنین در پایان آن می توانید با کلیک بر روی hopCountStats در تب Contents مربوط به هر ماژول نمودار توزیع فراوانی تعداد گام های پیام های رسیده به آن ماژول را مشاهده نمایید.برای مثآل شکل زیر نمودار توزیع فراوانی پیام های رسیده به tic[۲] است که در بالا اطلاعات مربوط به آن را با هم بررسی کردیم:

www.NoavaranGermi.ir



شکل ٦٣

این نمودار دارای ٢٦ ستون است که نه ستون آن دارای ارتفاع ١ و یک ستون آن که همان ستون ١٣ است دارای ارتفاع ٢ است و دیگر ستون ها دارای ارتفاع ٠ می باشند.در شکل موس را بر روی ستون ١٣ برده ایم تا اطلاعات مربوط به آن در نوار زیر نمودار نشان داده شود.n نشان دهنده ی ارتفاع ستون است.

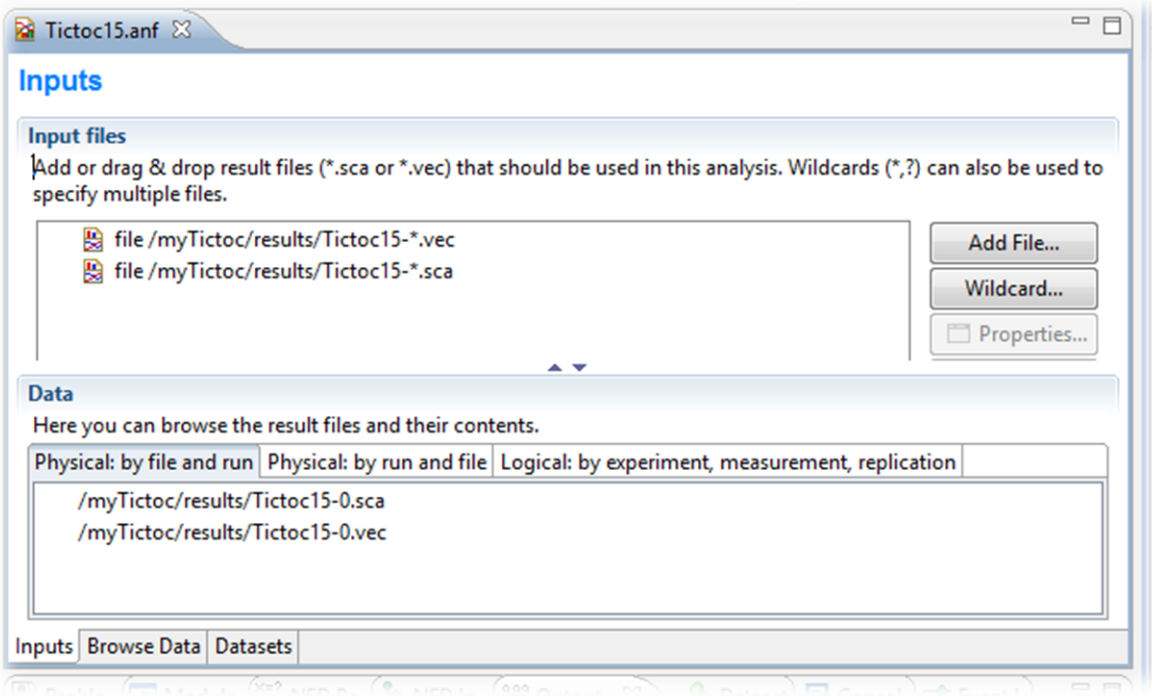
۴-۵- نمایش تصویری نتایج به وسیله ی OMNeT++ IDE

٤-٥-١- نمای ویژوال خروجی بردارها و اعداد اسکالر بدست آمده

OMNeT++ IDE میتواند با پشتیبانی از فیلتر، پردازش و نمایش داده های بردار و اسکالر و همچنین نمایش نمودارها، به شما در تحلیل نتایج خود کمک زیادی نماید. نمودارهای که در ادامه معرفی می شوند همگی با کمک ابزار Result Analysis در این IDE ایجاد شده اند.

برای استفاده از امکانات IDE باید پس از اتمام شبیه سازی به پوشه results رفته و بر روی فایل *.vec ویا *.sca کلیک کنید.به این نکته توجه کنید که در صورتی که این فایلها به صورتی متنی باز شدند باید بر روی فایل کلیک راست کرده و Open With و سپس New Analysis را انتخاب کنید.

در مثال Tictoc ما پس از کلیک بر روی یکی از این دو فایل نمای زیر را خواهیم دید:



شکل ٦٤

حال از تب های پایین صفحه بر روی تب Browse Data کلیک می کنیم:

Tictoc15.anf

Browse Data

Here you can see all data that come from the files specified in the Inputs page.

All (66 / 66) | Vectors (6 / 6) | Scalars (54 / 54) | Histograms (6 / 6)

runID filter | module filter

Name	Value
Tictoc15 : #0	
Tictoc15.tic[0]	
Tictoc15.tic[1]	
Tictoc15.tic[2]	
#received (scalar)	11.0
#sent (scalar)	11.0
hop count (histogram)	8.636363636363637 (11)
hop count:count (scalar)	11.0
hop count:max (scalar)	17.0
hop count:mean (scalar)	8.6363636363636
hop count:min (scalar)	1.0
hop count:sqrsum (scalar)	1119.0
hop count:stddev (scalar)	5.4639313186153
hop count:sum (scalar)	95.0
HopCount (vector)	8.636363636363637 (11)
Count	11
End event number	979
End time	97.8
Max	17.0
Mean	8.636363636363637
Min	1.0
Module name	Tictoc15.tic[2]
Start event number	145
Start time	14.4
StdDev	5.463931318615329
Type	double
Variance	29.854545454545452
Tictoc15.tic[3]	
Tictoc15.tic[4]	
Tictoc15.tic[5]	

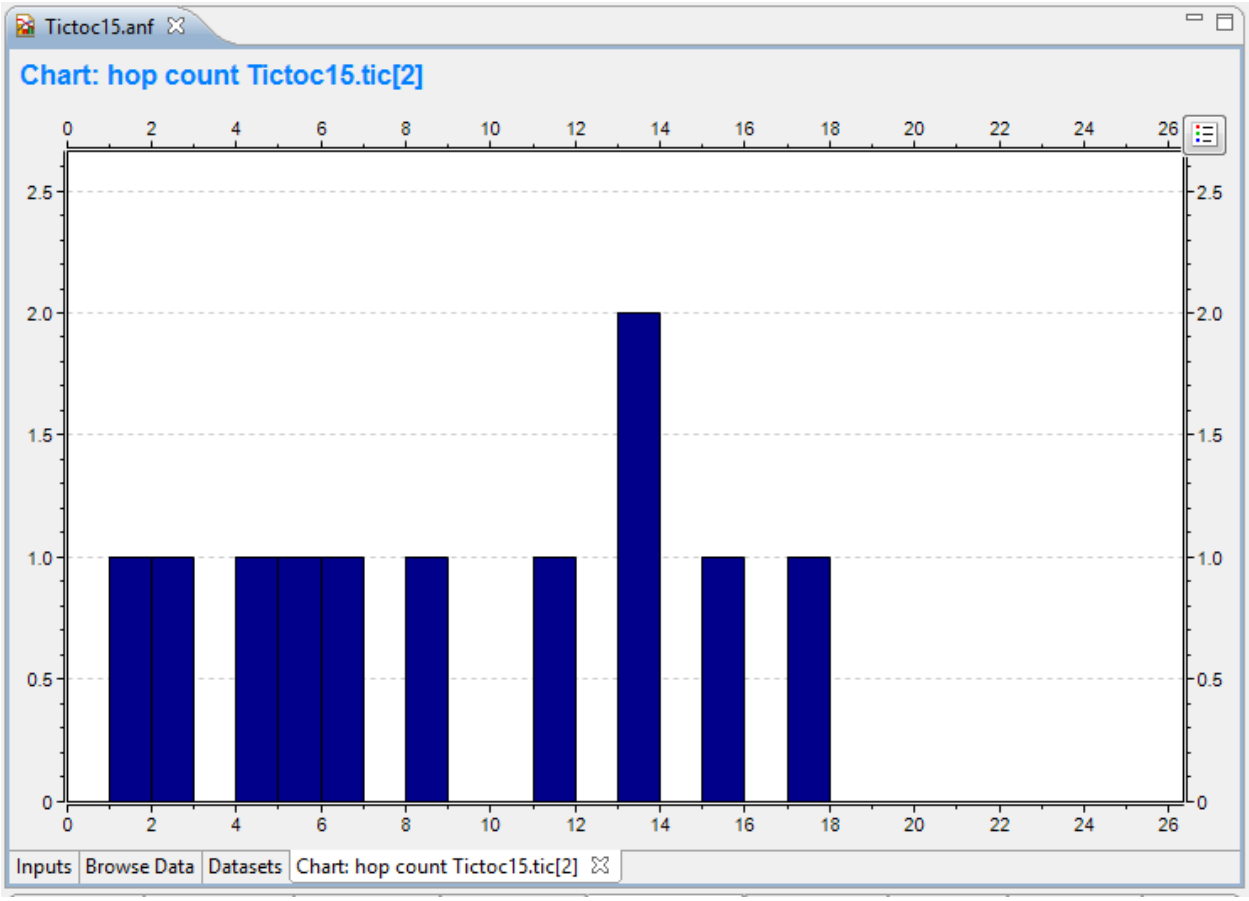
Inputs | Browse Data | Datasets

شکل ٦٥

این تب شامل چهار تب All، Vectors، Scalars و Histograms می شود.در تب اول یعنی All همانطور که در بالا مشاهده می کنید تمام داده های آماری اعم از وکتورها و اسکالرهای تمام ماژول های موجود در شبکه آورده شده است.می توانید این داده ها را که در تصویر بالا

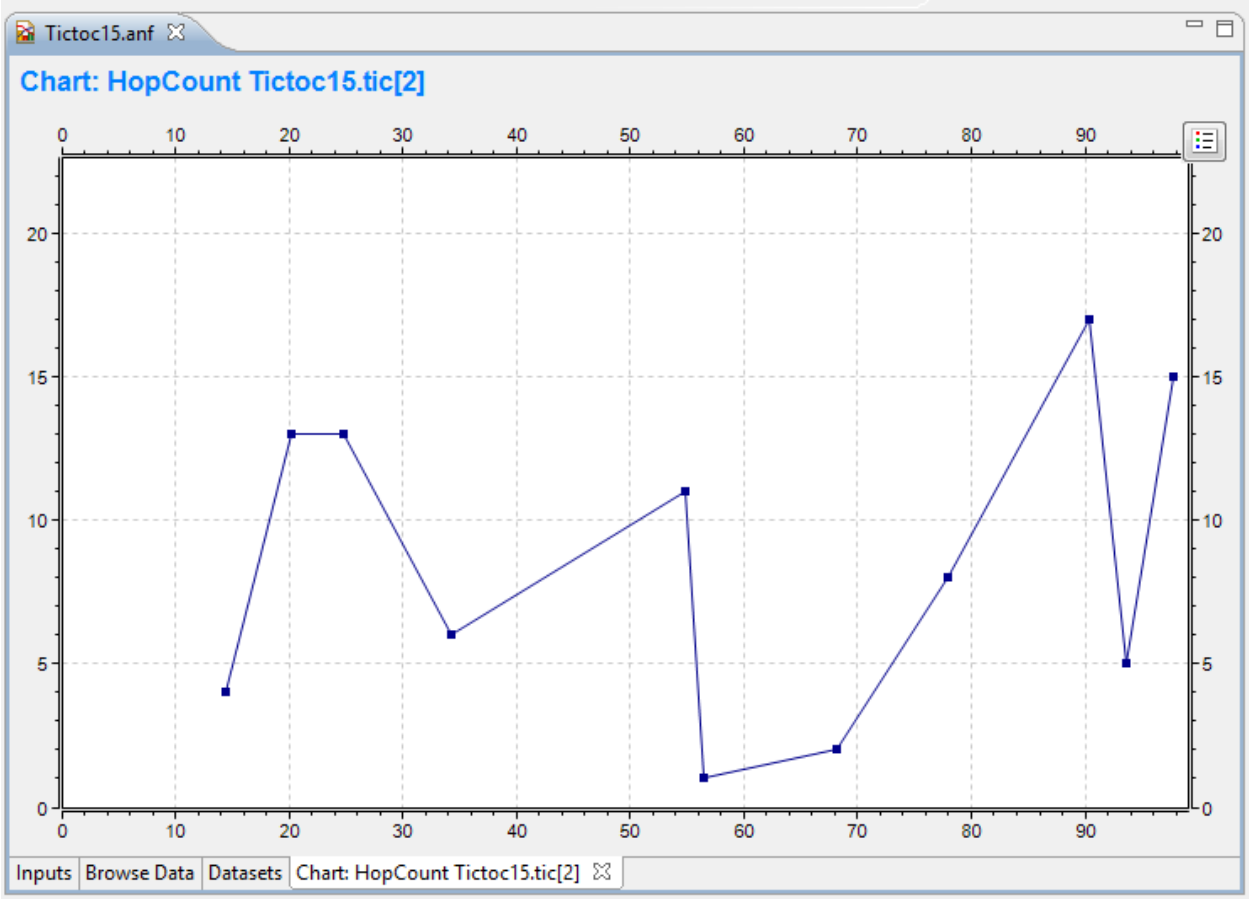
www.NoavaranGermi.ir

برای tic[۲] با داده هایی که در بخش قبل برای همین ماژول نشان داده شد،مقایسه کنید.در این صورت متوجه یکسان بودن این داده ها خواهید شد.حتی در اینجا اطلاعات بیشتری در رابطه با ماژول tic[۲] آورده شده است.این داده ها علاوه بر پیام های رسیده به ماژول می باشند برای مثال در قسمت HopCount (vecor) شماره آخرین رخداد اتفاق افتاده در این ماژول که ۹۷۹ است و شماره اولین رویداد رخ داده شده در آن که برابر ۱٤٥ است دیده می شود.اگر کمی در نمودار توالی جستجو کنید خواهید دید اولین رویدادی که در tic[۲] شروع می شود برای ارسال پیام tic-۲-to-۰ است که در لحظه ۱٤٥ اتفاق افتاده است و همانگونه که از نام آن پیداست پیام باید به tic[۰] ارسال می گشت.همچنین از دیگر اطلاعاتی که می توان در اینجا بدان دست یافت واریانس تعداد گام های طی شده توسط پیام ها برای رسیدن به tic[۲] است که این مقدار با توجه به تصویر برابر ۲۹,۸۵است.برای دیدن نمودار فراوانی تعداد گام های طی شده(گفته شده در قسمت قبل)می توان بر روی hop count(histogram) کلیک کنید.که در این زیر این نمودار برای tic[۲] نشان داده شده است:



شکل ٦٦

همچنین برای دیدن نمودار خطی مشابه آنچه در قسمت قبلی گفته شد می توان بر روی HopCount(vector) کلیک کرد.این نمودار برای tic[۲] بدین شکل می باشد:

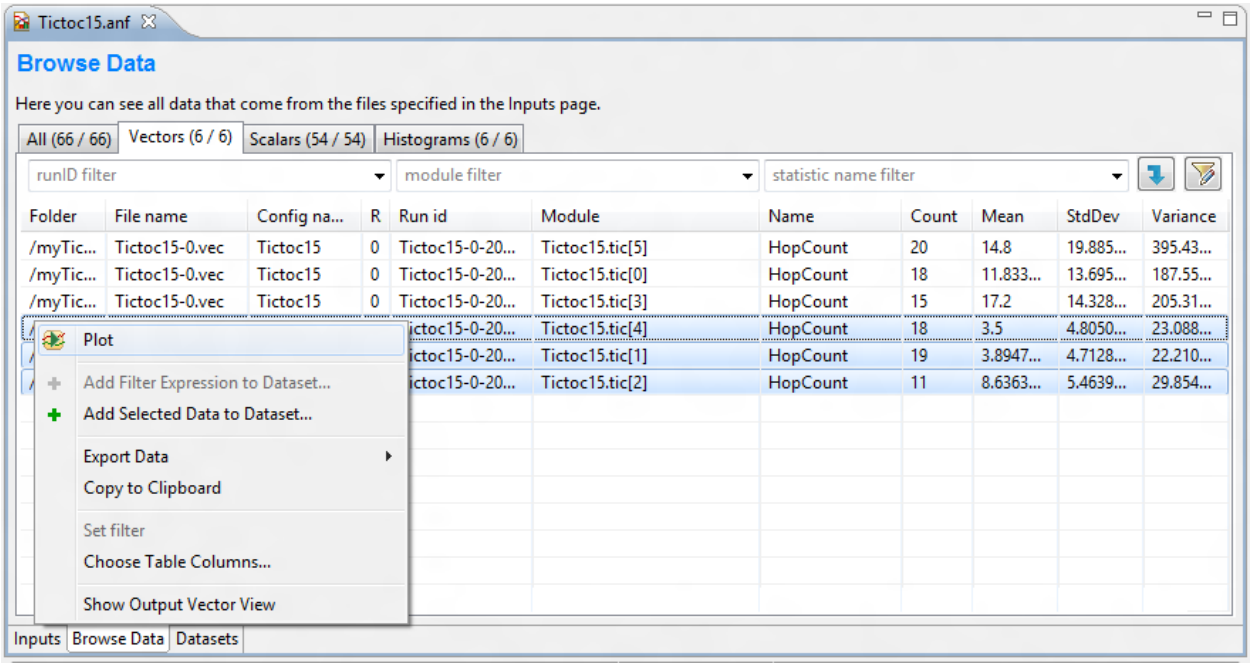


شکل ٦٧

می توانید این نمودار را با تصویر نمودار خطی که در بخش قبل نشان داده شد، مقایسه کنید.

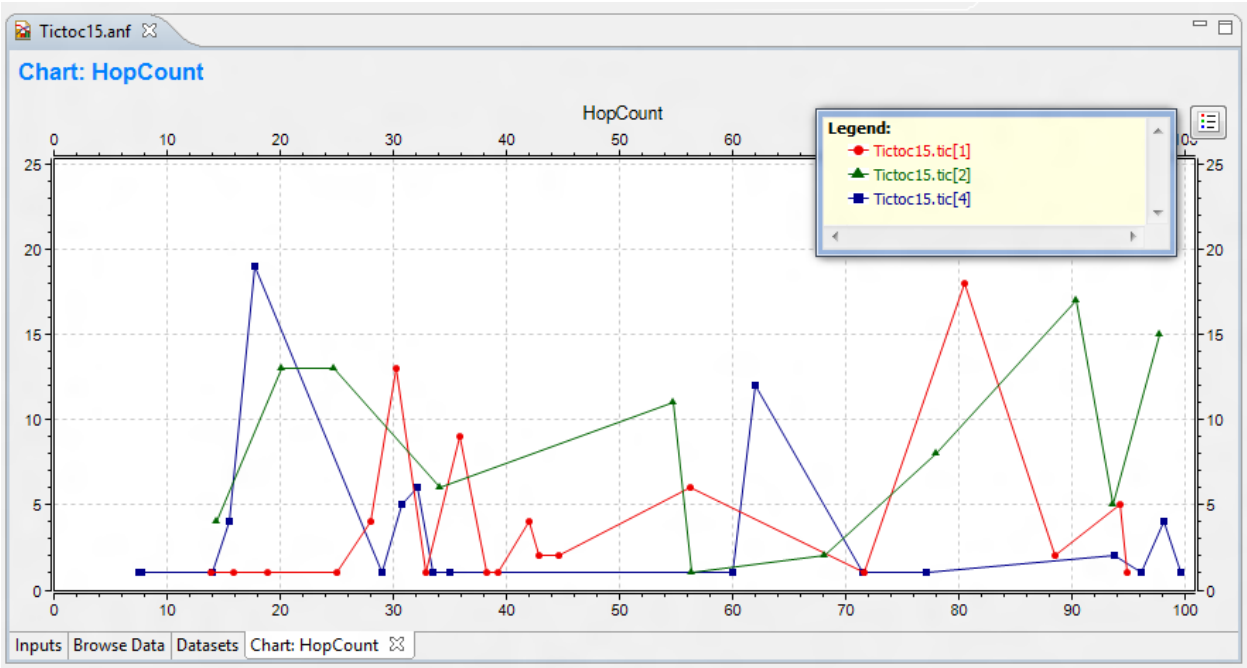
تا بدین جا امکاناتی که در IDE دیدیم قدرتی بیش از آنچه در قبل شاهد آن بودیم به ما نمیداد.در ادامه سعی می کنم ابزارها و امکانات بیشتری را معرفی کنیم.

برای مقایسه ی بردارهای خروجی چندین ماژول می توانید به تب Browse Data رفته و هر تعداد ماژول می خواهید انتخاب کنید و سپس بر روی یکی از آنها کلیک راست کرده و همانطور که در تصویر زیر نشان داده شده است از منوی باز شده گزینه ی Plot را انتخاب کنید تا نمودارهای هر یک رسم شود:



شکل ٦٨

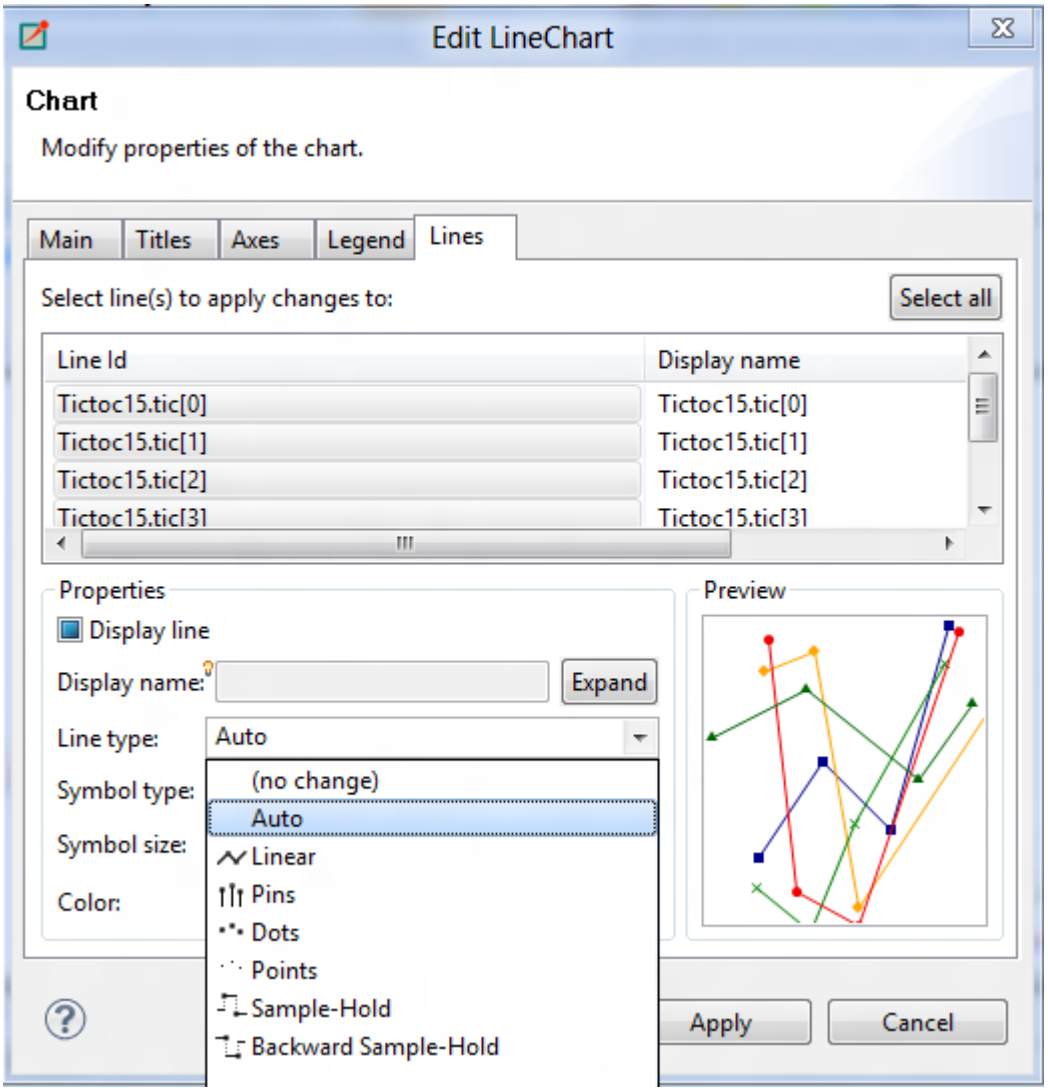
نمودار خطی ایجاد شده برای سه ماژول tic[٤] و tic[١] و tic[٢] به شکل زیر می باشد:



شکل ٦٩

با استفاده از اینگونه نمودارها می توان براحتی و در کمترین زمان مقایسه ای بین چندین ماژول انجام داد. برای مثال از این نمودار متوجه می شویم بیشترین گام های طی شده توسط پیام ها متعلق به پیامی است که به ماژول tic[٤] رسیده است وحدود ١٩ گام می باشد.

از دیگر قابلیت های این نمودار امکان تغییر نوع دید آن می باشد.برای این کار بر روی نمودار کلیک راست کرده و ...propreties را انتخاب کنید.از پنجره باز شده تب Lines را کلیک کنید.سپس در بخش propreties بر روی فلش کمبوباکس Line type کلیک کنید تا لیست نوع نمودارهای موجود باز شود:

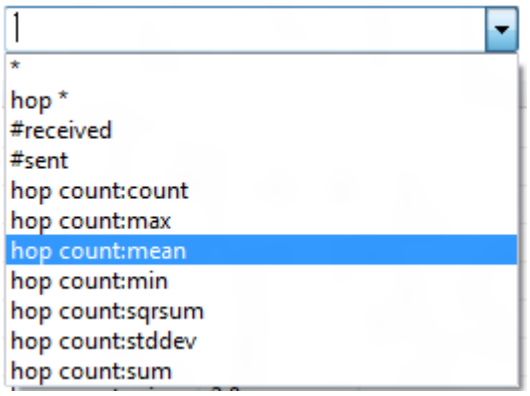


شکل ۷۰

داده های اسکالر از ماژول های مختلف را نیز می توان با یکدیگر مقایسه نمود.برای این کار باید ابتدا به تب Scalars برویم و سپس داده های دلخواه از ماژول های مورد نظر خود را انتخاب کنیم.سپس همانند مثال قبل بر روی آنها کلیک راست کرده و گزینه Plot را برگزینیم تا داده ها را بر روی نمودار ببینیم.یکی از امکانات مفیدی که OMNeT++ IDE در اختیار ما قرار

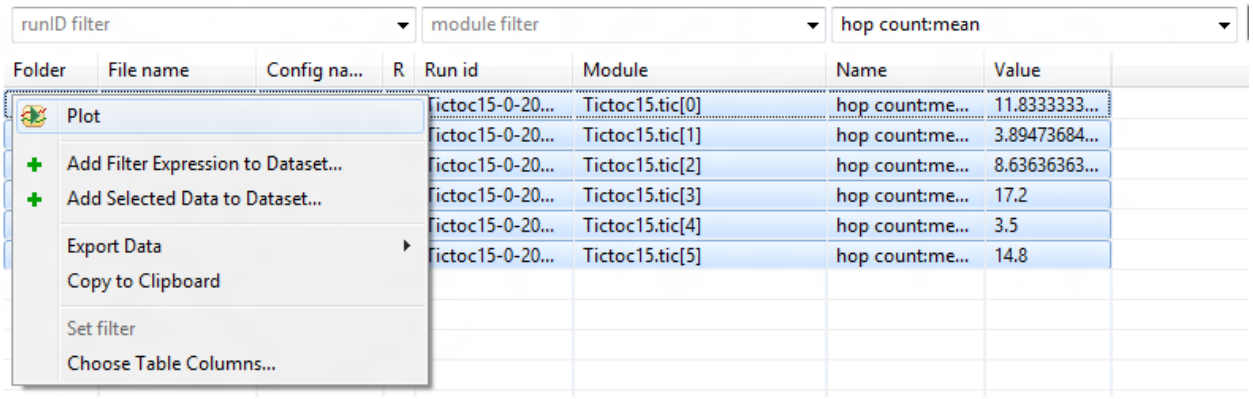
می دهد امکان Filter بر روی RunID(هر بار اجرای شبیه سازی یک آی دی منحصر به فرد دارد که با آن میتوان اجرای مورد نظر خود را مشخص کرد)، ماژول ها و داده های آماری است.برای مثال می خواهیم متوسط تعداد گامهای طی شده پیام ها رسیده به هریک از ماژول ها را با هم مقایسه کنیم.در اینجا یک راه غیر هوشمندانه پیدا کردن Mean مربوط به هریک از ماژولها و انتخاب آن است اما این کار را می توان با کمک فیلتر ها در OMNeT++ انجام داد.هر چند که داده های آماری و ماژولهای شبکه را در مثال Tictoc به راحتی می توان پیدا کرد اما موارد بسیاری وجود دارد که ماژولهای شبکه و داده های آماری ما بسیار زیاد می باشند که راهی جز استفاده از فیلتر باقی نمی ماند.

برای استفاده از فیلتر ابتدا بر روی کمبوباکس static name filter کلیک کرده و سپس hop count:mean را انتخاب نمایید:



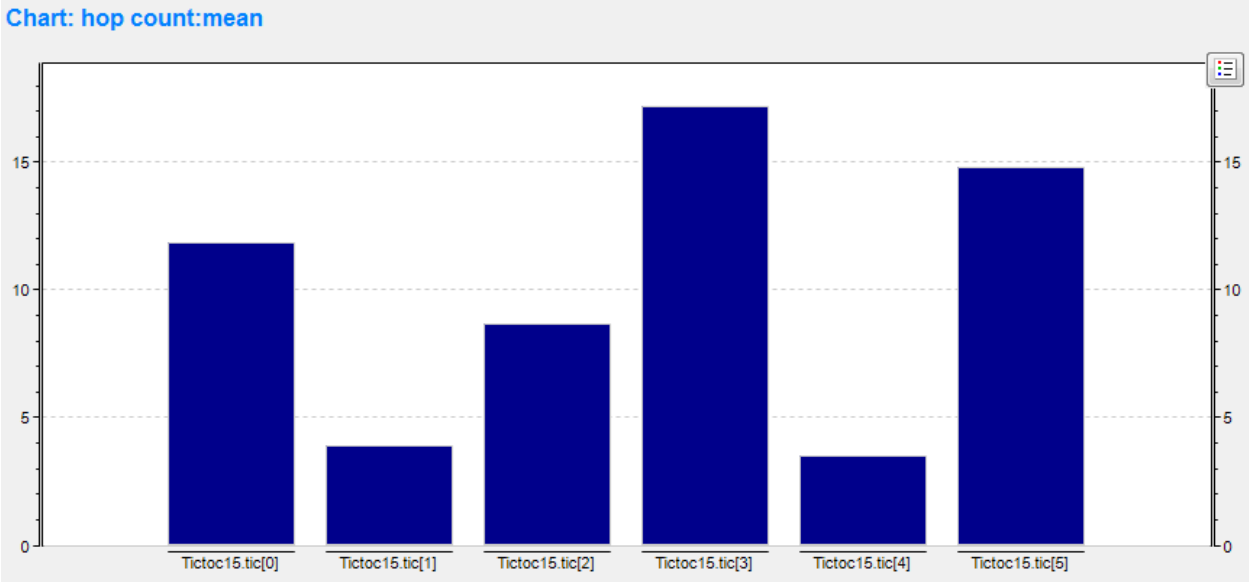
شکل ۷۱

سپس همه ماژول ها را انتخاب کرده و کلیک راست نمایید و گزینه ی Plot را برگزینید:



شکل ۷۲

چارت نمایش داده شده به شکل زیر خواهد بود:



شکل ۷۳

با استفاده از این نمودار می توانید به راحتی و سرعت دریابید که بیشترین متوسط تعداد گامهای طی شده توسط پیام ها ۱۷ گام میباشد که برای رسیدن به tic[۳] طی کرده اند .همچنین کمترین متوسط گام طی شده یعنی ۳ گام مربوط به پیام های رسیده به tic[٤] است.

www.NoavaranGermi.ir

تا به اینجا تا حدودی با نحوه ی کار شبیه ساز OMNeT++ آشنا شدیم و با آن یک شبکه ی فرضی را شبیه سازی کردیم.حال سعی می کنیم به بررسی یک نمونه ی پروتکل واقعی پرداخته وسپس شبیه سازی آن را در OMNet++ نمایش می دهیم.

۵- Directed diffusion در شبکه های حسگر بیسیم

۵-۱- مقدمه

مدل های حس[۱] سنتی یک یا چند سنسور قدرتمند و مرکزی برای محاسبات درنظر می گرفتند.امروزه،جهت گیریهای تکنولوژیکی ساخت دستگاه های هوشمند،کوچک و ارزان برای حس کردن را ممکن ساخته است.اگر تعداد زیادی سنسورهای کوچک بتوانند بایکدیگر به شکل یه شبکه ی حسگری کار کنند، چندین مزایا نسبت به حسگری متمرکز سنتی بدست خواهند آورد.با قرار دادن سنسور در نزدیک شئ ای که مورد حس قرار می گیرد، مشکلات پردازش سیگنال و تشخیص هدف در حسگری می تواند تا حد زیادی مرتفع گردد.در ارتباطات به دلیل آنکه پیام به جای طی کردن یک مسیر طولانی تنها چندین مسیر کوتاه را طی می کند، انرژی مصرفی می تواند کاهش یابد.همچنین به دلیل پردازش داده در شبکه اغلب میزان داده ی انتقال داده شده می تواند کاهش یابد که به صرفه جویی بیشتر انرژی می انجامد.

با انگیزه نیاز به صرفه جویی انرژی و قابلیت گسترش این گونه شبکه های سنسوری،یک نمونه ی جدید انتشار داده به نام directed diffusion که data-centric می باشد آزمایش گردید.داده های تولید شده به وسیله ی گره ها (سنسورها) با جفت صفت-مقدار نامگذاری شده اند.یک نود داده ای را از طریق ارسال علاقه مندیها[۲] برای داده ای مشخص، تقاضا می کند.سپس داده ای که با آن علاقه مندی مطابقت داشته باشد به سمت گره تقاضا دهنده پس فرستاده می شود.گره های میانی می توانند داده را کش کرده یا آن را تغییر شکل دهند و یا ممکن است علاقه مندیها را بر اساس داده های کش شده قبلی به مسیری هدایت کنند.

---

[۱] sense

[۲] interests

www.NoavaranGermi.ir

Directed diffusion بسیار متفاوت با ارتباط بر مبنای IP است. در این پروتکل گره ها در شبکه از برنامه ها آگاهی دارند و به کد برنامه اجازه ی اجرا در شبکه را می دهند و با کش کردن و پردازش پیام ها به انتشار کمک می کنند که با این کار میزان ترافیک کاهش پیدا کرده و نتیجه ی آن میزان بسیار زیادی صرفه جویی در مصرف انرژی است.

۵-۲- خانواده پروتکل directed diffusion

خانواده پروتکل directed diffusion دارای سه الگوریتم به نام های one-phase push، one-phase pull و two-phase pull است که ما در اینجا به تشریح two-phase pull خواهیم پرداخت.

الگوریتم two-phase directed diffusion از چندین پیام کنترلی استفاده می کند.چاهک[۱] ها برای پیدا کردن منابع[۲] پیام های interest را ارسال می کنند، منابع از پیام های داده های اکتشافی[۳] برای یافتن منابع استفاده می کنند و پیام های تقویتی[۴] مثبت و منفی بخش هایی از مسیر را حذف و یا انتخاب می کنند.

۵-۳- Two-Phase Pull Diffusion

هدف directed diffusion برقراری ارتباطی بهینه با n مسیر بین یک یا چند منبع و چاهک است.Directed diffusion یک نمونه ارتباط data-centric می باشد که به کلی با ارتباطات برمبنای host در شبکه های سنتی متفاوت است.برای شرح المان های انتشار، مثالی ساده از یک شبکه سنسوری طراحی شده برای رد گیری حیوانات در حیات وحش را بیان می کنیم.

فرض کنید کاربری در شبکه می خواهد حرکات حیوانات را در بخش دوری از یک ناحیه از پارک زیر نظر بگیرد.کاربر با تعیین یک سری ویژگی ها، مشترک اطلاعات "ردیابی حیوان" می شود.حسگرهای سرتاسر شبکه اطلاعات ردیابی حیوان را تولید می کنند.

---

[۱] sinks

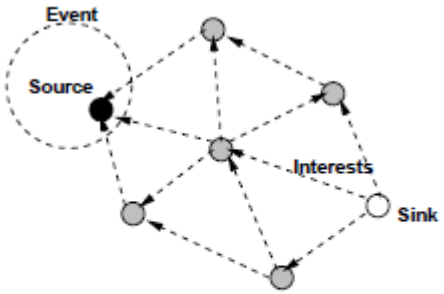
[۲] sources

[۳] exploratory data

[۴] reinforcement

www.NoavaranGermi.ir

برنامه کاربر به کمک لیستی از جفت "صفت-مقدار" که وظیفه ای را شرح می دهد، مشترک[1] داده ها می شود.صفت ها، داده های مطلوب را با تعیین نوع های حسگر و منطقه جغرافیایی تعیین می کند.گره کاربر چاهک است و صفت های مورد علاقه[2] خود را که تعیین کننده ی نوع خاصی از داده ها است تعیین می کند. علاقه، همسایه به همسایه به سوی حسگرهای تعیین شده در ناحیه ای خاص حرکت می کنند.ویژگی کلیدی directed diffusion آن است که هر گره حسگر می تواند آگاه از وظیفه[3] باشد، بدین گونه که گره ها به جای اینکه تنها منتقل کننده محض علاقه مندی ها باشند آنها را ذخیره و تفسیر می کنند.در مثال گفته شده هر حسگر که علاقه مندی دریافت می کند، همسایه ای که آن پیام را ارسال کرده به خاطر می سپارد.برای هر همسایه شیبی[4] تنظیم می کند.یک شیب نشان دهنده ی مسیری است به سمت داده هایی که با علاقه مند مطابقت دارند.بعد از تنظیم یک شیب، حسگر مجددا علاقه مندی را به همسایگان خود توزیع می کند.زمانی که نود پی ببرد که منبع احتمالی کجاست(براساس شیب یا اطلاعات جغرافیایی)، علاقه مندی می تواند به زیر مجموعه ای از همسایگان ارسال گردد.در غیر این صورت به صورت ساده به تمام همسایه ها ارسال می گردد.

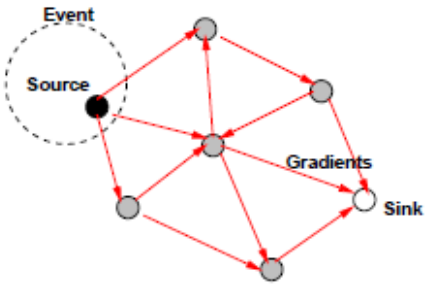


(a) Interest propagation

شکل ۷٤

[1] subscribe
[2] interest
[3] task-aware
[4] gradient

www.NoavaranGermi.ir



(b) Initial gradients set up

شکل ۷۵

حسگرها نیز با انتشار یک سری از صفات، تعیین می کنند چه داده هایی تولید می نمایند.بنابراین آنها تبدیل به منابع بلقوه می گردند.به محض حرکت علاقه مندی ها در شبکه، حسگرهایی با داده های تطابق یافته با آنها، رها شده[۱] و برنامه حسگرهای محلی خود را فعال کرده تا شروع به جمع آوری داده ها کنند(قبل از فعال سازی این طور در نظر می گیریم که گره ها در حالت کم مصرف قرار داشته اند).سپس حسگر پیام های "داده" را که با علاقه مندی ها مطابقت دارد را تولید می کند.

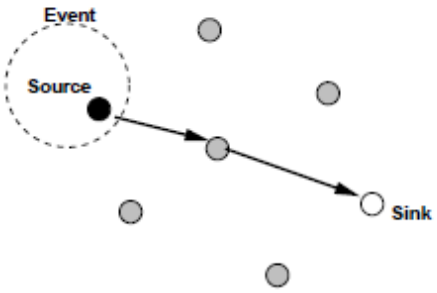
برای اهداف گوناگون و در مراحل مختلف انتشار داده ها در گره های میانی کش می شوند.هسته ی اصلی مکانیزم انتشار از کش برای اجتناب از حلقه ها و حذف پیام های تکراری استفاده می کن وهمچنین می تواند از آن برای ارسال علاقه مندی ها براساس ترجیح استفاده کند.همچنین از داده های کش برای انجام پردازش های برنامه در شبکه استفاده می گردد.مثلا داده های دریافت شده از چندین حسگر مختلف که مربوط به یک شئ می باشند در جوابی واحد ادغام شود.

پیام داده ی ابتدایی از منبع به عنوان داده اکتشافی علامت گذاری می شود و به تمام همسایه هایی که دارای شیب مورد نظر باشند ارسال می گردد.اولین علاقه مندی سیل آسا به همراه داده های اکتشافی، اولین فاز از two-phase pull diffusion را تشکیل می دهند.در صورتی که

[۱] trigger

www.NoavaranGermi.ir

چاهک دارای چندین همسایه باشد، انتخاب می کند که زیرمجموعه ای از داده ها برای یک علاقه مندی یکسان از یک همسایه ترجیح داده شده را دریافت نماید (برای مثال، همسایه ای که اولین کپی از داده را تحویل داده است).به همین جهت،چاهک همسایه ی ترجیح داده شده را "تقویت"[1] می کند که آن نیز همسایه بعدی خود را تقویت می کند و به همین صورت ادامه پیدا می کند.همچنین ممکن است چاهک همسایه ترجیحی خود را در صورتی که همسایه دیگری با تاخیر کمتری داده را تحویل دهد، به صورت منفی تقویت[2] کند.این تقویت منفی همسایه به همسایه انتشار پیدا کرده و شیب ها را ازبین می برد و مسیر موجود را در صورتی که دیگر نیازی به آن نباشد از بین می برد.



(c) Data delivery along re-inforced path

شکل ۷٦

پس از پیام های داده ی اکتشافی، پیام های بعدی تنها بر روی مسیرهای تقویت شده ارسال خواهند شد.تقویت مسیر و انتقالات بعدی بر روی مسیرهای تقویت شده، فاز دوم two-phase pull diffusion را تشکیل می دهند.به صورت دوره ای منبع پیام های اکتشافی دیگری ارسال کرده تا به دلیل تغییرات شبکه(به دلیل از کار افتادن یک گره یا حرکت آن و...) شیب ها را تنظیم نماید.

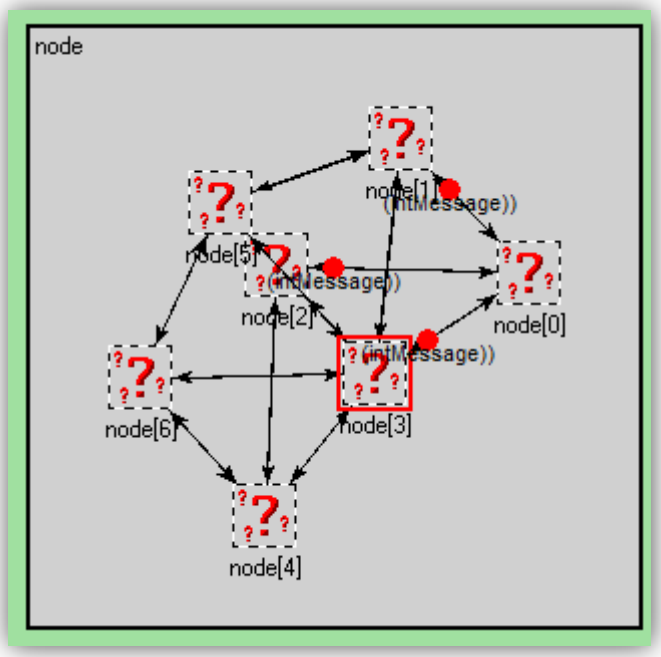
[1] reinforces

[2] negatively reinforce

www.NoavaranGermi.ir

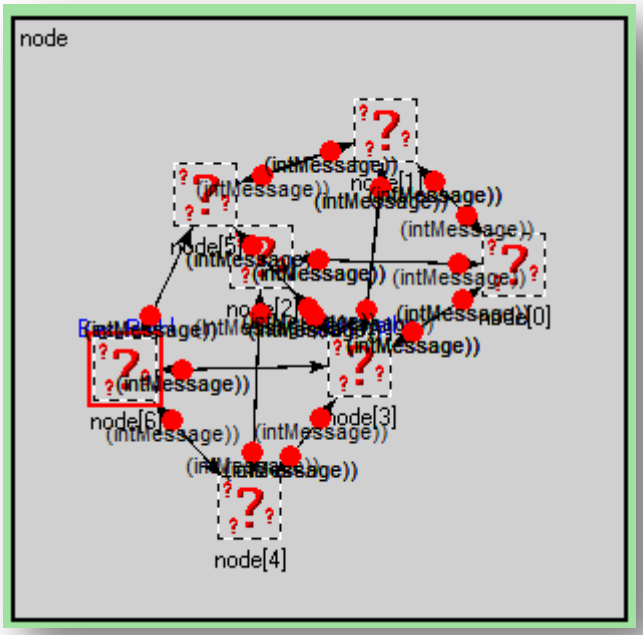
۵-۴- شبیه سازی directed diffusion در OMNeT++

در این شبیه سازی مدل ما نیز دارای هفت حسگر می باشد و نود ۰ چاهک درنظر گرفته شده است و سعی دارد از طریق ارسال علاقه مندی های خود به تمام همسایه ها، داده های مورد نظر خود را دریافت نماید.همسایه ها نیز علاقه مندی ها را دریافت و آنها هم به همین صورت به تمام همسایه های خود ارسال می کنند.سپس شیب تنظیم می شود.



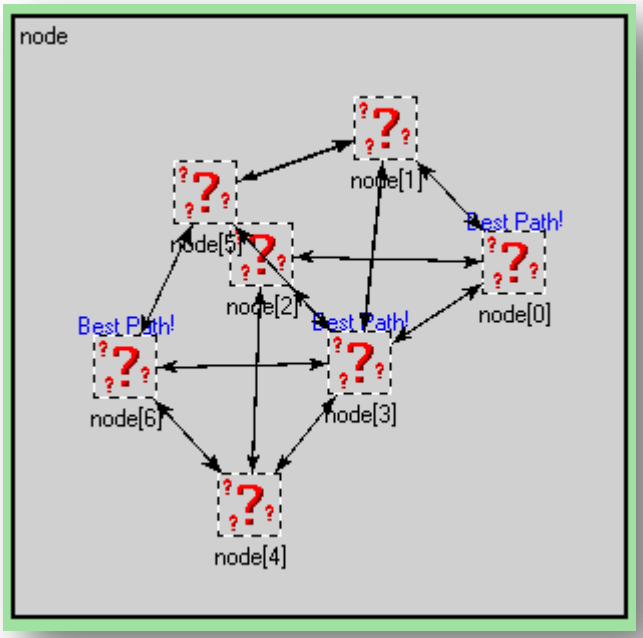
شکل ۷۷

در اینجا منبع گره ٦ است و با مطابقت دادن علاقه مندی ها با داده های خود شروع به ارسال داده های اکتشافی می کند.



شکل ۷۸

مسیری کم کم تقویت شده و به عنوان Best Path تعیین می گردد.پس از آن داده ها تنها از نود ٦ به نود ۰ انتقال می یابند.



شکل ۷۹

www.NoavaranGermi.ir

٦- منابع و ماخذ:

www.omnetpp.org

www.omnest.com

www.forum.wsnlab.ir

Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks.In Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom'٢٠٠٠), Boston, Massachusetts, August ٢٠٠٠.