



دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده مهندسی برق و کامپیوتر

گروه مهندسی برق - مخابرات سیستم

درس شبکه‌های مخابرات نوری

آموزش کار با نرم افزار omnet++

توسط:

میثم یاری بیگی

دی ماه ۱۳۹۵

فهرست مطالب

صفحه	عنوان
۳	فهرست شکل‌ها
۶	فصل ۱ - نصب نرم افزار
۶	۱-۱ - مقدمه
۶	۲-۱ - مراحل نصب omnet++ در Windows 7
۱۵	فصل ۲ - آشنایی با محیط نرم افزار
۱۵	۱-۲ - مقدمه
۱۶	۲-۲ - بررسی مثال عملی شبیه سازی TicToc
۱۶	۳-۲ - شروع به کار
۱۶	۱-۳-۲ - قدم ۱: پیاده سازی اولیه
۱۹	۴-۲ - ارتقا و بهبود مدل دو گره ای TicToc
۱۹	۱-۴-۲ - قدم ۲: بهبود گرافیک و افزودن خروجی debug
۲۱	۵-۲ - قدم ۳: افزودن متغیرهای وضعیت
۲۲	۶-۲ - قدم ۴: افزودن پارامترها
۲۴	۷-۲ - قدم ۵: استفاده از ارث بری
۲۵	۸-۲ - قدم ۶: مدل سازی تاخیر پردازش
۲۶	۹-۲ - قدم ۷: شماره تصادفی و پارامترها
۲۷	۱۰-۲ - قدم ۸: لغو تایمرها و اتمام زمان
۲۹	۱۱-۲ - قدم ۹: ارسال دوباره یک پیغام
۲۹	۱۲-۲ - تبدیل به یک شبکه واقعی
۲۹	۱-۱۲-۲ - قدم ۱۰: بیش از دو گره
۳۱	۱۳-۲ - قدم ۱۱: کانالها و تعریف آنها
۳۲	۱۴-۲ - قدم ۱۲: استفاده از اتصالات دو طرفه
۳۳	۱۵-۲ - قدم ۱۳: تعریف کلاس پیغام
۳۶	۱۶-۲ - افزودن مجموعه های آماری
۳۶	۱-۱۶-۲ - قدم ۱۴: نمایش شماره بسته های ارسال شده و دریافت شده

۳۸	۱۷-۲ - قدم ۱۵: افزودن مجموعه های آماری
۳۸Event log - ۱-۱۷-۲
۴۰	۲-۱۷-۲ - راه هایی برای کوچک کردن حجم لگ و بررسی بخشی از شبیه سازی
۴۱Output Vector - Output Scalars - ۳-۱۷-۲
۴۲	۴-۱۷-۲ - استفاده از بردارهای خروجی و اسکالارهای خروجی در مثال TicToc
۴۵	۵-۱۷-۲ - توضیح تابع () finish
۴۶	۶-۱۷-۲ - نمایش داده در زمان اجرای شبیه سازی
۴۹	۲-۱۸-۲ - نمایش ویژوال خروجی بردارها و اعداد اسکالار بدست آمده
۵۷	فصل ۳ - ایجاد پروژه و ذخیره آن
۵۷	۱-۳ - مقدمه
۵۷	۲-۳ - نحوه ایجاد یک پروژه
۶۲	۳-۳ - نحوه Export کردن یک پروژه

فهرست شکل‌ها

صفحه	عنوان
۶	شکل ۱-۱ استخراج فایل Zip
۷	شکل ۲-۱ اجرای محیط کنسول
۸	شکل ۳-۱ محیط کنسول در اولین اجرا
۸	شکل ۴-۱ اجرای دستور ./configure
۸	شکل ۵-۱ make کردن omnet++
۹	شکل ۶-۱ اجرا کردن omnet++ با دستور omnetpp
۱۰	شکل ۷-۱ تنظیمات آدرس Workspace
۱۰	شکل ۸-۱ تنظیمات اولیه نرم افزار
۱۱	شکل ۹-۱ اضافه کردن پروژه های آماده
۱۲	شکل ۱۰-۱ نحوه اجرا کردن یک پروژه(الف)
۱۳	شکل ۱۱-۱ نحوه اجرا کردن یک پروژه(ب)
۱۴	شکل ۱۲-۱ نحوه اجرا کردن یک پروژه(ج)
۱۴	شکل ۱۳-۱ نحوه اجرا کردن یک پروژه(د)
۱۶	شکل ۱-۲ پیاده سازی اولیه شبکه tictoc1 (الف)
۱۷	شکل ۲-۲ پیاده سازی اولیه شبکه tictoc1 (ب)
۱۸	شکل ۳-۲ محیط اجرای شبیه سازی برای tictoc1
۱۹	شکل ۴-۲ تعریف شبکه tictoc2
۲۰	شکل ۵-۲ شبکه tictoc2 در محیط اجرای شبیه سازی
۲۰	شکل ۶-۲ تغییرات شبکه tictoc2 در محیط اجرای شبیه سازی (الف)
۲۱	شکل ۷-۲ تغییرات شبکه tictoc2 در محیط اجرای شبیه سازی (ب)
۲۲	شکل ۸-۲ مشاهده مقدار یک متغیر در شبکه tictoc3 در محیط اجرای شبیه سازی
۲۶	شکل ۹-۲ مدلسازی تاخیر پردازش در شبکه tictoc6
۲۸	شکل ۱۰-۲ نمایش از بین رفتن پکت ها در شبکه tictoc8 در محیط اجرای شبیه سازی
۳۰	شکل ۱۱-۲ تصویری از شبکه tictoc10 در محیط شبیه اجرای شبیه سازی
۳۵	شکل ۱۲-۲ تصویری از شبکه tictoc14 در محیط اجرای شبیه سازی (الف)
۳۵	شکل ۱۳-۲ تصویری از شبکه tictoc14 در محیط اجرای شبیه سازی (ب)

..... ۳۶	شکل ۱۴-۲ نمایش بسته های دریافت شده و ارسال شده
..... ۳۷	شکل ۱۵-۲ تغییر محیط گرافیکی در حین اجرای شبیه سازی
..... ۳۹	شکل ۱۶-۲ Record کردن در محیط اجرای شبیه سازی
..... ۳۹	شکل ۱۷-۲ اجرای فایل Tictoc15.eLog برای
..... ۴۰ شکل ۱۸-۲ خروجی خاصل از اجرای فایل Tictoc15.eLog برای
..... ۴۴ شکل ۱۹-۲ فراخوانی تابع finish()
..... ۴۴ شکل ۲۰-۲ اطلاعات تابع finish (الف)
..... ۴۶ شکل ۲۱-۲ اطلاعات تابع finish (ب)
..... ۴۶ شکل ۲۲-۲ نمایش داده در زمان اجرای شبیه سازی
..... ۴۷ شکل ۲۳-۲ نمودار اطلاعات ذخیره شده در cOutVector
..... ۴۹ شکل ۲۴-۲ نمودار اطلاعات ذخیره شده در cLongHistogram
..... ۵۰ شکل ۲۵-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (الف)
..... ۵۰ شکل ۲۶-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (ب)
..... ۵۱ شکل ۲۷-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (ج)
..... ۵۲ شکل ۲۸-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (د)
..... ۵۳ شکل ۲۹-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (ه)
..... ۵۳ شکل ۳۰-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (و)
..... ۵۴ شکل ۳۱-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (ز)
..... ۵۵ شکل ۳۲-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (ر)
..... ۵۵ شکل ۳۳-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (ح)
..... ۵۶ شکل ۳۴-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (ط)
..... ۵۷ شکل ۱-۳ نحوه ایجاد یک پروژه (الف)
..... ۵۸ شکل ۲-۳ نحوه ایجاد یک پروژه (ب)
..... ۵۸ شکل ۳-۳ نحوه ایجاد یک پروژه (ج)
..... ۵۹ شکل ۴-۳ نحوه ایجاد یک پروژه (د)
..... ۶۰ شکل ۵-۳ نحوه ایجاد یک پروژه (ه)
..... ۶۱ شکل ۶-۳ نحوه Build کردن یک پروژه
..... ۶۲ شکل ۷-۳ نحوه Export کردن یک پروژه (الف)
..... ۶۳ شکل ۸-۳ نحوه Export کردن یک پروژه (ب)



فصل ۱ - نصب نرم افزار

۱-۱ مقدمه

نرم افزار omnet++ قابلیت نصب بر روی سیستم عامل های زیر را دارد:

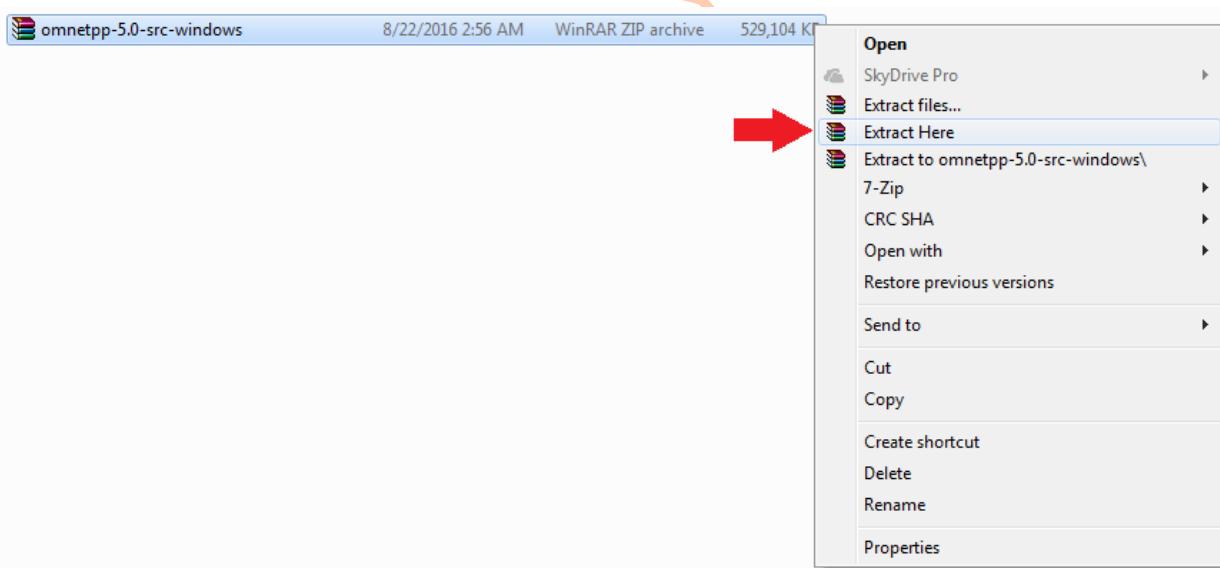
- Windows 7, 8 and XP
- Mac OS X 10.7, 10.8 and 10.9
- Linux x86 32/64-bit

۲-۱ مراحل نصب omnet++ در Windows 7

۱- فایل Zip را از مسیر زیر دانلود کنید.

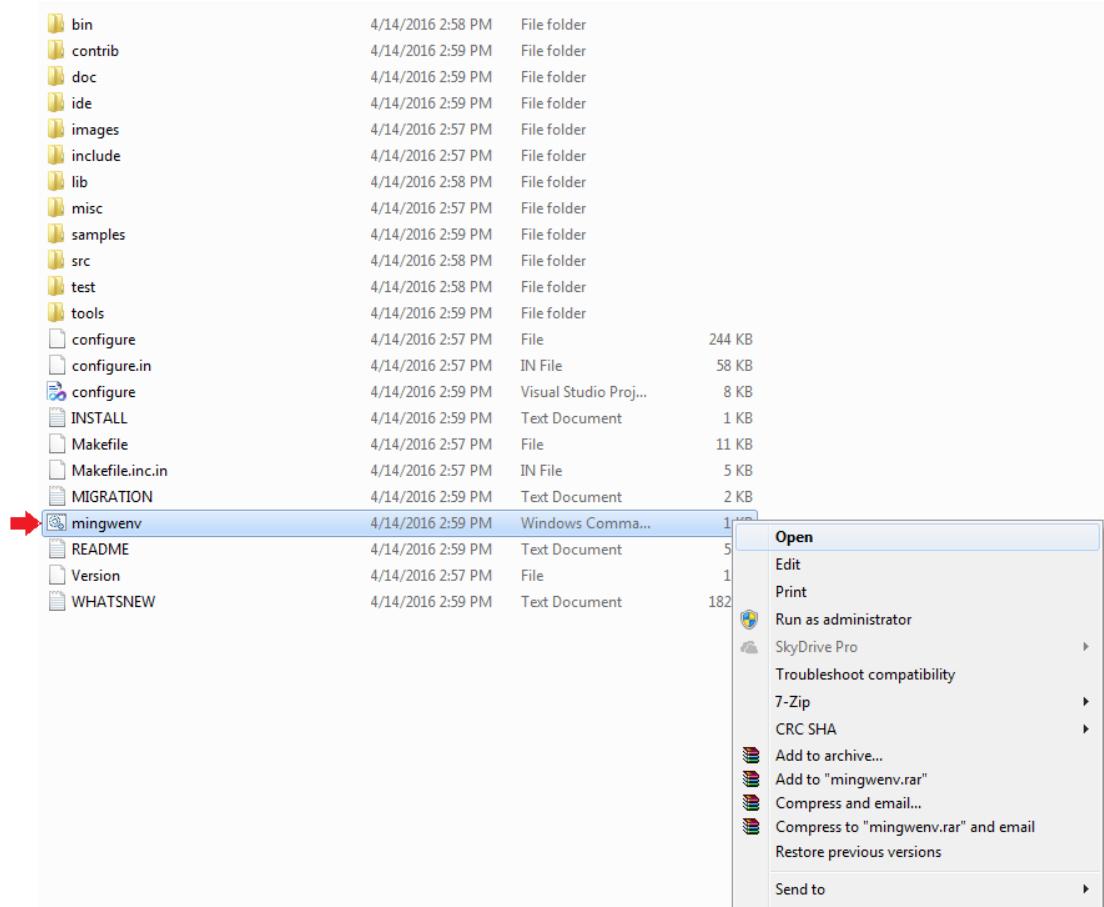
<https://omnetpp.org/omnetpp/send/30-omnet-releases/2307-omnetpp-50-windows>

۲- فایل را در یکی از درایوها ذخیره کرده و در همانجا از حالت فشرده خارج کنید.



شکل ۱-۱ استخراج فایل Zip

۳- در پوشه ایجاد شده فایل mingwenv را اجرا کنید.



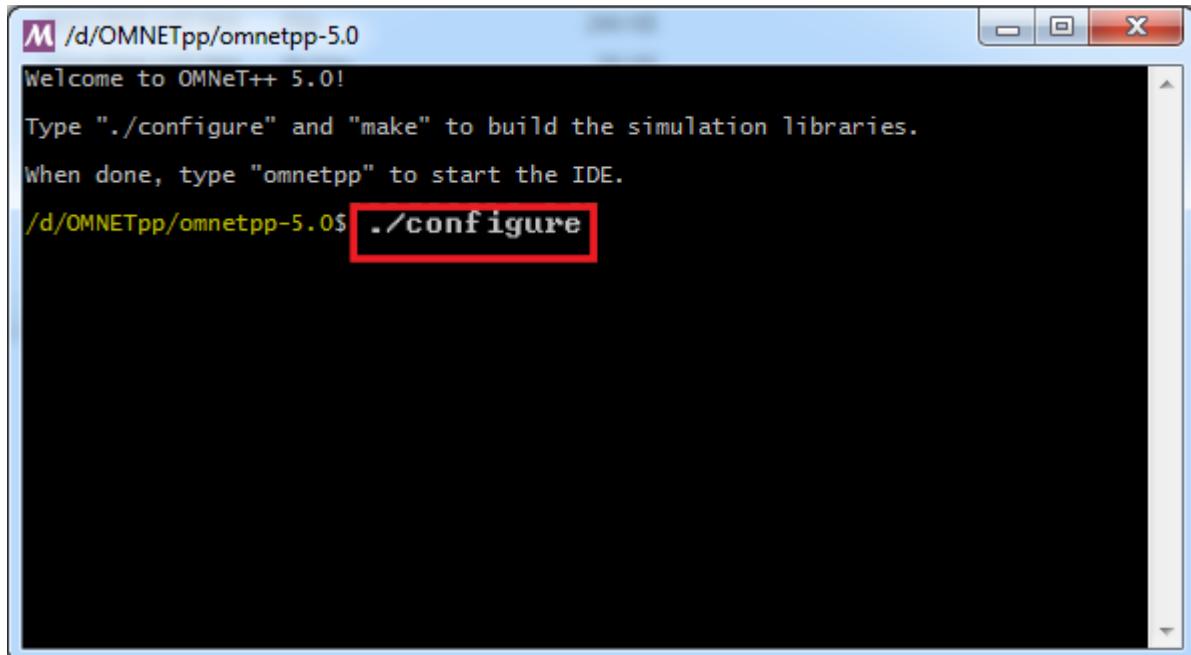
شکل ۲-۱ اجرای محیط کنسول

۴- یک صفحه کنسول به شکل زیر باز می‌شود که تنظیمات اضافه کردن path برای استفاده از ابزارهای مختلف را نیز انجام می‌دهد. برای ادامه روند نصب یکی از دکمه‌های صفحه کلید را فشار دهید.

```
C:\Windows\system32\cmd.exe
*** Welcome to OMNeT++! ***
We need to unpack the MinGW toolchain before continuing.
This can take a while, please be patient.
Press any key to continue . . .
```

شکل ۱-۳ محیط کنسول در اولین اجرا

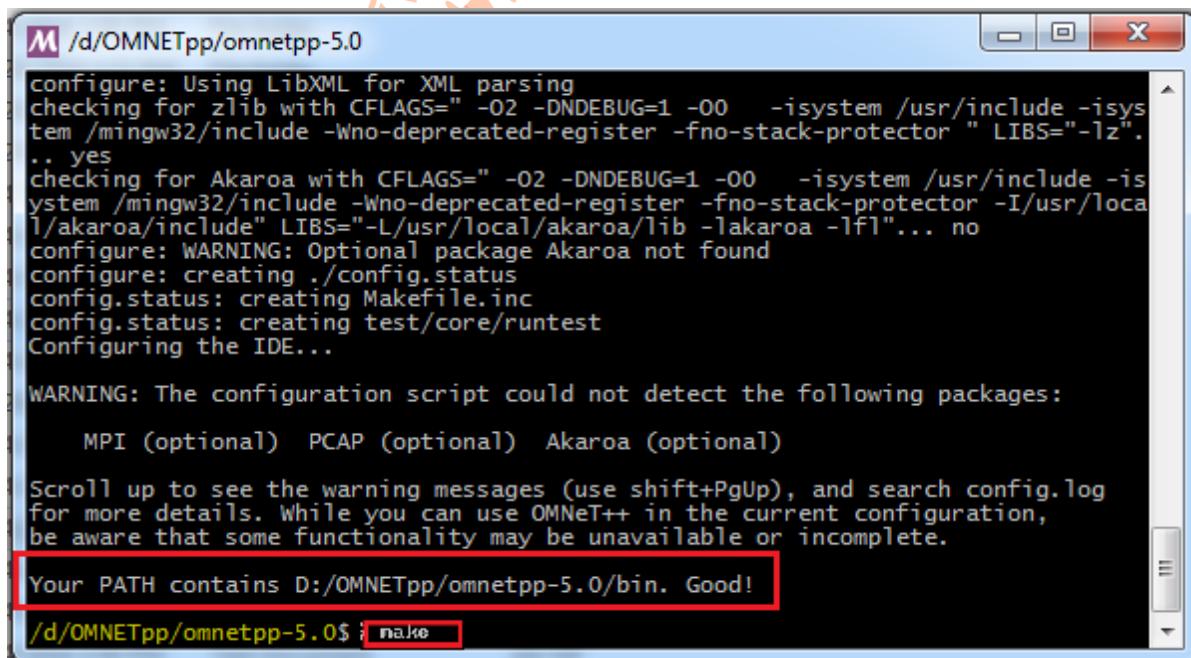
۵- با باز شدن صفحه زیر، دستور `./configure` را تایپ کرده و `enter` بزنید.



```
Welcome to OMNeT++ 5.0!
Type "./configure" and "make" to build the simulation libraries.
When done, type "omnetpp" to start the IDE.
/d/OMNETpp/omnetpp-5.0$ ./configure
```

شکل ۱-۴ اجرای دستور `./configure`

۶- بعد از اتمام دستورات بالا خط فرمان کنسول آزاد می شود و باید خروجی شبیه ساز به شکل زیر باشد. این بار دستور `make` را تایپ کرده و `enter` بزنید.



```
configure: Using LibXML for XML parsing
checking for zlib with CFLAGS="-O2 -DNDEBUG=1 -O0 -isystem /usr/include -isystem /mingw32/include -Wno-deprecated-register -fno-stack-protector" LIBS="-lz".
.. yes
checking for Akaroa with CFLAGS="-O2 -DNDEBUG=1 -O0 -isystem /usr/include -isystem /mingw32/include -Wno-deprecated-register -fno-stack-protector -I/usr/local/akaroa/include" LIBS="-L/usr/local/akaroa/lib -lakaroa -lf1"... no
configure: WARNING: Optional package Akaroa not found
configure: creating ./config.status
config.status: creating Makefile.inc
config.status: creating test/core/runtest
Configuring the IDE...

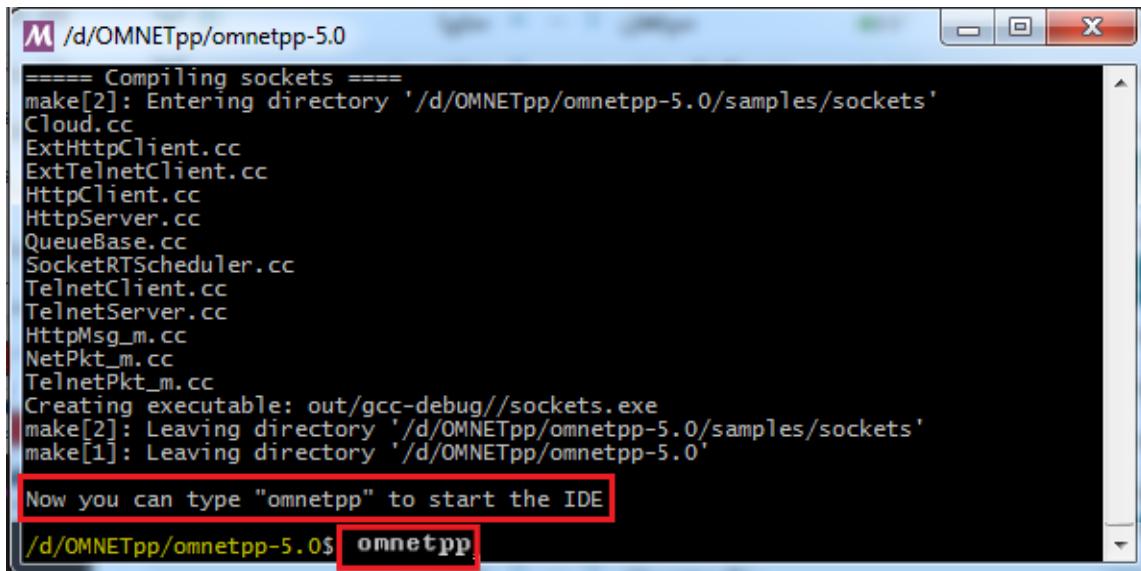
WARNING: The configuration script could not detect the following packages:
          MPI (optional)  PCAP (optional)  Akaroa (optional)

Scroll up to see the warning messages (use shift+PgUp), and search config.log
for more details. While you can use OMNeT++ in the current configuration,
be aware that some functionality may be unavailable or incomplete.

Your PATH contains D:/OMNETpp/omnetpp-5.0/bin. Good!
/d/OMNETpp/omnetpp-5.0$ make
```

شکل ۱-۵ omnet++ make کردن

۷- بعد از اتمام این مرحله خروجی به صورت شکل زیر خواهد بود و خط فرمان دوباره در دسترس می‌باشد. omnet++ به صورت کامل نصب شده است و با اجرای دستور omnetpp محیط شبیه‌سازی باز خواهد شد.

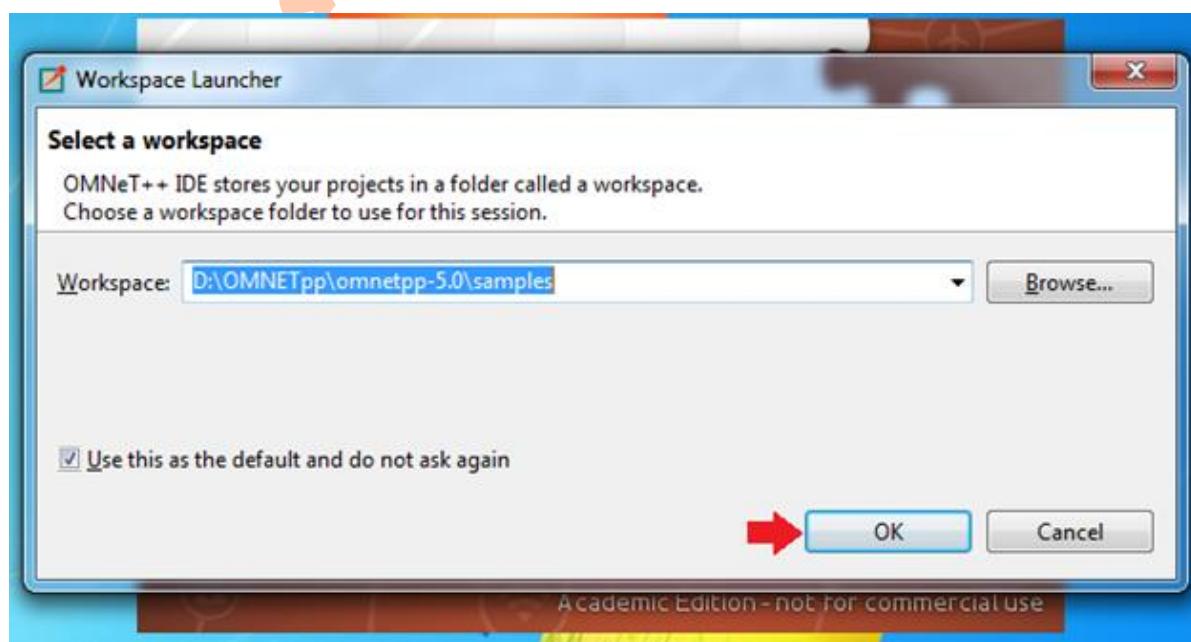


```
==== Compiling sockets ====
make[2]: Entering directory '/d/OMNETpp/omnetpp-5.0/samples/sockets'
Cloud.cc
ExtHttpClient.cc
ExtTelnetClient.cc
HttpClient.cc
HttpServer.cc
QueueBase.cc
SocketRTScheduler.cc
TelnetClient.cc
TelnetServer.cc
HttpMsg_m.cc
NetPkt_m.cc
TelnetPkt_m.cc
Creating executable: out/gcc-debug//sockets.exe
make[2]: Leaving directory '/d/OMNETpp/omnetpp-5.0/samples/sockets'
make[1]: Leaving directory '/d/OMNETpp/omnetpp-5.0'

Now you can type "omnetpp" to start the IDE
/d/OMNETpp/omnetpp-5.0$ omnetpp
```

شکل ۱-۶ اجرا کردن omnet++ با دستور omnetpp

۸- در هنگام باز شدن محیط شبیه‌سازی از شما آدرسی برای Workspace پرسیده می‌شود. توصیه می‌شود که محل پیش فرض را انتخاب کرده و تیک use this as the default and do not ask again را فعال کنید که در مراجعات بعدی انتخاب مسیر از شما پرسیده نشود.



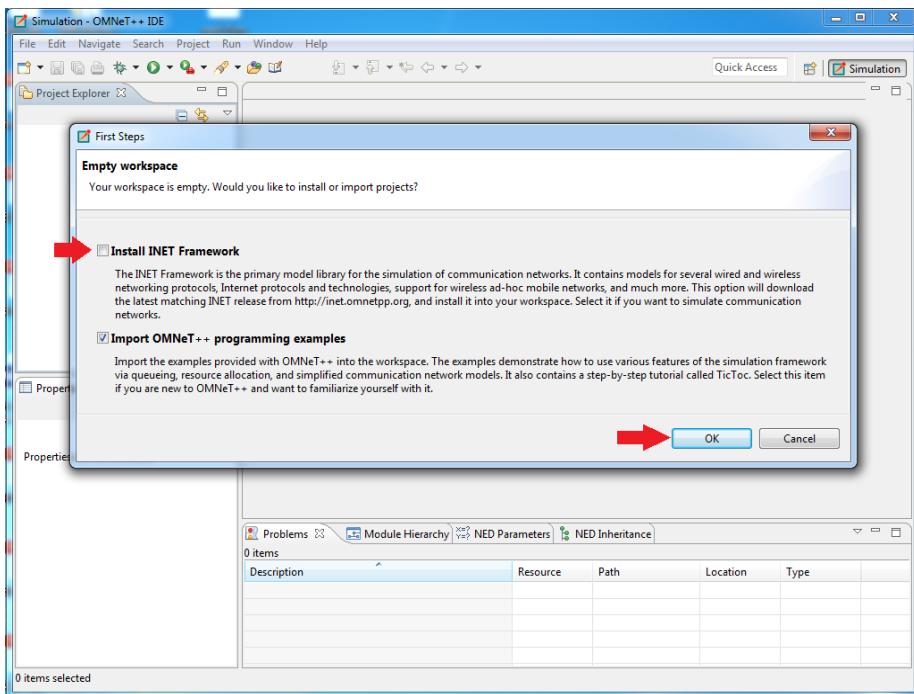
شکل ۱-۷ تنظیمات آدرس Workspace

۹- در اولین اجرا محیط شبیه سازی به صورت می باشد که منوی Workbench را انتخاب کنید.



شکل ۸-۱ تنظیمات اولیه نرم افزار

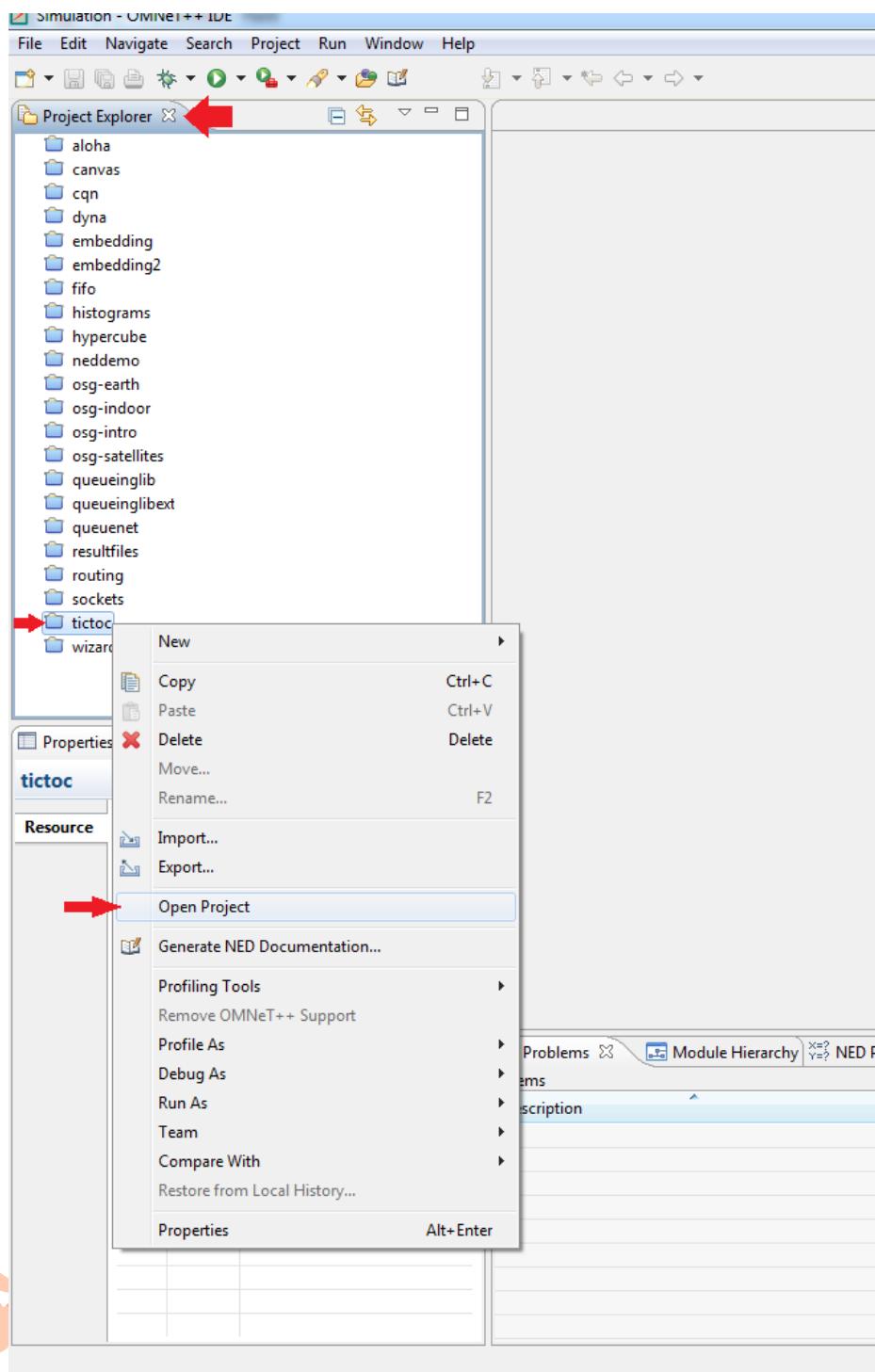
۱۰- همچنین در اجرای اول اجزا اضافه شدن مثال های omnet++ و نیز نصب INET پرسیده می شود که تیک نصب INET را غیر فعال کنید (INET شامل مثال هایی برای انواع پروتکل های شبکه می باشد که به صورت استاندارد نوشته شده اند و در این مرحله نیازی به نصب آن نیست).



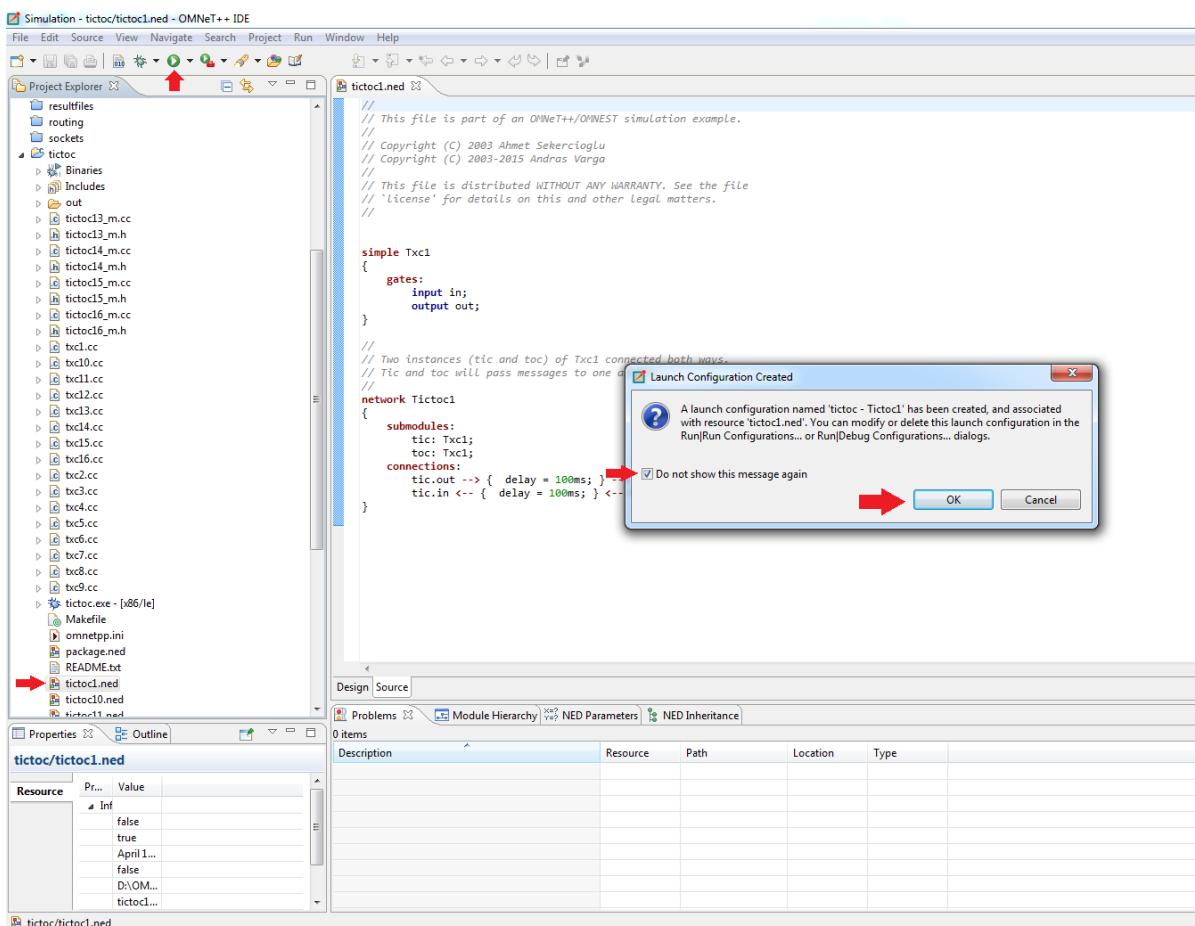
شکل ۱-۹ اضافه کردن پروژه های آماده

۱۱- به منظور اطمینان از درستی مراحل نصب در این قسمت یک مثال از پروژه های اضافه شده در را اجرا به صورت زیر اجرا کنید.

در قسمت Project Explorer روی tictoc کلیک راست کرده و Open Project را انتخاب کنید.

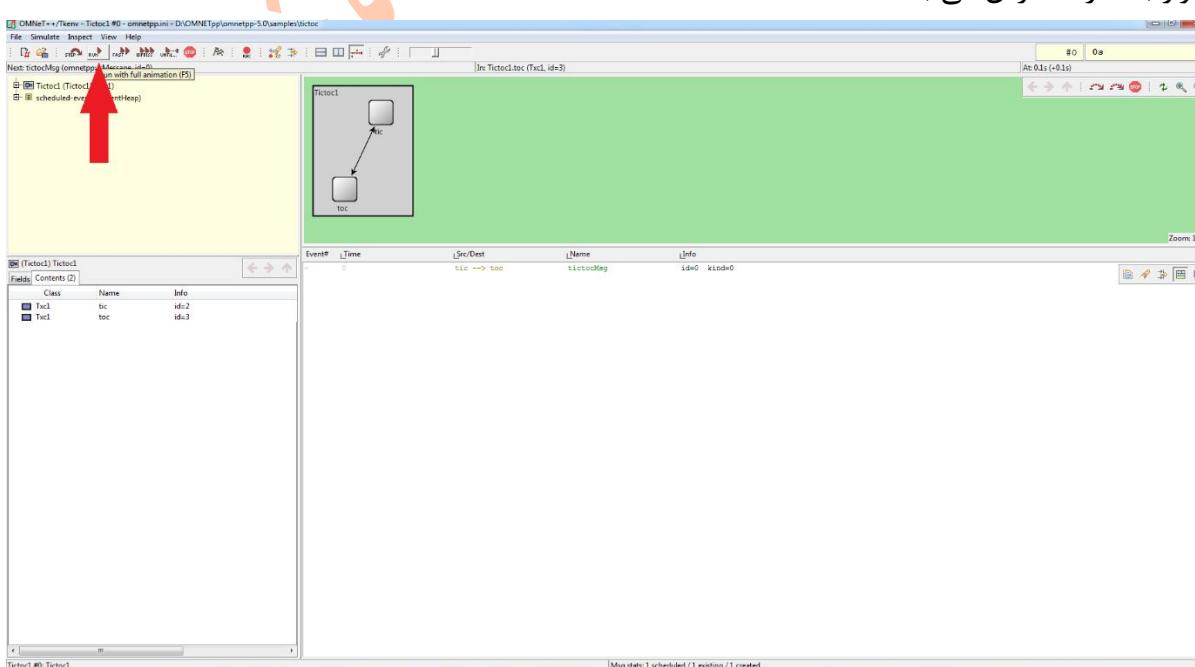


شکل ۱۰-۱ نحوه اجرا کردن یک پروژه(الف)

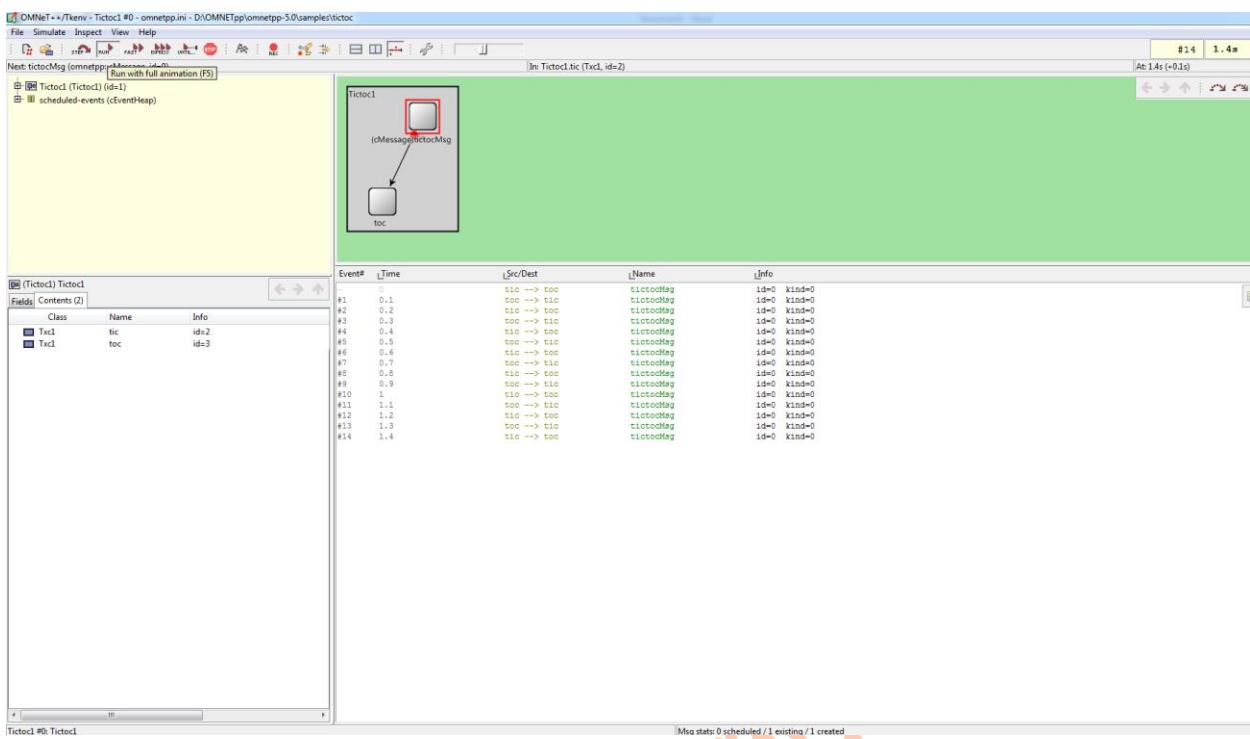


شکل ۱۱-۱ نحوه اجرا کردن یک پروژه(ب)

۱۲- در صفحه باز شده می توان در حالت های مختلفی (Step, Run, ...) پروژه را اجرا کرد که در نوار ابزار بالا در دسترس می باشند.



شکل ۱۲-۱ نحوه اجرا کردن یک پروژه(ج)



شکل ۱۳-۱ نحوه اجرا کردن یک پروژه(د)

فصل ۲ - آشنایی با محیط نرم افزار

-۱-۲ مقدمه

یک نرم افزار شبیه ساز مبتنی بر زبان برنامه نویسی C++ می باشد که از کتابخانه ها و چارچوب های متنوع این زبان استفاده می کند. وظیفه اصلی این نرم افزار در درجه اول، شبیه سازی ساختمان شبکه است.

نرم افزار OMNET++ قابلیت شبیه سازی انواع شبکه ها، پروتکل و استانداردهای متنوع شبکه را دارد و محبوبیت مطلوبی را در مقایسه با نرم افزار های مشابه شبیه ساز شبکه نظریer 2 , NS3, OPNet, NS2 QualNET, GloMoSIM, JSIM و ... به خود اختصاص داده است.

شبکه در این نرم افزار یک مفهوم گسترده تر را در بر می گیرد که شامل شبکه های ارتباطی سیمی و بی سیم است. با استفاده از این نرم افزار می توان روی تراشه شبکه ها برنامه ریزی کرد. بسته های ساخته شده در این شبیه ساز یک شبکه را صفت بندی کرد. همپنیین می توان با این نرم افزار، عملیات پیشرفته مسیریابی در سطوح مختلف را پیاده سازی کرد.

این شبیه ساز شبکه در سیستم عامل های windows و Mac توزیع های مختلف Linux قابل نصب می باشد. نرم افزار OMNET دارای Frame-work های متنوعی است که هر یکی از آنها عملکرد خاصی INET, INETMANET, OverSIM Veins, ReaSE, Castalia را در شبیه سازی ارائه می کند که از معروف ترین آنها.

توجه: تمام توضیحاتی که در ادامه می خوانید برگرفته از سایت omnetpp.org می باشد که لینک آن در زیر قرار داده شده است.

<https://omnetpp.org/doc/omnetpp/tictoc-tutorial>

بورسی مثال عملی شبیه سازی TicToc - ۲-۲

از آنجایی که بخش بسیار مهمی از نرم افزار OMNET++ شبیه سازی ارتباطات در شبکه است، در این جا به نحوه 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 طور این کار را تابی نهایت انجام می دهند. نام این دو گره "tic" و "toc" می باشد.

نحوه باز کردن فایل‌ها در انتهای فصل قبل توضیح داده شد. در ادامه برای شماره هر قدم tictoc مربوط به آن بررسی می شود که شامل فایلهای ned, .cc, .msg مربوطه خواهد شد.

۳-۲- شروع به کار

- ۱-۳-۲ قدم ۱: پیاده سازی اولیه

همان طور که در انتهای فصل قبل توضیح داده شد، فایل tictoc1.ned را باز کنید.

```
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 ۱-۲ پیاده سازی اولیه شبکه tictoc1 (الف)

بهتر است این فایل را از پایین به بالا بخوانید:

Tictoc1 شبکه‌ای است که از دو زیر مژول به نام tic و toc ساخته شده است. tic هر دو نمونه هایی از یک مژول یکسان به نام Txc1 هستند. گیت خروجی tic را که out نام دارد به گیت ورودی toc به نام in متصل است و به صورت عکس نیز همین کار انجام شده است، تاخیر انتشار در هر میسر نیز ۱۰۰ میلی ثانیه در نظر گرفته شده است.

Txc1 یک مژول ساده است و یک گیت خروجی به نام out و یک گیت ورودی به نام in دارد.

حال باید نحوه‌ی عملکرد مژول ساده Txc1 پیاده سازی شود.. این کار با نوشتن فایل txc1.cc با زبان c++ انجام می‌شود.

```
#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");
    }
}

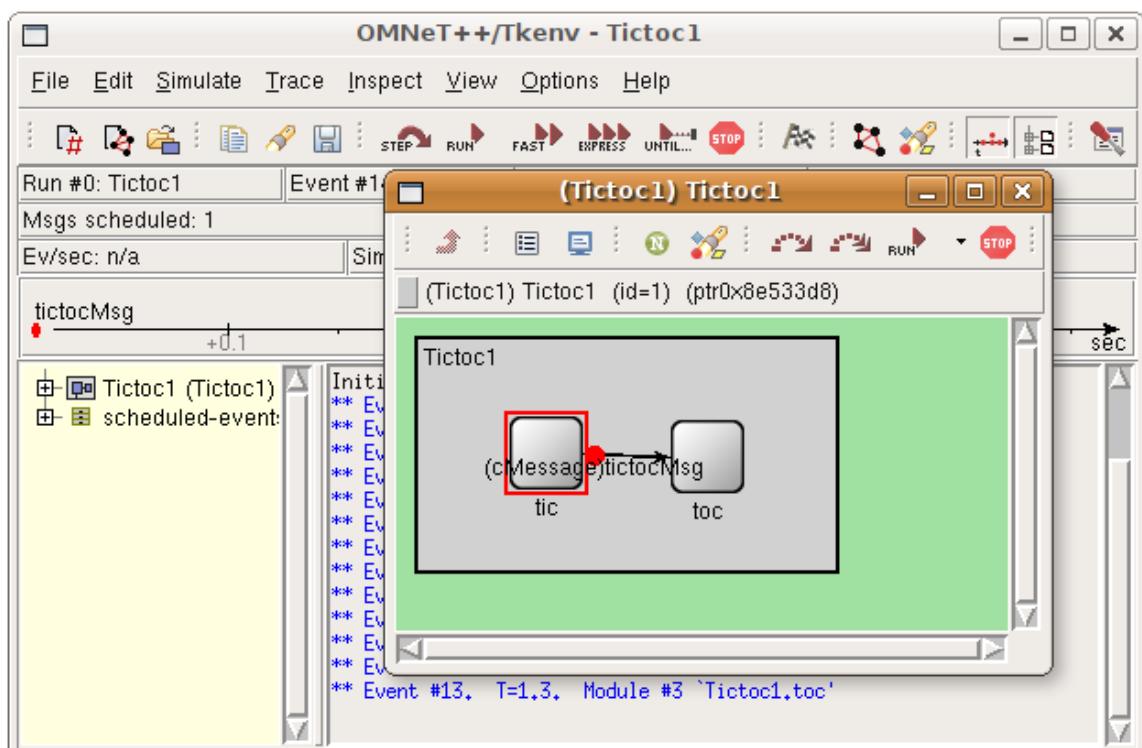
void Txc1::handleMessage(cMessage *msg)
{
    // The handleMessage() method is called whenever a message arrives
    // at the module. Here, we just send it to the other module, through
    // gate 'out'. Because both 'tic' and 'toc' does the same, the message
    // will bounce between the two.
   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

شکل ۲-۲ پیاده سازی اولیه شبکه 1 (ب)

ماژول ساده Txc1 با کلاس cSimpleModule از زبان C++ ارائه شده است که زیر کلاسی از OMNeT++ Define_Module() در ثبت شده است. همچنین دو تابع از cSimpleModule به نام های initialize() و handleMessage() به شوند: اولی تنها یک مرتبه و دومی هر زمان که پیغامی به ماژول برسد فراخوانی خواهد شد.

در initialize() شی از نوع cMessage ساخته شده و از طریق گیت out ارسال می شود. از آنجایی که این گیت به گیت ورودی ماژول دیگری متصل است، هسته شبیه ساز پیام را به ماژول دیگر متصل به سمت دیگر گیت، از طریق آرگمنی 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 میلی ثانیه تاخیر باز پس می فرستد.

پیام ها (پکت ها، فریم ها، job و...) و رویدادها (timers, timeout) به وسیله ی شی ای از cMessage یا یکی از زیر کلاس 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 آنها به پایان برسد و سپس از طریق handleMessage() به ماژول مقصد تحویل داده می شوند.



شکل ۳-۲ محیط اجرای شبیه سازی برای tictoc1

دکمه Run در toolbar شبیه ساز را بزنید تا تبادل پیام شروع شود. Toolbar پنجره 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 اختصاص داده نشده است و تنها زمان شبیه سازی سپری شده در اینجا مربوط به تاخیر انتقال پیام در اتصالات نودها است. می توانید سرعت اینیمیشن را از طریق دکمه slider موجود در بالای پنجره کم یا زیاد کنید.

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

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

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

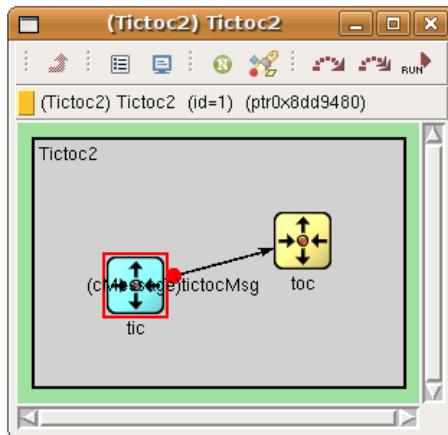
```
simple Txc2
{
    parameters:
        @display("i=block/routing"); // add a default icon
    gates:
        input in;
        output out;
}

// 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 ۴-۲ تعریف شبکه tictoc2

نتیجه این تغییرات به شکل زیر خواهد بود:



شکل ۵-۲ شبکه tictoc2 در محیط اجرای شبیه سازی

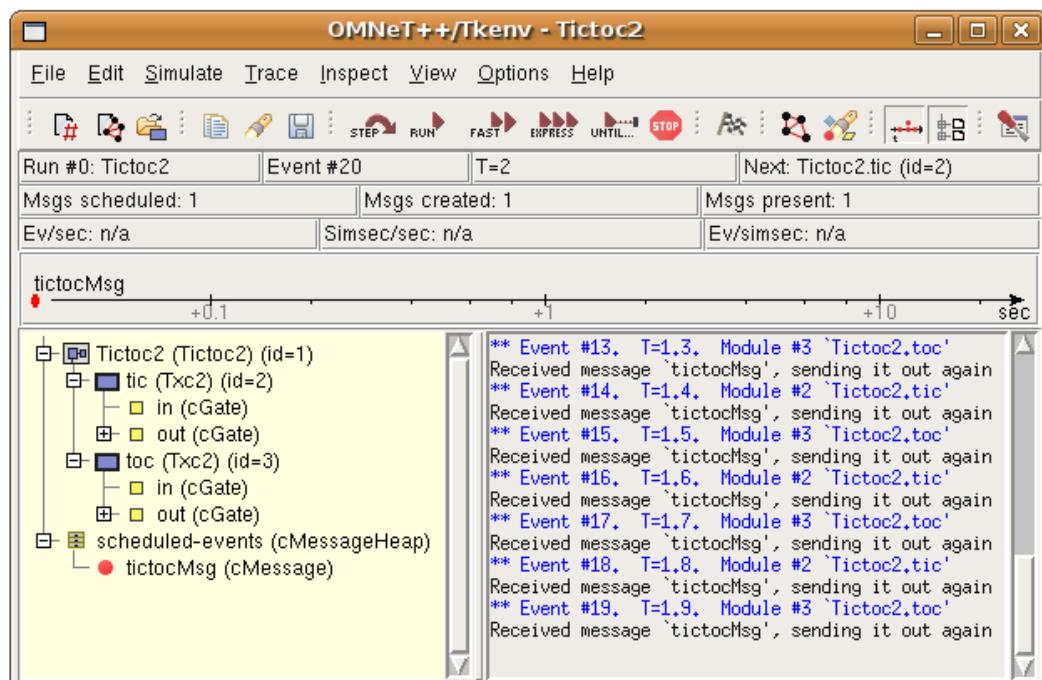
همچنین فایل C++ برای افروzen پیام‌های دیباگ به Txcl ، به وسیله شی EV تغییر کرده است:

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

۶

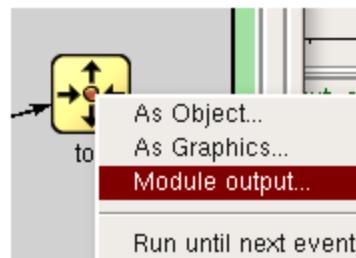
```
EV << "Received message '" << msg->getName() << "' , sending it out again\n";
```

اگر شبیه سازی را اجرا کنید، خروجی زیر را در پنجره متنی خواهید دید:



شکل ۶-۲ تغییرات شبکه tictoc2 در محیط اجرای شبیه سازی (الف)

همچنین می توانید پنجره های جداگانه ای برای هر یک از toc یا tic باز کنید تا خروجی مربوط به خودشان را در آن مشاهده نمایید..این کار مناسب زمانی است که مدل بزرگی دارد و تنها علاقمند هستید که خروجی مربوط به یکی از مازول 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 ۷-۲ تغییرات شبکه tictoc2 در محیط اجرای شبیه سازی (ب)

۲-۵-۳- قدم ۳: افزودن متغیرهای وضعیت

در این مرحله شمارنده ای به مازول اضافه شده و پیام را بعد از ده مرتبه رد و بدل شدن بین گره ها حذف می کند.

شمارنده به شکل عضو کلاس اضافه می شود::

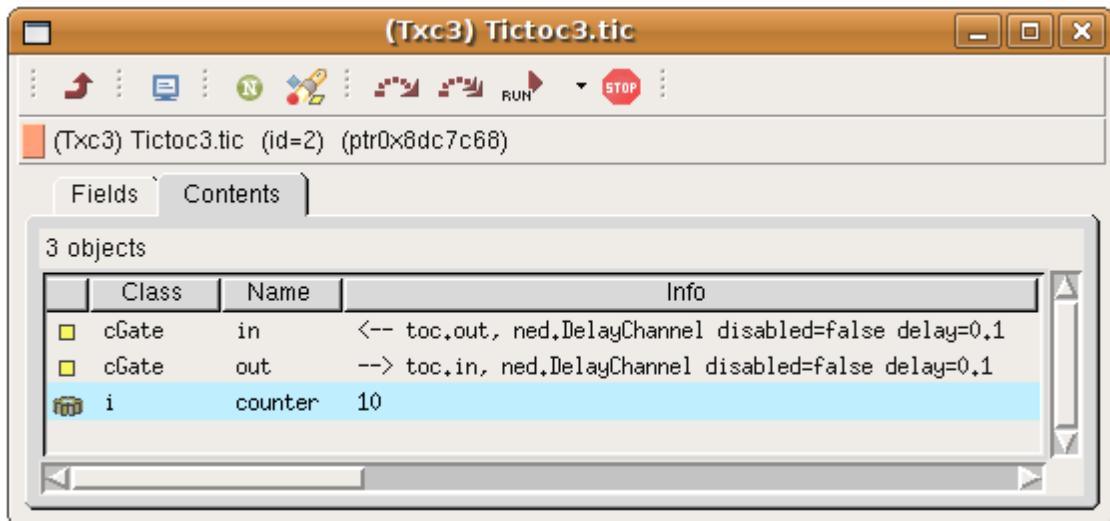
```
class Txc3 : public cSimpleModule
{
    private:
        int counter; // Note the counter here
    protected:
```

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 را ممکن می سازد..بر روی آیکن tic کلیک کنید، سپس صفحه Contens را از پنجره باز شده انتخاب کنید.



شکل ۸-۲ مشاهده مقدار یک متغیر در شبکه tictoc3 در محیط اجرای شبیه سازی

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

۶-۴- قدم ۴: افزودن پارامترها

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

پارامترهای مازول باید در فایل NED تعریف شوند. نوع داده می تواند عددی، رشته، بولی و یا 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:

```

برای خواندن پارامتر و اختصاص آن به شمارنده در () initialize() کد C++ نیز کمی تغییر کرده است:

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

از دومین پارامتر نیز برای تصمیم اینکه آیا پیغام 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 پیش فرض تعریف کنید که در این صورت می‌توانید 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 یکی از پارامترها در NED :

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

و دیگری در omnet.ini تعریف شده است:

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

از آنجایی که wildcard omnetpp.ini پشتیبانی می‌کند و مقداردهی پارامترها از طریق فایل‌های NED بر اولویت دارند، می‌توان از این روش‌ها استفاده کرد:

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

یا

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

یا حتی

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

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

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

۷-۵- قدم ۵: استفاده از ارث بری

اگر با دقت به فایل NED نگاه کنید خواهید فهمید که tic و toc تنها در مقدار پارامترها و رشته display خود با یکدیگر متفاوت‌اند، می‌توان نوع ماثول ساده جدیدی از طریق ارث بری ساخت و بعضی از پارامترها را override کرد. در اینجا دو ماثول ساده Tic و Toc داریم که مشتق شده‌اند.

ارث بری از یک ماثول ساده به آسانی انجام می‌شود:

```
simple Tx5
{
    parameters:
        bool sendMsgOnInit = default(false);
        int limit = default(2);
        @display("i=block/routing");
    gates:
        input in;
        output out;
}
```

در ماثول مشتق شده تنها مقدار پارامترها مشخص می‌شود:

```
simple Tic5 extends Tx5
{
    parameters:
        @display("i=,cyan");
        sendMsgOnInit = true; // Tic modules should send a message on init
}
```

در ماثول Toc نیز به همین صورت است، تنها مقادیر پارامترها متفاوت‌اند.

```

simple Toc5 extends Tx5
{
    parameters:
        @display("i=gold");
        sendMsgOnInit = false; // Toc modules should NOT send a message on init
}

```

هنگامی که مژول‌های ساده جدید می‌سازیم، می‌توانیم از آنها به عنوان submodule در شبکه 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:

```

همانطور که می‌بینید تعریف شبکه بسیار کوتاه‌تر و ساده‌تر شده‌است.

۸-۲- قدم ۶: مدل سازی تاخیر پردازش

در مدل‌های قبلی، tic و toc بلافصله بعد از دریافت پیغام آن را باز پس می‌فرستادند. در اینجا tic و toc به گونه‌ای تغییر می‌کنند که پیغام را قبل از ارسال مجدد به اندازه یک ثانیه شبیه سازی شده نگه دارند. در OMNeT++ یک چنین شبیه سازی از طریق ارسال پیغام به خود بدست می‌آید. این پیغام‌ها self-messages نامیده می‌شوند (تنها به دلیل نحوه استفاده از آنها و گرنه آنها نیز همانند دیگر پیغام‌ها هستند). دو متغیر از نوع اشارگر به نامهای cMessage و event به کلاس tictocMsg به کلاس افروده می‌شود. یکی برای ارسال پیغام به خود و دیگری برای ارسال پیغام به مژول دیگر.

```

class Tx6 : 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:

```

به وسیله‌ی تابع scheduleAt() به خود گره فرستاده می‌شود و از طریق آرگمندان Self-messages آن می‌توان زمانی که باید پیغام تحويل داده شود را تعیین کرد.

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

حال در handleMessage() باید نوع پیام دریافتی را تشخیص داد:

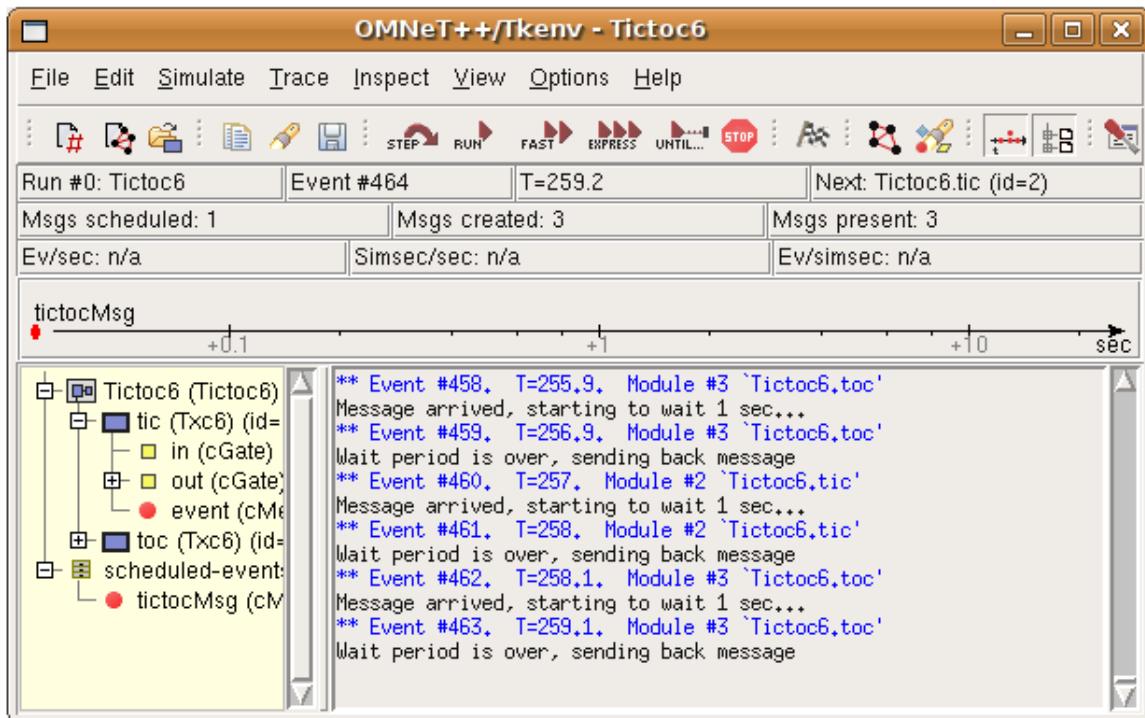
```
if (msg==event)
```

همچنین این کار را می توان به روش زیر انجام داد:

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

در اینجا به منظور سادگی بخش مربوط به شمارنده حذف شده است.

نتیجه اجرای شبیه‌سازی در زیر نشان داده شده است:



شکل ۹-۲ مدل‌سازی تاخیر پردازش در شبکه tictoc6

۹-۷- قدم ۷: شماره تصادفی و پارامترها

در این مرحله شماره های تصادفی معرفی می‌شوند، زمان تاخیر از ۱ ثانیه به یک مقدار تصادفی که می‌تواند از طریق فایل NED و یا omnetpp.ini تنظیم شود تغییر می‌کند. پارامترهای مازول قادر هستند تا متغیرهای تصادفی بازگردانند؛ به هر حال برای استفاده از این ویژگی باید پارامتر را هر بار که از آن استفاده می‌شود در handleMessage() فراخوانی کرد.

```

// 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);

```

همچنین با یک احتمال بسیار کم پکت از بین می‌رود (آن را به دست خودمان پاک می‌کنیم).

```

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

```

پارامترها را در omnetpp.ini مقداردهی می‌شوند:

```

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

```

مهم نیست چند بار شبیه سازی اجرا می‌شود (یا restart می‌شود)، می‌توان امتحان کرد، در هر بار نتایج دقیقاً یکسان دریافت خواهد شد. این بدان خاطر است که OMNeT++ برای تولید اعداد تصادفی از یک الگوریتم قطعی استفاده می‌کند و آنرا در هر بار با یک seed یکسان مقداردهی می‌کند. این امر برای شبیه سازی‌های قابل تکرار بسیار مهم است. در صورت تمایل می‌توانید شبیه سازی را با seed متفاوت امتحان کنید. برای اینکار تنها کافی است خط زیر را به omnetpp.ini بیافرایید:

```

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

```

۱۰-۲- قدم ۸: لغو تایمراه و اتمام زمان

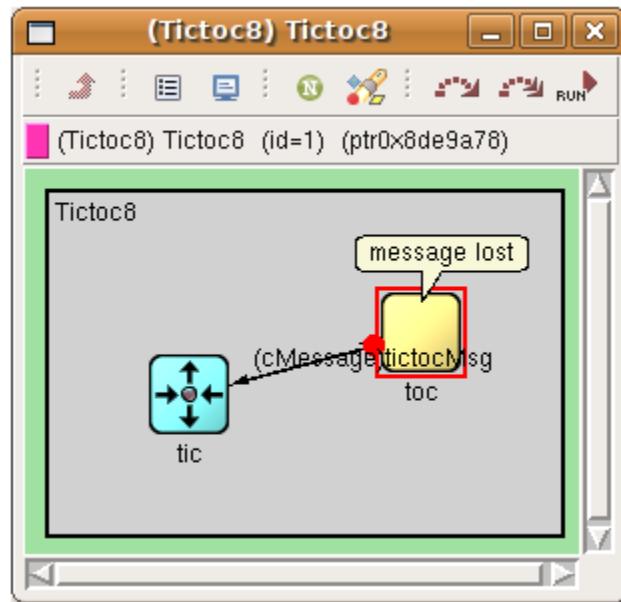
برای اینکه یک گام به مدلسازی پروتکلهای شبکه نزدیکتر شویم، مدل تبدیل به یک مدل از نوع شبیه سازی stop-and-wait می‌شود. این بار کلاسهای tic و toc از هم جدا می‌شوند. همانند دفعه قبل tic و toc یک پیغام را به هم پاس می‌دهند و toc با یک احتمال کم پیغام را گم می‌کند اما این بار در صورت رخ دادن یک چنین اتفاقی tic شروع به ارسال مجدد پیغام می‌کند..

```

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

```

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



شکل ۱۰-۲ نمایش از بین رفتن پکت‌ها در شبکه tictoc8 در محیط اجرای شبیه‌سازی

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

```

scheduleAt(simTime() + timeout, timeoutEvent);

cancelEvent(timeoutEvent);

```

لغو تایمر:

۱۱-۹- قدم ۹: ارسال دوباره یک پیغام

در مرحله قبل تنها در صورتی که نیاز به ارسال مجدد یک پکت ساخته می شد اما کاری که ما در اینجا انجام می شود نگهداری پیغام اصلی و تنها ارسال کپی هایی از آن است که دیگر نیاز به ساخت مجدد آن نباشد و زمانی که پیغام تایید toc رسید پیغام اصلی را پاک می کنیم. برای آسان تر شدن درک مدل در محیط visual، به نام پیغام ها شماره پیغام نیز اضافه شده است. برای اجتناب از بزرگ شدن بیش از حد() دوتابع handleMessage() و generateNewMessage() به نام های sendCopyOf() و handleMessage() ساخته شده و در handleMessage() فراخوانی می شوند:

```
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;
}

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

۱۲-۱- قدم ۱۰: تبدیل به یک شبکه واقعی

۱۲-۲- قدم ۱۰: بیش از دو گره

در اینجا: چندین مژول tic ساخته شده و به یک شبکه متصل شده است. فعلاً کاری که انجام می دهن را ساده در نظر می گیریم: یکی از گره ها پیغامی ساخته و ارسال می کند، این پیغام بین گره ها به صورت تصادفی پاس داده می شود تا زمانی که پیغام به مقصد از قبل تعیین شده برسد. فایل NED نیاز به کمی تغییر دارد. اول از همه مژول 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[];
}
```

کاراکترهای [] گیت ها را تبدیل به برداری از گیت ها می کند. اندازه بردار (تعداد گیت ها) زمانی تعیین می شود که از Txc10 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++;

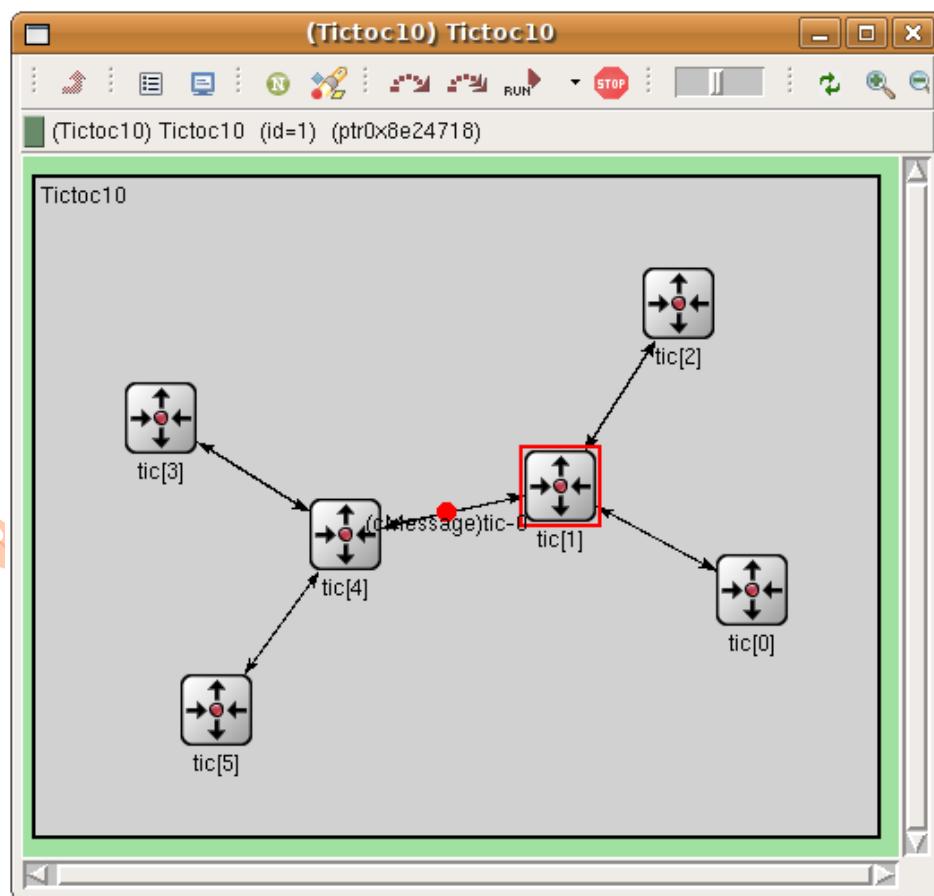
        tic[1].out++ --> { delay = 100ms; } --> tic[2].in++;
        tic[1].in++ <-- { delay = 100ms; } <-- tic[2].out++;

        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++;
}
```

در اینجا ۶ مازول با کمک یک بردار مازول ایجاد شده و به هم متصل شده‌اند. نتیجه این کار این می‌شود::



شکل ۱۱-۲ تصویری از شبکه tictoc10 در محیط شبیه اجرای شبیه سازی

در این نسخه، tic[0] پیغامی را برای ارسال ایجاد می کند. قسمت اصلی کد تابع forwardMessage() است که آن را هر زمان که پیغامی به گره می رسد در تابع handleMessage() فراخوانی می کنیم. این تابع یک عدد تصادفی از شماره گیت‌ها تولید کرده و پیغام 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);
}
```

زمانی هم که پیغام به دست tic[3] برسد، پیغام را در handleMessage() خود حذف خواهد کرد..

۱۳-۱۱- قدم کانال‌ها و تعریف آنها

تعریف ساختار شبکه تا بدین جا بسیار پیچیده و طولانی شده است، این امر مخصوصا در مورد بخش ارتباطات صادق است. 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 را با ارتباطات موجود تعویض می کنیم.

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

نوع channel جدید درون شبکه و با افزودن بخش types تعریف شده است. این نوع تعریف کردن تنها درون شبکه قابل رویت است که به آن نوع محلی یا داخلی می گویند. در صورت تمایل می توانیم ماژول های ساده را نیز به شکل نوع های داخلی تعریف کنیم. در ادامه تغییرات به وجود آمده در بخش connections را بررسی می 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++;
}

```

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

۱۴-۱۲- قدم: استفاده از اتصالات دو طرفه

اگر بخش connections را کمی بیشتر بررسی کنید، متوجه خواهید شد که هر جفت از گره ها از طریق دو اتصال به یکدیگر متصل هستند. می توان از این امکان OMNeT++ نیز استفاده کرد. برای انجام این کار ابتدا باید گیت های دو طرفه inout یعنی input و output نمود.

```

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

```

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++;
}

```

با تغییر نام گیت ها کمی در کد C++ نیز تغییر ایجاد خواهد شد :

```

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);
}

```

۱۵-۲-قدم ۱۳: تعریف کلاس پیغام

در این مرحله دیگر آدرس مقصد به صورت دستی مشخص نمی شود بلکه یک مقصد تصادفی تولید شده و آدرس مقصد پیغام اضافه می شود. بهترین راه، ساختن کلاسی مشتق شده از کلاس cMessage و افزودن مقصد به شکل data member به آن است. با تعیین مشخصات کلاس پیغام در فایل msg دیگر نیاز به نوشتن خود کلاس پیغام به زبان C++ نیست و OMNeT++ این قابلیت را دارد که کلاس مربوط به پیغام را تولید کند.. کلاس تولید شده برای هر یک از فیلد ها متدهای getter و setter دارد.. نام فایل تعریف کلاس پیغام که در زیر نشان داده شده TicTocMsg13 است. بنابراین کامپایلر دو فایل به نام های tictoc13_m.cc و tictoc13_m.h تولید خواهد کرد..

```

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

```

حال باید C++ در کد tictoc13_m.h include شود و پس از آن می توان از TicTocMsg13 مانند هر کلاس دیگر استفاده کرد.
برای مثال در زیر که بخشی از تابع generateMessage() است نحوه ایجاد یک پیغام و پر کردن فیلد های آن نشان داده شده است:

```

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

```

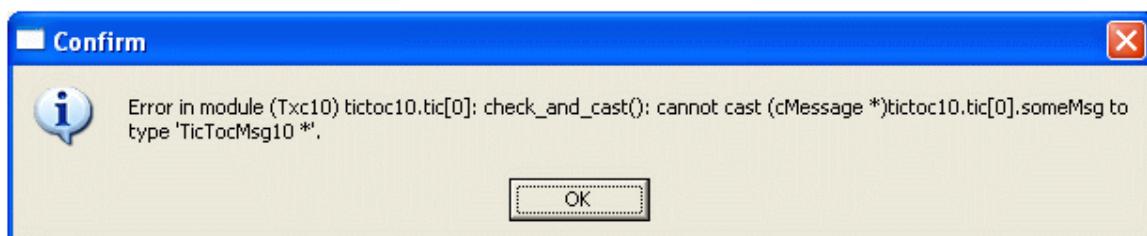
تابع handleMessage() نیز بدین شکل شروع می شود:

```

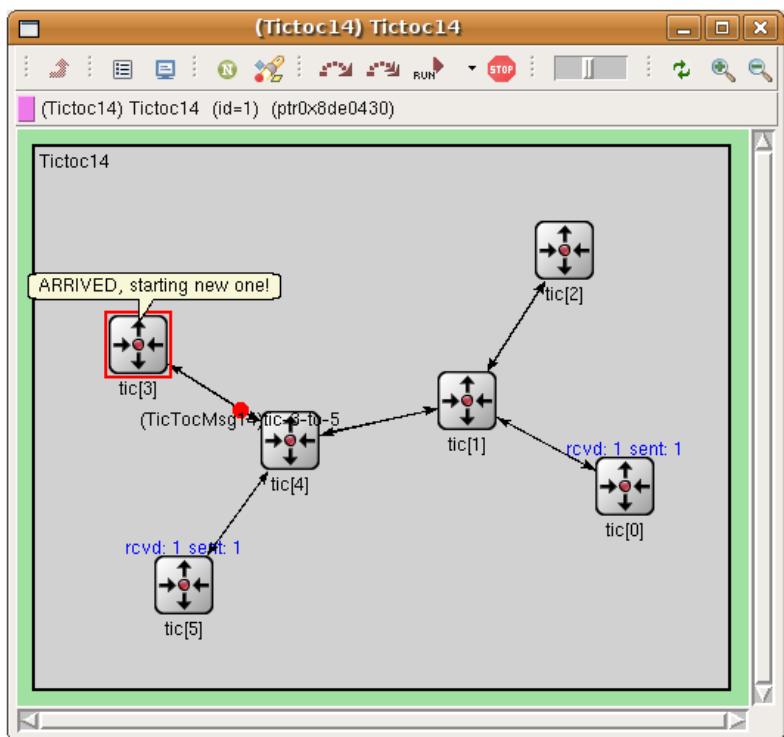
void Txc13::handleMessage(cMessage *msg)
{
    TicTocMsg13 *ttmsg = check_and_cast<TicTocMsg13 *>(msg);
    if (ttmsg->getDestination() == getIndex())

```

در آرگمان تابع handleMessage() اشاره گری به کلاس cMessage دریافت شده، در صورتی که msg دریافت شده را به صورت عادی به اشاره گری از TicTocMsg13 تبدیل (cast) کنیم امن نخواهد بود؛ زیرا در صورتی که msg دریافت شده به هر دلیل از نوع TicTocMsg13 نباشد برنامه کرش کرده و یافتن علت خطا بسیار سخت خواهد شد. C++ راه حل دیگری به نام dynamic_cast معرفی کرده است. در اینجا از check_and_cast استفاده شده که توسط OMNeT++ فراهم شده است که سعی می‌کند 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 می‌کند.

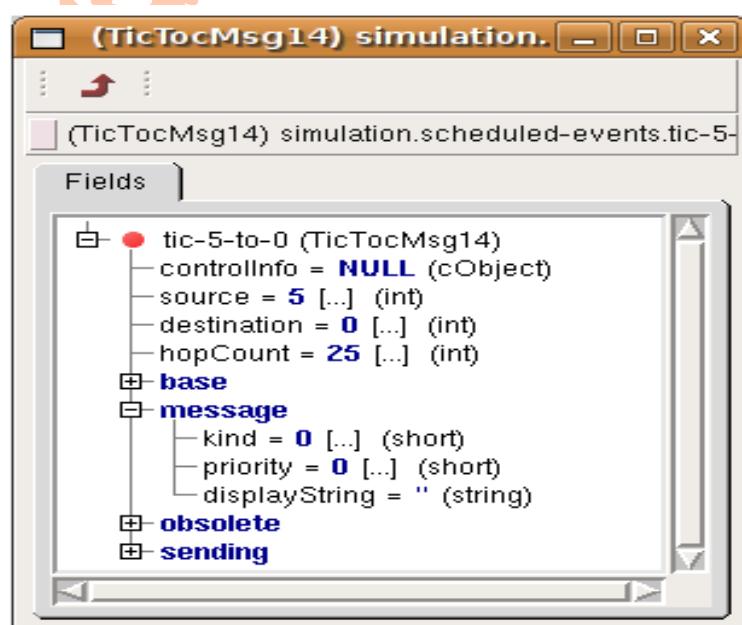


در خط بعد از کد نشان داده شده در بالا، بررسی می‌کنیم که آیا آدرس مقصد با آدرس گره یکسان است یا خیر. تابع عضو getIndex() اندی ماذول در بردار زیرماژول را بر می‌گرداند (فایل NED را به خاطر بیاورید). برای اجرای طولانی مدت تر مدل، پس از آنکه پیغامی به مقصد خود باز می‌گردد، گره مقصد پیغامی دیگر با یک آدرس مقصد تصادفی تولید کرده و به همین ترتیب این کار ادامه می‌یابد. اگر مدل را اجرا نمایید، خروجی زیر را خواهید دید.



شکل ۱۲-۲ تصویری از شبکه tictoc14 در محیط اجرای شبیه سازی (الف)

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



شکل ۱۳-۲ تصویری از شبکه tictoc14 در محیط اجرای شبیه سازی (ب)

۱۶-۲-افزودن مجموعه های آماری

۱۶-۲-۱: نمایش شماره بسته های ارسال شده و دریافت شده

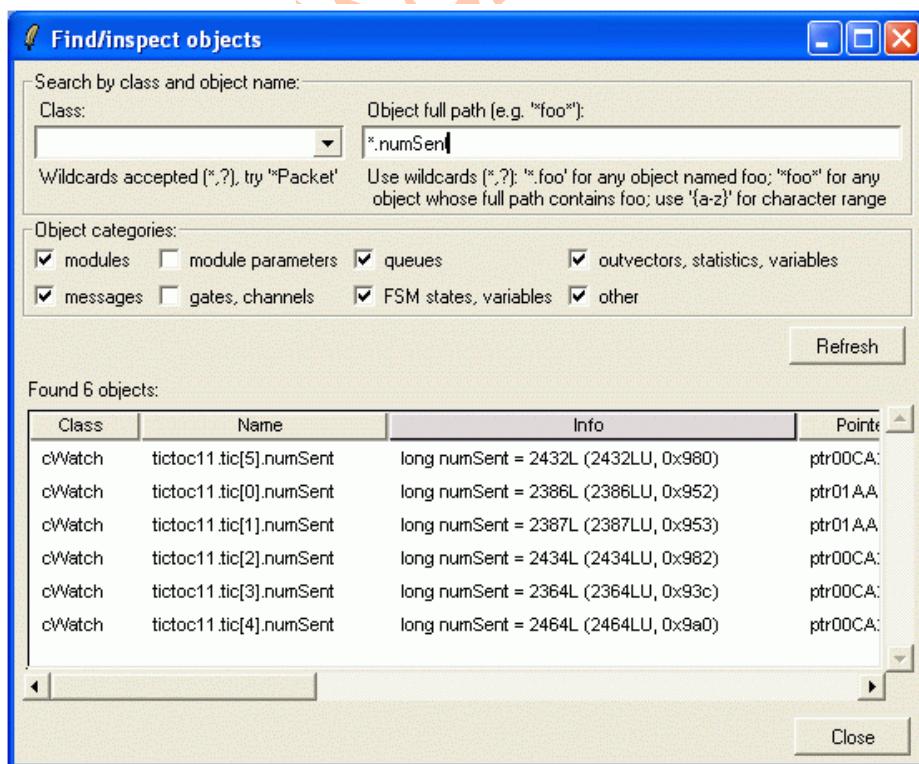
برای دیدن تعداد پیغام هایی که هر گره ارسال یا دریافت کرده است، دو شمارنده به کلاس ماژول اضافه شده است::

numReceived و numSent

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

    protected:
```

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



شکل ۱۶-۲ نمایش بسته های دریافت شده و ارسال شده

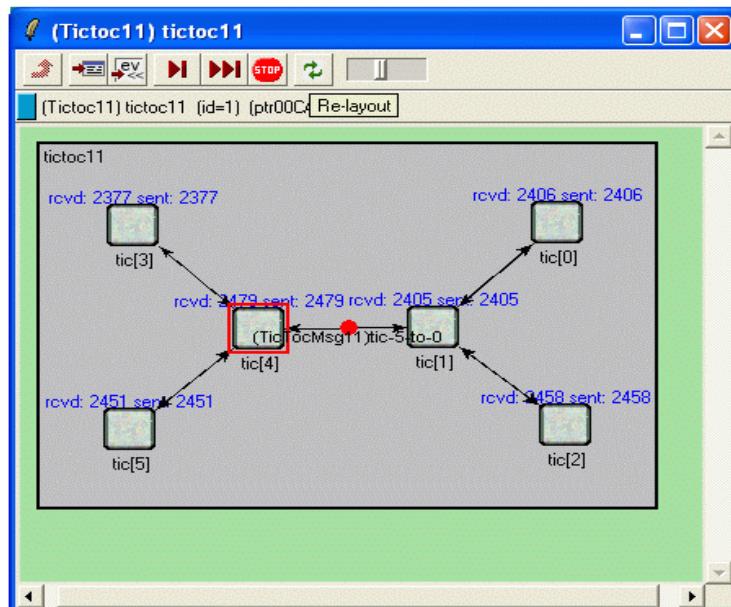
درست است که در این مدل شبیه سازی شده تعداد ارسالی و دریافتی یکسان است اما در شبیه سازی های واقعی این قابلیت که بتوان مروری سریع بر وضعیت گره های مختلف در مدل داشت بسیار ارزشمند خواهد بود. همچنین این اطلاعات را می توان در بالای آیکن هر ماژول نیز نشان داد. رشته `t` در متن موجود در بالای آیکن را مشخص می کند؛ تنها باید این رشته را در زمان اجرا تغییر دهیم. کد زیر این کار را انجام می دهد.

```
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 خطأ مواجه می شود. مثلاً غیرفعال بودن GUI زمانی اتفاق می افتد که در حین اجرای برنامه تنها پنجره‌ای که نمایش دهنده گرافیک مدل است (و نه پنجره اصلی Tkenv) را ببندیم. در این صورت 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);
}
```

نتیجه شکل زیر خواهد شد:



شکل ۱۵-۲ تغییر محیط گرافیکی در حین اجرای شبیه سازی

۱۷-۲-۱۵: افزودن مجموعه های آماری

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

1 Event log -۱

Output Vector -۲

3 Output Scalars -۳

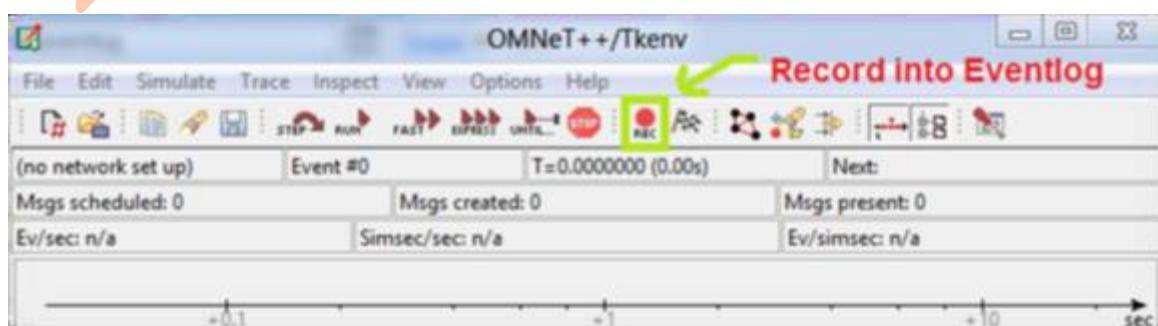
که در ادامه هر یک را مختصرا توضیح می دهیم:

Event log -۱-۱۷-۲

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 برنامه قرار می گیرد. این قابلیت در نسخه های قبلی وجود ندارد. هدف از ایجاد این قابلیت در OMNeT++ درک آسان تر 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 می توان لحظه به لحظه شبیه سازی را از ابتدا تا انتهای و بر عکس از انتهای 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 مثلاً تنها پیغام 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 مثلاً وابستگی علت و معلول (causality) که در سیستم عامل بسیار برای ما مهم است را به راحتی بررسی نمود و یا بر اساس زمان شبیه سازی، شماره رخدادها (مثلاً ارسال یا دریافت و یا پردازش یک پیغام، یک رخداد محاسب می شود)، مارژول ها و پیام ها، فیلترهایی را در این لگ اعمال کرد و هر یک را به تنها یی تحلیل نمود. هسته شبیه ساز OMNeT++ می تواند با تنظیم دستور زیر در فایل omnetpp.ini، لگ دقیق خودکاری از تبادل پیام ضبط کند.

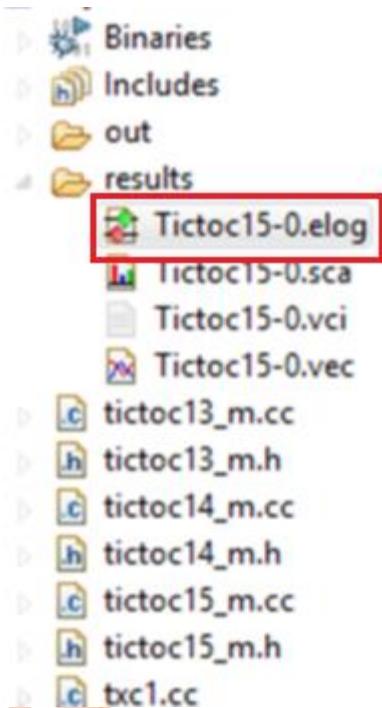
```
record-eventlog = true
```

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



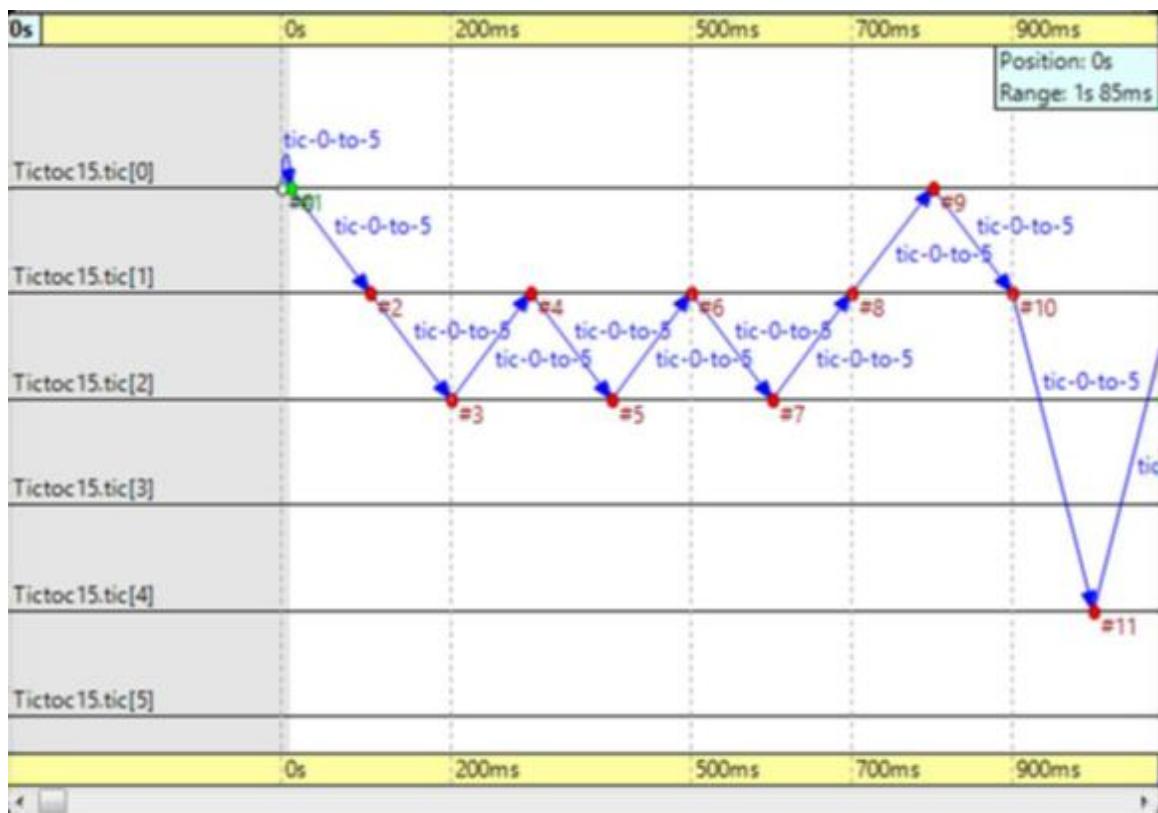
شکل ۱۶-۲ Record کردن در محیط اجرای شبیه سازی

در مورد مثال Tictoc ، پس از اتمام شبیه سازی در پوشه results درون پوشه برنامه، فایل لاغی با نام Tictoc15-0.elog ایجاد می شود . این لگ را در IDE می توان به کمک نمودار توالی ۱ نمایش داد. دقت کنید که در صورتی که تبادل پیام در سیستم زیاد باشد این فایل واقعاً بزرگ خواهد شد پس تنها در صورتی که بدان نیاز دارید این امکان را فعال کنید. برای مشاهده ای نمودار توالی ابتدا باید در IDE بر روی فایل *.elog دابل کلیک نمایید.



شکل ۱۷-۲ اجرای فایل elog. برای Tictoc15

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



شکل ۱۸-۲ خروجی خاصل از اجرای فایل eLog برای Tic toc 15

همانطور که مشاهده می کنید محور افقی این نمودار نشان دهنده زمان می باشد و در سمت چپ نمودار و در محور عمودی نام مازول های موجود در شبکه نشان داده می شود. در بخش اصلی این نمودار به وسیله فلش های آبی تبادل پیام مابین مازول ها نشان داده می شود. برای مثال در نمودار بالا فلش نشان می دهد که [0] ابتدا پیغام tic-0-to-5 را برای خود ارسال کرده است و پس از دریافت آن، فلش آبی مستقیم نشان دهنده ای آن است که پیغام را به [1] فرستاده است. همچنین در این نمودار شماره 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 نمایش داده شده بوسیله کادر سیزرنگ گوشه بالا سمت راست تصویر مشخص می شود. نرم افزار OMNeT++ IDE ابزارهای زیادی را برای کار با این نمودار ارائه کرده است.

۲-۱۷-۲ راه هایی برای کوچک کردن حجم لگ 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 مازول ها ذخیره می شود. فاکتور دیگری که می توان برای کوچک کردن اندازه لاغ انجام داد، مشخص کردن مازول هایی است که می خواهید رفتار آنها را ضبط کنید. گزینه module-eventlog-recording به کرنل دستور می دهد تا تنها رویدادهایی که برای مازول مشخص شده رخ می دهد را ذخیره کند. این آپشن تنها بر روی مازول های ساده قابل اعمال است. مثال:

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

این مثال به کرنل دستور می دهد که تنها رویدادهای مسیریاب هایی که اندیس آنها بین ۱۰ تا ۲۰ است را ذخیره کند و ضبط رویداد را برای دیگر مازول ها غیر فعال نماید.

Output Vector - Output Scalars -۳-۱۷-۲

OMNeT++ از طریق output vectors و output scalars امکان ضبط نتیجه شبیه سازی را فراهم کرده است. بردارهای خروجی، داده های سری های زمانی می باشند که از مازول های ساده و یا کانال ها ضبط شده اند و می توان از آنها برای ضبط تا خیره های از مبدأ به مقصد (یا زمان end - to - end) رفت و برگشت پکت ها، طول صفر، وضعیت مازول، 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 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، فایل هایی متنی می باشند. مزیت فرمت متنی آن است که بوسیله 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 (OutputVector) می گویند. از متدها record() برای نوشتن یک مقدار به همراه مهر زمانی (timestamp) استفاده می شود. در صورتی که چندین شیء 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 در صورت 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 خاص زمانی انجام دهید.

در حالی 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 scalers) 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 seed 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 مت () از record() کلاس cSimpleModule بر روی یک فایل با پسوند .sca. ضبط می گردد. اغلب این مت را درون تاب finish() قرار می دهد..

٤-١٧-٢ 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 پیغام برای رسیدن به مقصد باید طی کند، بدست آورد. ما تعداد گام های (hop) هر پیغام 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 Tictoc15-0.vec ذخیره خواهد کرد و یک شئ نمودار cLongHistogram (که میانگین و حداقل را محاسبه می کند و داده ها را در Tictoc15-0.sca ذخیره می کند) به کلاس ماثول اضافه می کنیم.

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

    protected:
```

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

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

فراخوانی () hopCountVector.record() داده ها را در Tictoc15-0.vec می نویسد. از آنجایی که اجرای طولانی مدت این شبیه سازی و یا شبیه سازی یک مدل بزرگ باعث حجمیم 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

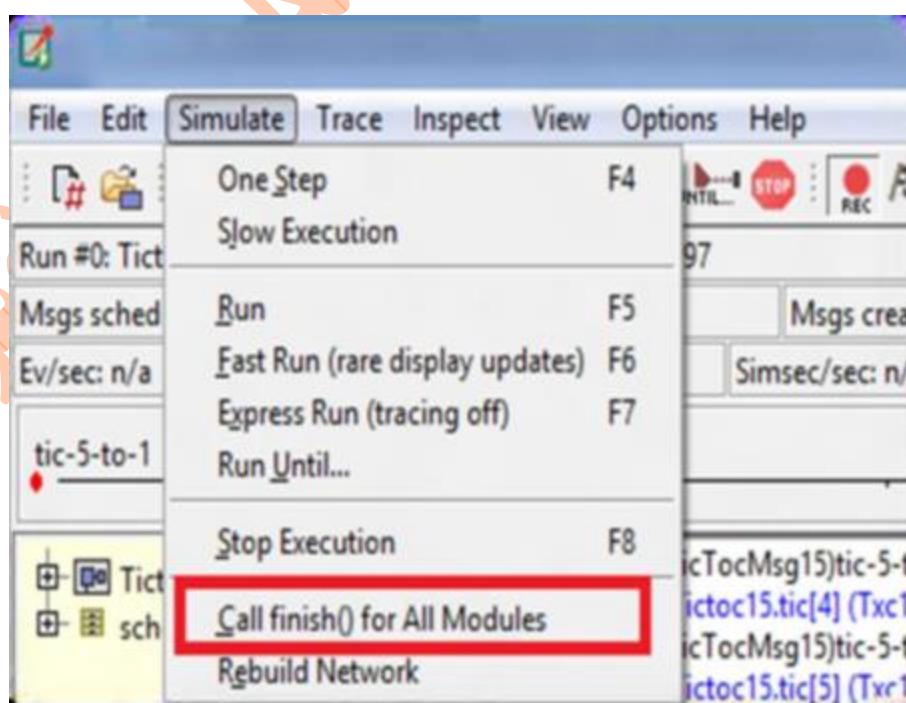
اسکالر که توسط شی histogram در این شبیه سازی جمع آوری شده اند باید به طور دستی در متند finish() ضبط گردند. این تابع بعد از اتمام شبیه سازی و در صورت اجرای موفق آن فراخوانی می گردد و در صورتی که شبیه سازی با خطا متوقف شود این تابع فراخوانی نمی شود. فراخوانی تابع Tictoc15-0.sca در کد زیر باعث نوشته شدن اطلاعات در recordScalar() می گردد.

```
void Txcl5::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");
}
```

هنگامی که فکر می کنید به اندازه 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 Tictoc15-0.sca باید از منوی پنجره‌ی اصلی Simulate و سپس Call finish() for All Modules را انتخاب کنید.



شکل ۱۹-۲ فرخوانی تابع `finish()`

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

```
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 ۲۰-۲ اطلاعات تابع `finish()` (الف)

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

۵-۱۷-۲ توضیح تابع finish()

در این مثال ۶ ماثول ساده از نوع `Txc15` تعریف شده که هر یک دارای متغیرهای `numSent` و `numReceived` هستند که مشخص کننده تعداد پیام های دریافتی و ارسالی آنها `hopCountStats` می باشد (که با هم برابرند). همچنین برای هر ماثول شی از نوع `LongHistogram` وجود دارد. در این تابع ابتدا در خروجی تعداد پیام های دریافتی و ارسالی چاپ شده و سپس با استفاده از توابع شی `hopCountStats` آمار گام های طی شده مربوط به پیام های رسیده به هر ماثول را در خروجی نمایش می دهد.

• تابع : `getMin()` کمترین تعداد گام هایی که یک پیغام دریافتی برای رسیدن به این ماثول طی کرده است را نشان می دهد.

• تابع : `getMax()` بیشترین تعداد گام هایی که یک پیغام دریافتی برای رسیدن به این ماثول طی کرده است را نشان می دهد.

• تابع : `getMean()` متوسط تعداد گام هایی که همه پیغام های دریافت شده توسط این ماثول، طی کرده اند را نشان می دهد.

• تابع : `getStddev()` انحراف از معیار تعداد گام های طی شده توسط پیغام های دریافتی این ماثول را نشان می دهد.

در تابع `finish` سپس مقدار متغیرهای `numSent` و `numReceived` به وسیله تابع `recordScalar` ذخیره شده اند و در خط بعد داده های آماری شی `hopCountStats` از طریق تابع `recordAs` با عنوان "hop" در فایل `Tictoc15-0.sca` ذخیره می گردند. از آنجا که فایل اسکالر یک فایل متنی است می توان به راحتی محتویات آن را مشاهده کرد. برای نمونه داده های نشان داده شده در زیر که بخشی از فایل `Tictoc15-0.sca` هستند داده های آماری مرتبط با `[2]` را پس گذشت زمان ۱۰۰ ثانیه نشان می دهد. این ماثول ۱۱ پیغام دریافت و ارسال کرده است. میانگین تعداد گام هایی که همه پیغام های دریافت شده توسط این ماثول، طی کرده اند $\frac{8}{636363}$ است. انحراف از معیار تعداد گام های طی شده توسط پیغام های دریافتی این ماثول حدودا $\frac{5}{463931}$ است و مجموع تعداد گام های ۱۱ پیامی که توسط این ماثول دریافت شده است ۹۵ می باشد. کمترین تعداد گام هایی که یک پیغام برای رسیدن به این ماثول طی کرده است ۱ گام می باشد و بیشترین گام طی شده ۱۷ است.

```

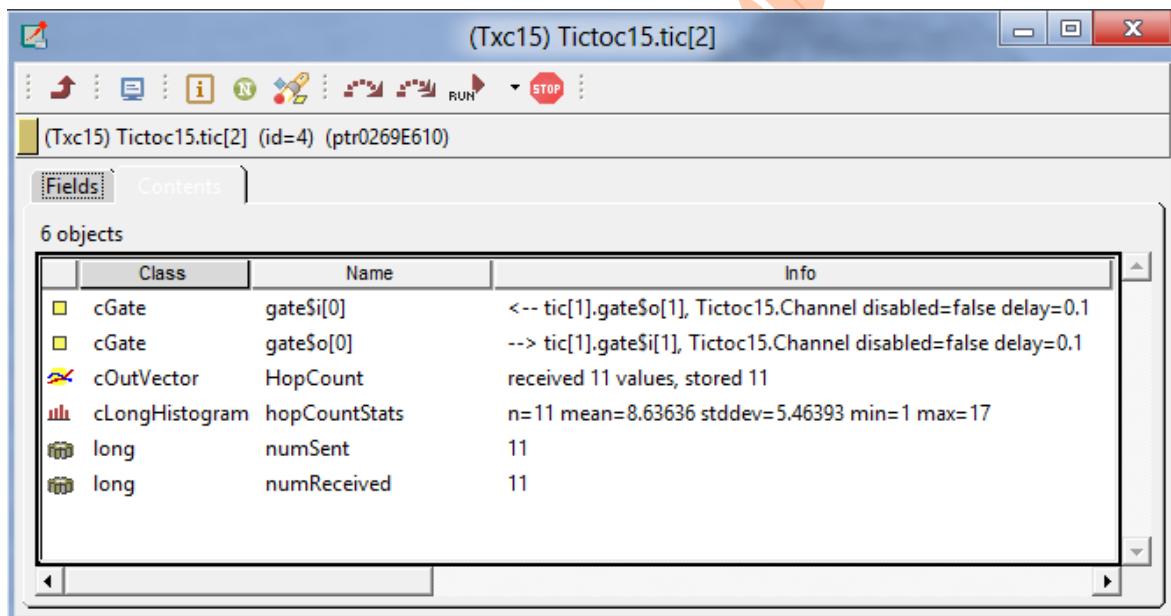
scalar Tictoc15.tic[2] #sent    11
scalar Tictoc15.tic[2] #received   11
statistic Tictoc15.tic[2]    "hop count"
field count 11
field mean 8.63636363636
field stddev 5.4639313186153
field sum 95
field sqrsum 1119
field min 1
field max 17

```

شکل ۲۱-۲ اطلاعات تابع finish() (ب)

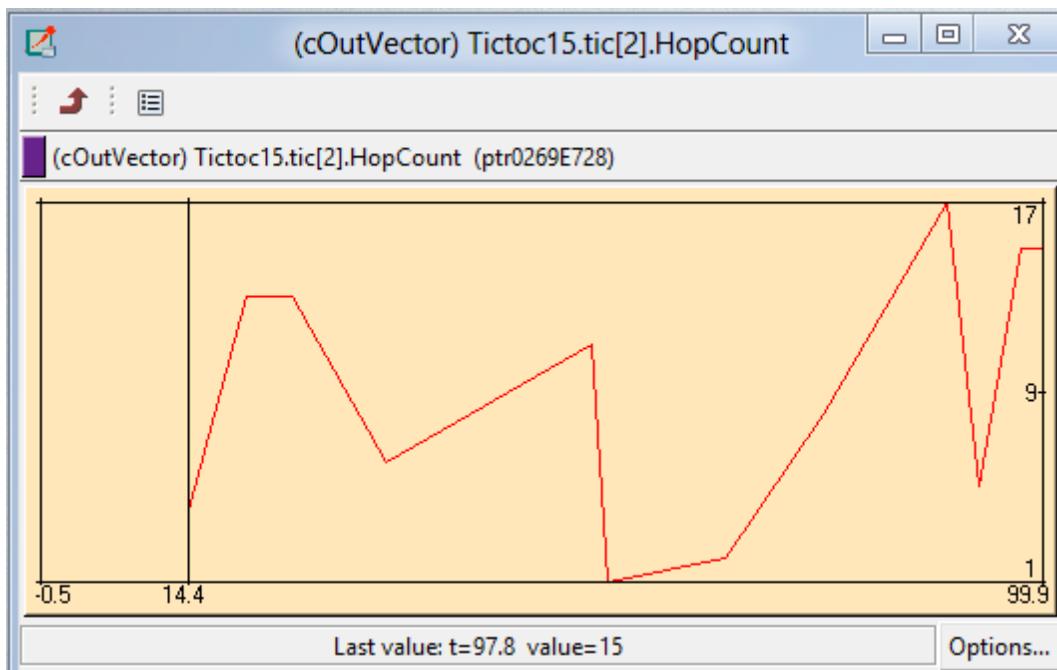
۶-۱۷-۲ نمایش داده در زمان اجرای شبیه‌سازی

داده‌ها را می‌توانید در زمان شبیه‌سازی نیز مشاهده کنید. برای این کار باید بر روی مazzoول دلخواه خود در زمان شبیه‌سازی دابل کلیک کرده تا پنجره‌ی module inspector باز شود. برای مثال بر آیکن [2] tic کلیک کرده تا پنجره‌ی زیر باز شود.



شکل ۲۲-۲ نمایش داده در زمان اجرای شبیه‌سازی

حال می‌توانید شیوه‌های hopCount و hopCountStats را پیدا کرده و با دابل کلیک بر روی آنها، آنها را باز نمایید. در ابتدا این شیوه‌ها مقداری ندارند اما در صورتی که شبیه‌سازی را با سرعت زیاد اجرا نمایید سریعاً داده‌های زیادی جمع آوری می‌گردد و تصویری مانند زیر مشاهده خواهید کرد:



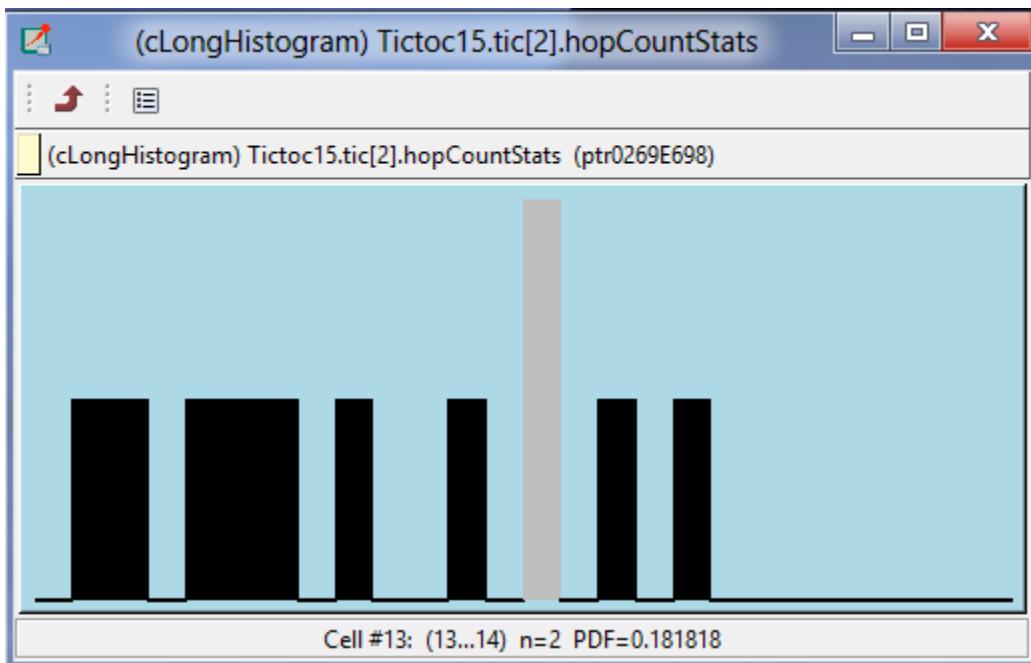
شکل ۲۳-۲ نمودار اطلاعات ذخیره شده در cOutVector

این نمودار که برای tic[2] رسم شده است در محور افقی خود نشان دهنده زمان و در محور عمودی نشان دهنده مقدار می باشد که مقدار در اینجا همان تعداد گام است. همان گونه که مشاهده می کنید زمان این نمودار از $0/5 - 0/9$ شروع گشته است (می توانید با resize نمودن پنجره این مقدار را بر روی صفر تنظیم کنید) و تا $99/9$ ادامه پیدا کرده زیرا مدت زمان شبیه سازی 100 ثانیه بود.. نمودار که با رنگ قرمز نشان داده شده است دارای نقطه ماکزیمم 17 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 گام های طی شده یک پیغام برای رسیدن به مأذول [tic[2] می باشد. همچنین تعداد خط های این نمودار که نقاط نمودار را به یکدیگر متصل می کند 11 عدد می باشد که دوباره با کمی توجه، متوجه خواهید شد این نیز همان تعداد کل پیام های دریافتی توسط tic[2] می باشد.

در ادامه اطلاعاتی که در فایل Tictoc15-0.sca دیده می شود آن را به صورت متنی باز نمایید، مجموعه داده هایی با عنوان attr type int است:

```
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 فراوانی تعداد گام های طی شده توسط پیغام ها برای رسیدن به مقصد می باشند. برای مثال اطلاعات تصویر بالا که مربوط به مازول tic[2] است نشان می دهد دو پیغام با طی کردن ۱۳ گام به این مازول رسیده اند. همچنین در صورتی که تمام اعداد ستون سوم را با هم جمع بزنید برابر خواهد بود با تعداد کل پیغام های دریافتی (یعنی ۱۱) توسط این مازول در زمان اجرای شبیه سازی و همچنین در پایان آن می توانید با کلیک بر روی hopCountStats در تب Contents مربوط به هر مازول نمودار توزیع فراوانی تعداد گام های پیام های رسیده به آن مازول را مشاهده نمایید. برای مثال شکل زیر نمودار توزیع فراوانی پیام های رسیده به tic[2] است که در بالا اطلاعات مربوط به آن بررسی شد.



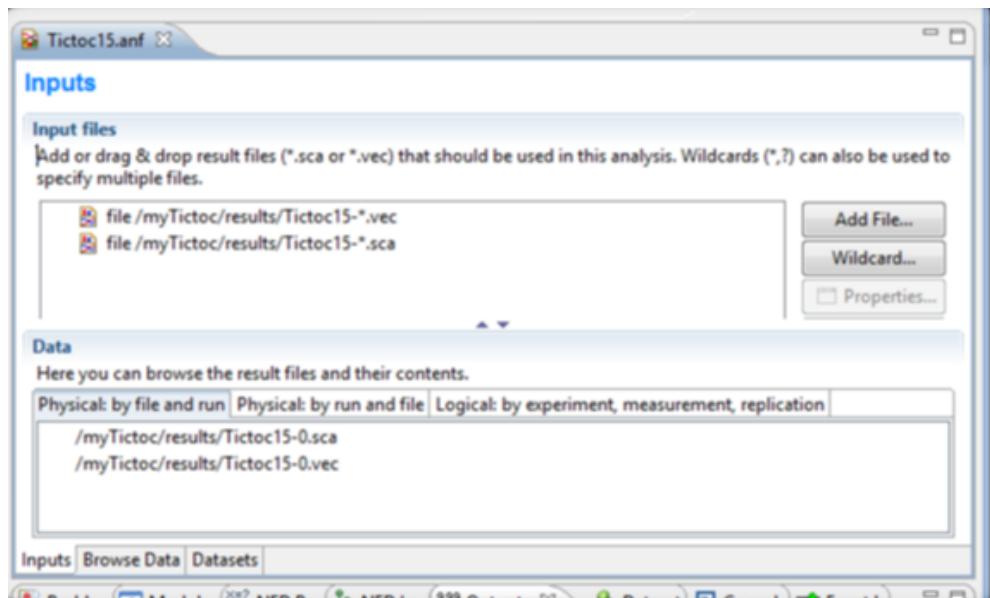
شکل ۲۴-۲ نمودار اطلاعات ذخیره شده در cLongHistogram

این نمودار دارای ۲۱ ستون است که ۹ ستون آن دارای ارتفاع ۱ و یک ستون آن که همان ستون ۱۳ است دارای ارتفاع ۲ است و دیگر ستون‌ها دارای ارتفاع ۰ می‌باشند. در شکل اشاره‌گر را بر روی ستون ۱۳ بردۀ این تا اطلاعات مربوط به آن در نوار زیر نمودار نشان داده شود. n نشان‌دهندهٔ ارتفاع ستون است.

۱۸-۲- نمایش ویژوال خروجی بردارها و اعداد اسکالر بدست آمده

OMNeT++ IDE می‌تواند با پشتیبانی از فیلتر، پردازش و نمایش داده‌های بردار و اسکالر و همچنین نمایش نمودارها، به شما در تحلیل نتایج خود کمک زیادی نماید. نمودارهای که در ادامه معرفی می‌شوند همگی با کمک ابزار Result Analysis در این IDE ایجاد شده‌اند. برای استفاده از امکانات IDE باید پس از اتمام شبیه‌سازی به پوشه results رفته و بر روی فایل *.vec و یا *.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 پس از کلیک بر روی یکی از این دو فایل نمای زیر را خواهیم دید:



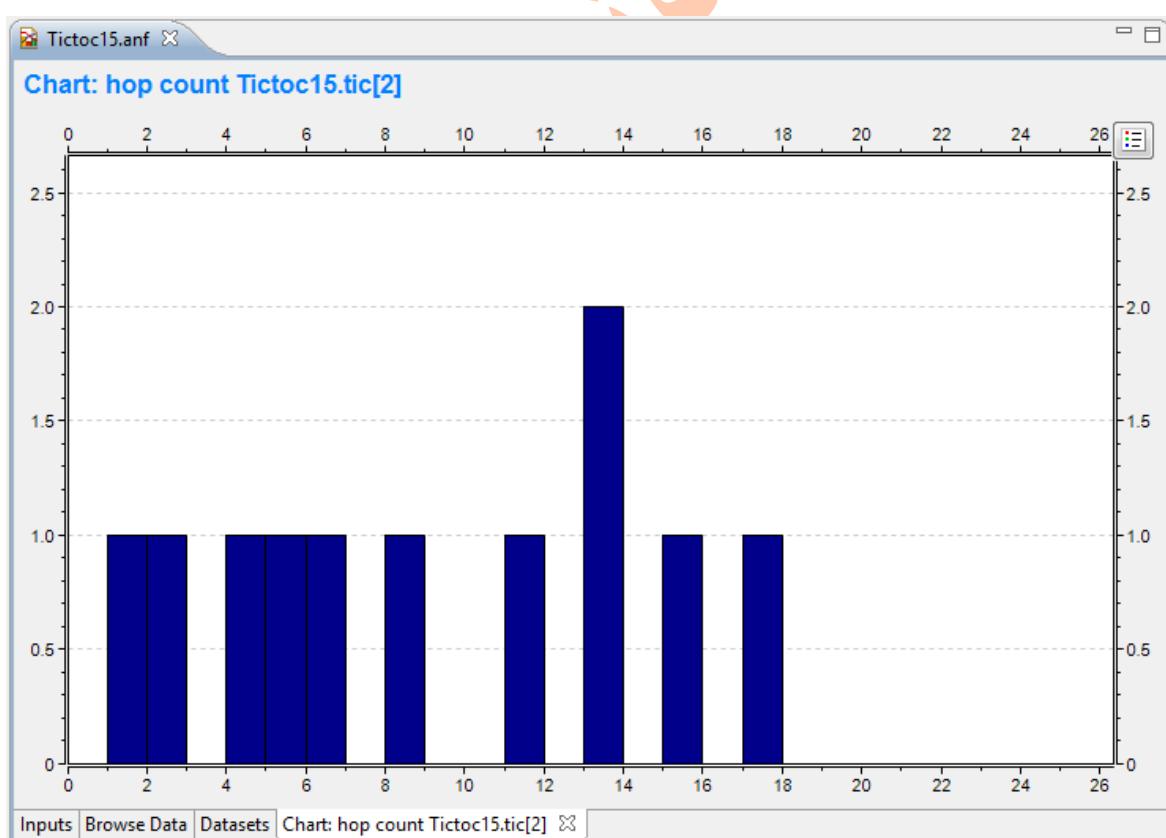
شکل ۲۵-۲ مراحل اجرای خروجی بردارها و اسکالرها بدست آمده (الف)

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

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 counts: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]	

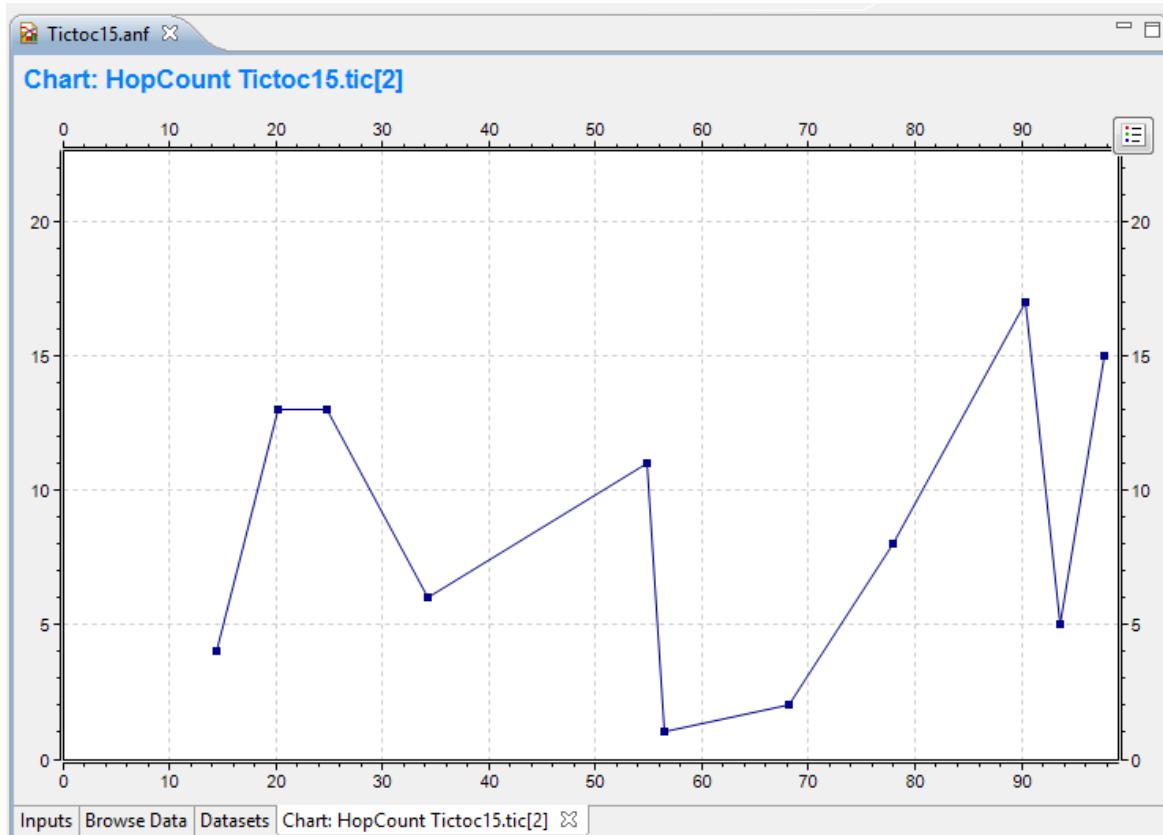
شکل ۲۶-۲ مراحل اجرای خروجی بردارها و اسکالرها بدست آمده (ب)

این تب شامل چهار تب All ، Vectors و Scalars می شود. در تب اول یعنی All همانطور که در بالا مشاهده می کنید تمام داده های آماری اعم از وکتورها و اسکالرهای تمام مازول های موجود در شبکه آورده شده است. می توانید این داده ها را که در تصویر بالا برای tic[2] با داده هایی که در بخش قبل برای همین مازول نشان داده شد، مقایسه کنید. در این صورت متوجه یکسان بودن این داده ها خواهید شد. حتی در اینجا اطلاعات بیشتری در رابطه با مازول tic[2] آورده شده است. این داده ها علاوه بر پیغام های رسیده به مازول می باشند. برای مثال در قسمت HopCount (شماره آخرین رخداد اتفاق افتاده در این مازول) که ۹۷۹ است و شماره اولین رویداد رخ داده شده در آن که برابر ۱۴۵ است دیده می شود. اگر کمی در نمودار توالی جستجو کنید خواهید دید اولین رویدادی که در tic[2] شروع می شود برای ارسال پیام tic2-to-0 است که در لحظه ۱۴۵ اتفاق افتاده است و همان گونه که از نام آن پیداست پیغام باید به tic[0] ارسال می گشت. همچنین از دیگر اطلاعاتی که می توان در اینجا بدان دست یافت واریانس تعداد گام های طی شده توسط پیغام ها برای رسیدن به tic[2] است که این مقدار با توجه به تصویر برابر $\frac{29}{45}$ است. برای دیدن نمودار فراوانی تعداد گام های طی شده (گفته شده در قسمت قبل) می توانید بر روی hop count(histogram) کلیک کنید. که در این زیر این نمودار برای tic[2] نشان داده شده است:



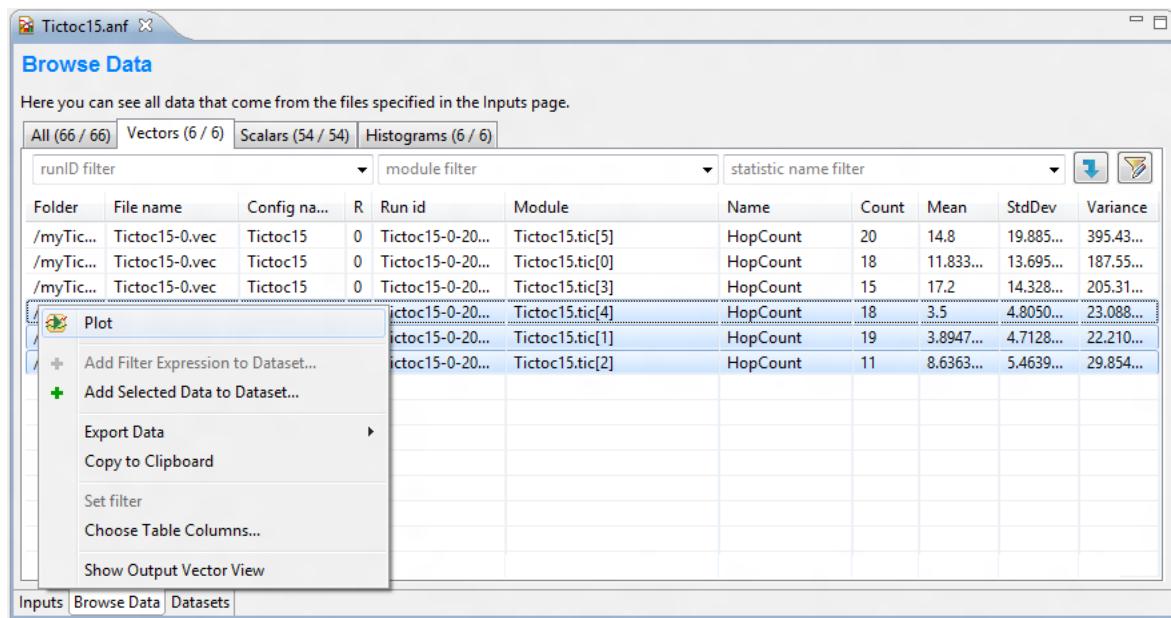
شکل ۲۷-۲ مراحل اجرای خروجی بردارها و اسکالرهای بدست آمده (ج)

همچنین برای دیدن نمودار خطی مشابه آنچه در قسمت قبلی گفته شد می‌توان بر روی کلیک کرد. این نمودار برای [tic[2] بدين شکل می‌باشد.



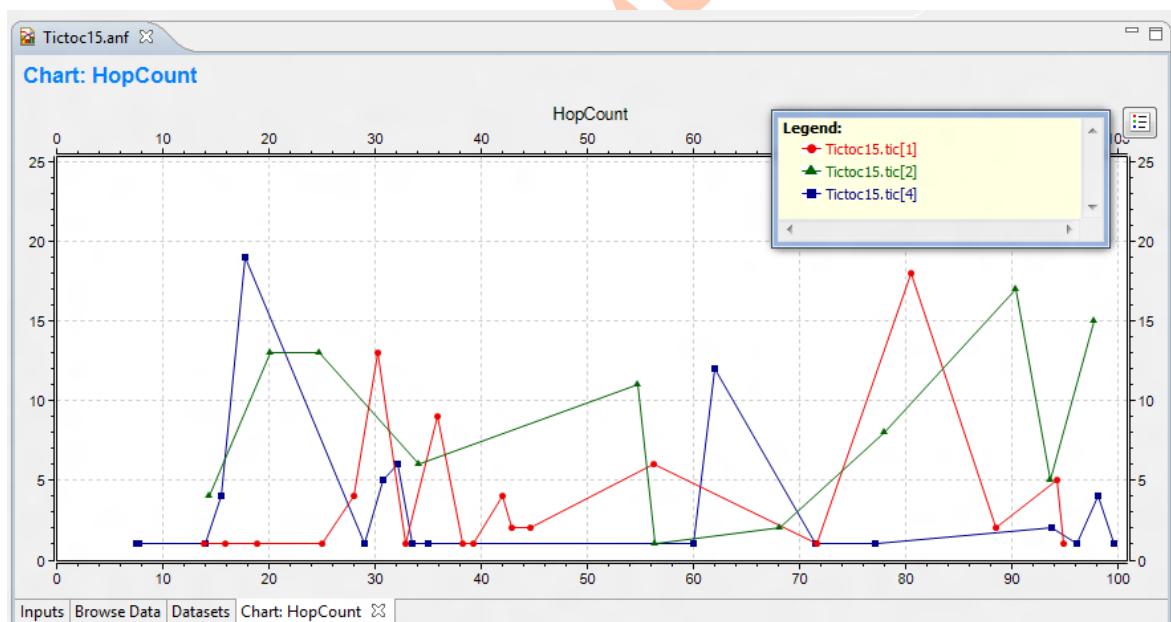
شکل ۲۸-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (د)

می‌توانید 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 می‌کنم ابزارها و امکانات بیشتری را معرفی کنیم. برای مقایسه بردارهای خروجی چندین ماثول می‌توانید به تب Browse Data رفته و هر تعداد ماثول می‌خواهید 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 Plot را انتخاب کنید تا نمودارهای هر یک رسم شود.



شکل ۲۹-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (۵)

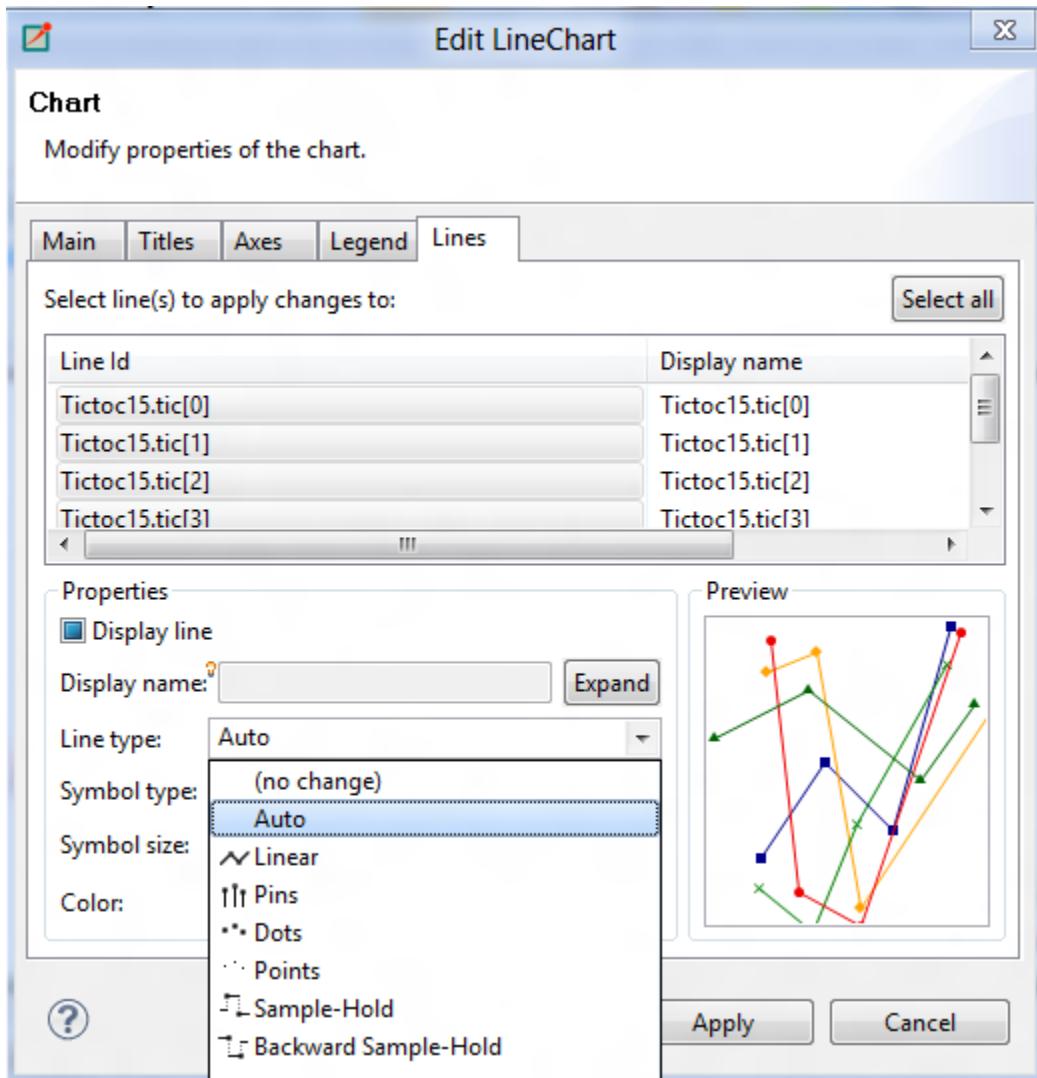
نمودار خطی ایجاد شده برای سه مازول tic[4] و tic[1] و tic[2] به شکل زیر می باشد:



شکل ۳۰-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (و)

با استفاده از اینگونه نمودارها می توان براحتی و در کمترین زمان مقایسه ای بین چندین مازول انجام داد. برای مثال از این نمودار متوجه می شویم بیشترین گام های طی شده توسط پیغام ها متعلق به پیغامی است که به مازول tic[4] رسیده است و حدود ۱۹ گام می باشد.

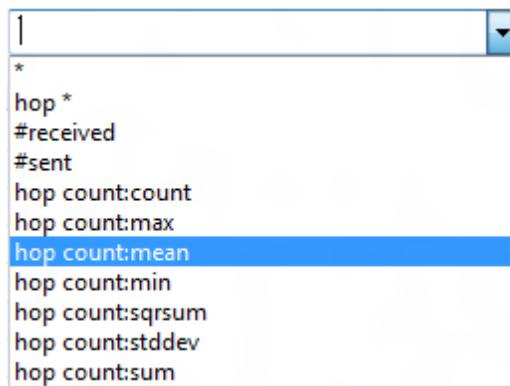
از دیگر قابلیت های این نمودار امکان تغییر نوع دید آن می باشد. برای این کار بر روی نمودار کلیک راست کرده و... properties را انتخاب کنید. از پنجره باز شده تب Lines را کلیک کنید. سپس در بخش Line type بر روی properties کلیک کنید تا لیست نوع نمودارهای موجود باز شود:



شکل ۳۱-۲ مراحل اجرای خروجی بردارها و اسکالرهای بدست آمده (۴)

داده های اسکالر از مازول 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 مازول 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 می باشد. هر بار اجرای شبیه سازی یک آی دی منحصر به فرد دارد که با آن می توان اجرای مورد نظر خود را مشخص کرد، برای مثال می خواهیم متوسط تعداد گامهای طی شده پیام ها رسیده به هریک از مازول ها را با هم مقایسه کنیم. در اینجا یک راه غیر هوشمندانه پیدا کردن Mean مربوط

به هریک از مازولها 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 مازولهای شبکه را در مثال Tic toc به راحتی می توان پیدا کرد اما موارد بسیاری وجود دارد که مازولهای 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 static name filter کلیک کرده و سپس hop count:mean را انتخاب نمایید.



شکل ۳۲-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (ر)

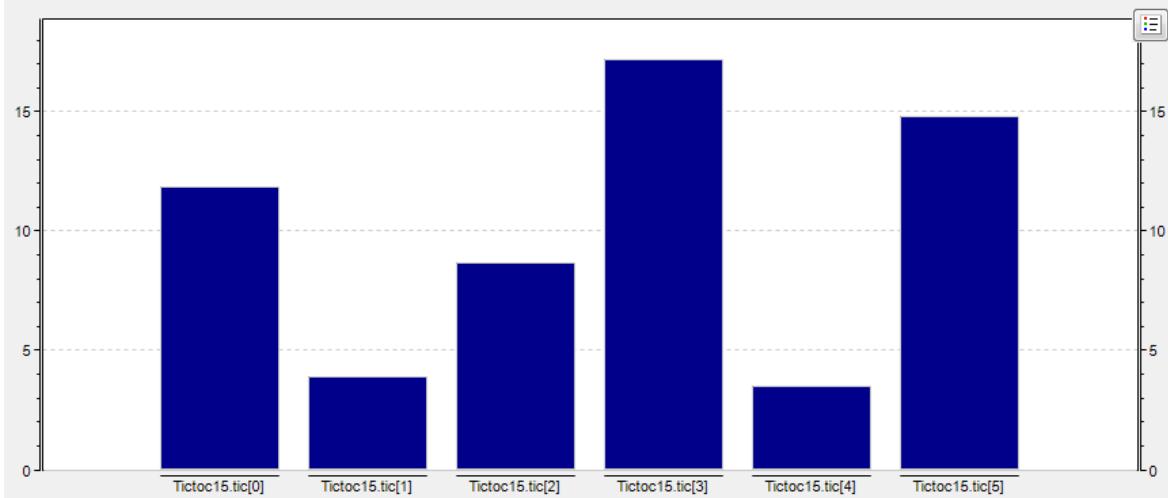
سپس همه مازولها را انتخاب کرده و کلیک راست نمایید و گزینه Plot را برگزینید:

runID filter	module filter	hop count:mean					
Folder	File name	Config na...	R	Run id	Module	Name	Value
Plot	Tictoc15-0-20...	Tictoc15.tic[0]			hop count:me...	11.8333333...	
	Tictoc15-0-20...	Tictoc15.tic[1]			hop count:me...	3.89473684...	
	Tictoc15-0-20...	Tictoc15.tic[2]			hop count:me...	8.63636363...	
	Tictoc15-0-20...	Tictoc15.tic[3]			hop count:me...	17.2	
	Tictoc15-0-20...	Tictoc15.tic[4]			hop count:me...	3.5	
	Tictoc15-0-20...	Tictoc15.tic[5]			hop count:me...	14.8	

شکل ۳۳-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (ح)

چارت نمایش داده شده به شکل زیر خواهد بود:

Chart: hop count:mean



شکل ۳۴-۲ مراحل اجرای خروجی بردارها و اسکالارهای بدست آمده (ط)

با استفاده از این نمودار می توانید به راحتی دریابید که بیشترین متوسط تعداد گامهای طی شده توسط پیغام ها ۱۷ گام می باشد که برای رسیدن به tic[3] طی کرده اند. همچنین کمترین متوسط گام طی شده یعنی ۳ گام مربوط به پیغام های رسیده به tic[4] است.

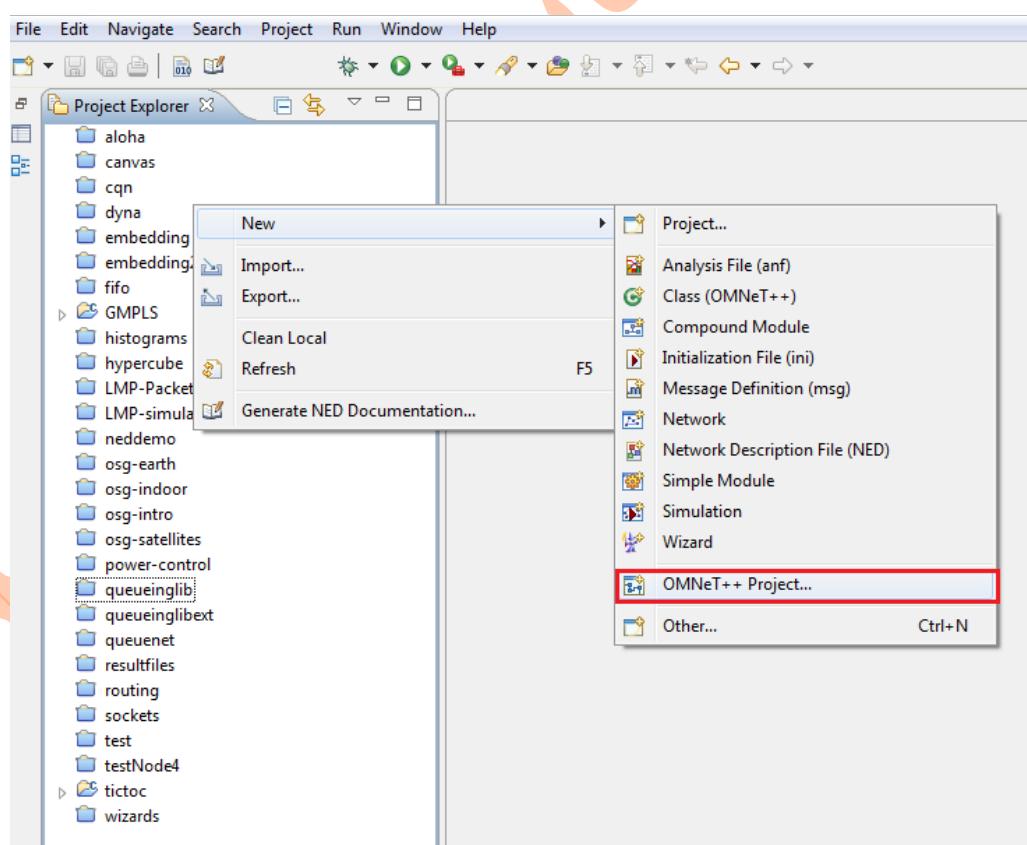
فصل ۳ - ایجاد پروژه و ذخیره آن

۱-۳ - مقدمه

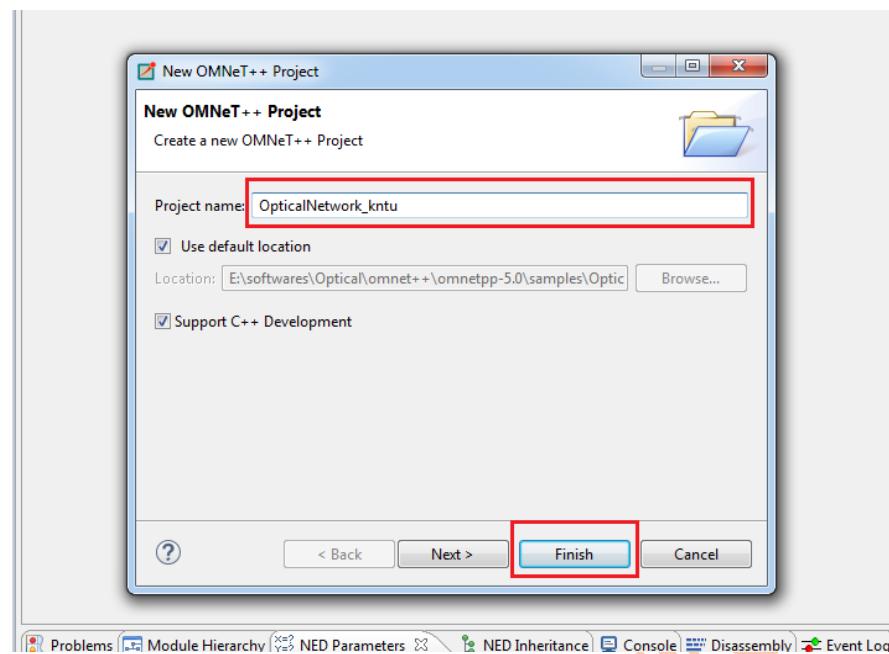
تا به اینجا تا حدودی با نحوه کار شبیه ساز OMNeT++ آشنا شدیم و با آن یک شبکه فرضی را شبیه سازی کردیم. در اینجا یک تمرین ساده عنوان شده است که هدف از آن آشنایی هرچه بیشتر خواننده با محیط نرم افزار و استفاده از مطالبی است که در فصل دوم بیان شدند.

۲-۳ - نحوه ایجاد یک پروژه

در شکل های زیر نحوه ایجاد کردن یک پروژه آورده شده است. ابتدا لازم است که در Project Explorer یک پروژه جدید با اسم دلخواه انتخاب کنید.

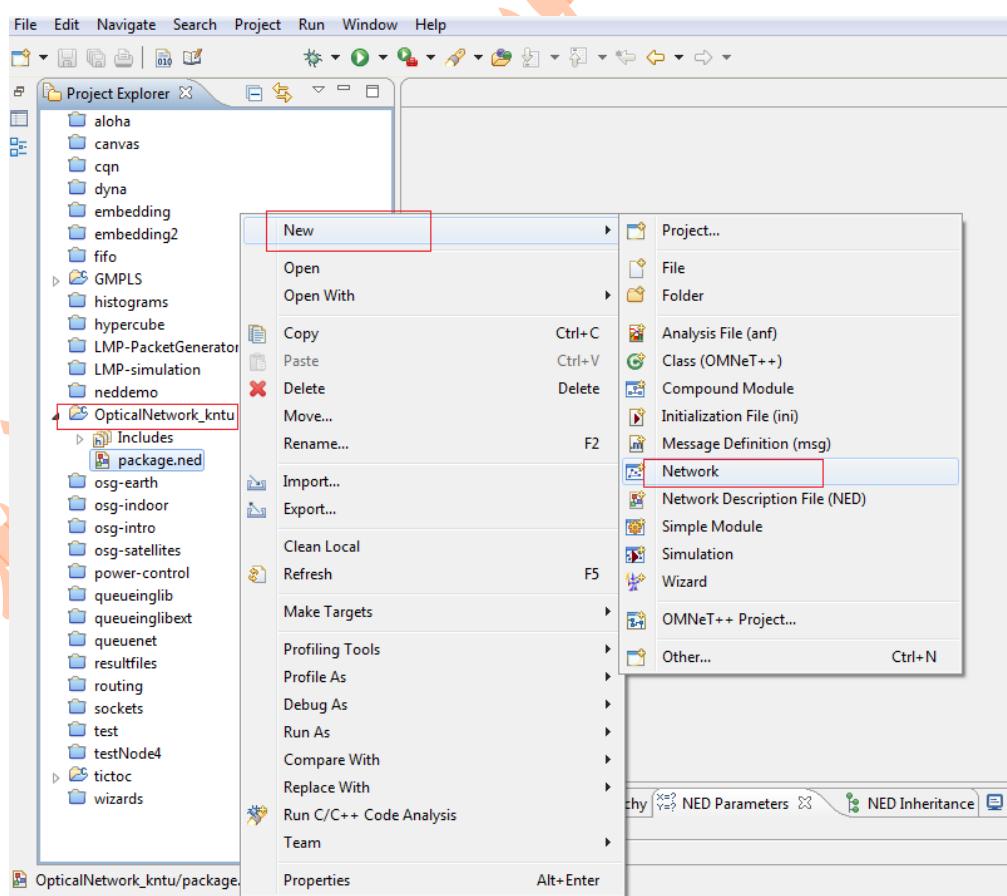


شکل ۱-۳ نحوه ایجاد یک پروژه (الف)

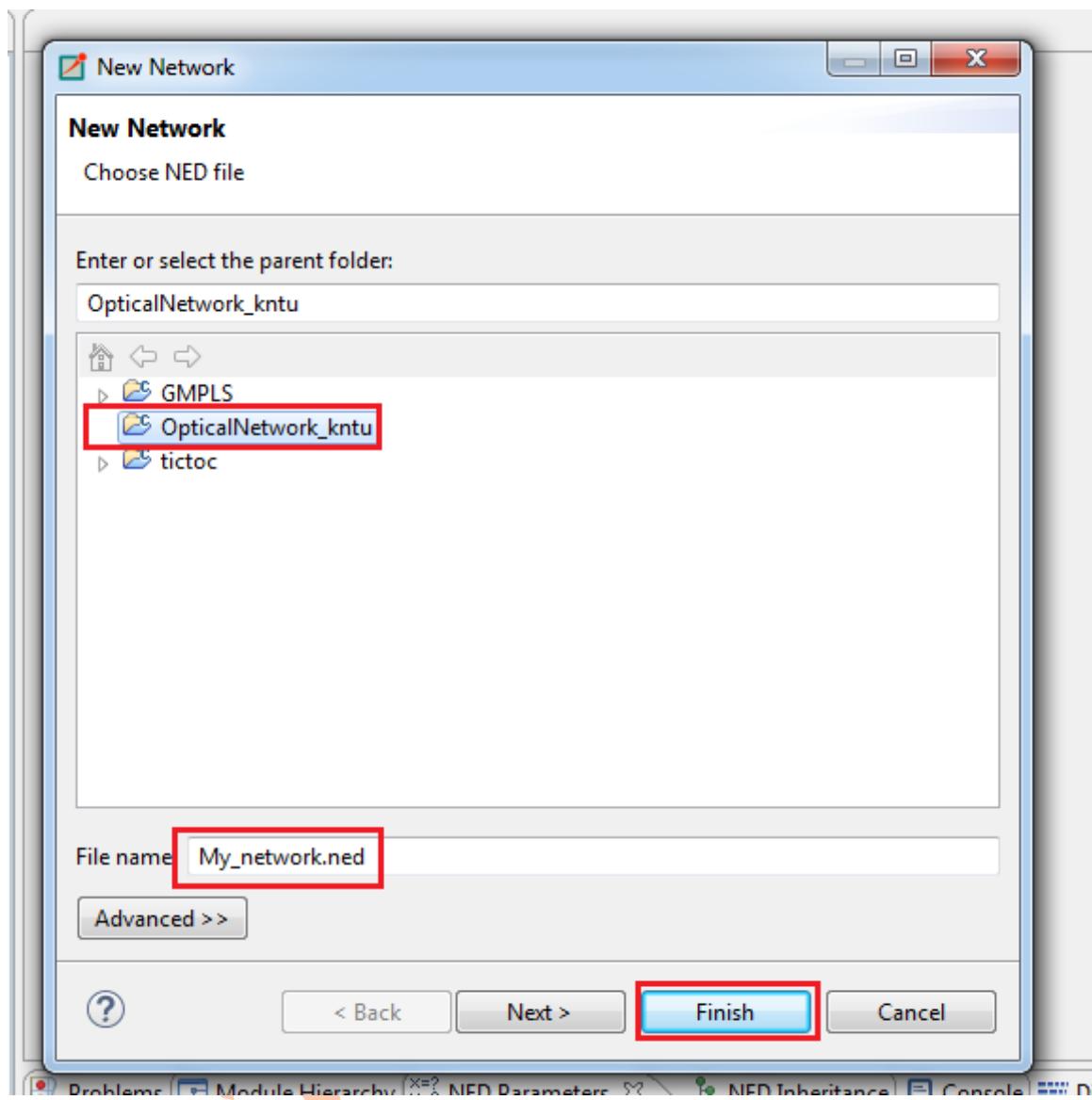


شکل ۲-۳ نحوه ایجاد یک پروژه (ب)

در پروژه ایجاد شده باید اجزای مختلف شبکه را اضافه کنید. اولین مورد تعریف شبکه یا همان ned. می باشد که در شکل زیر نحوه تعریف آن را مشاهده می کنید.

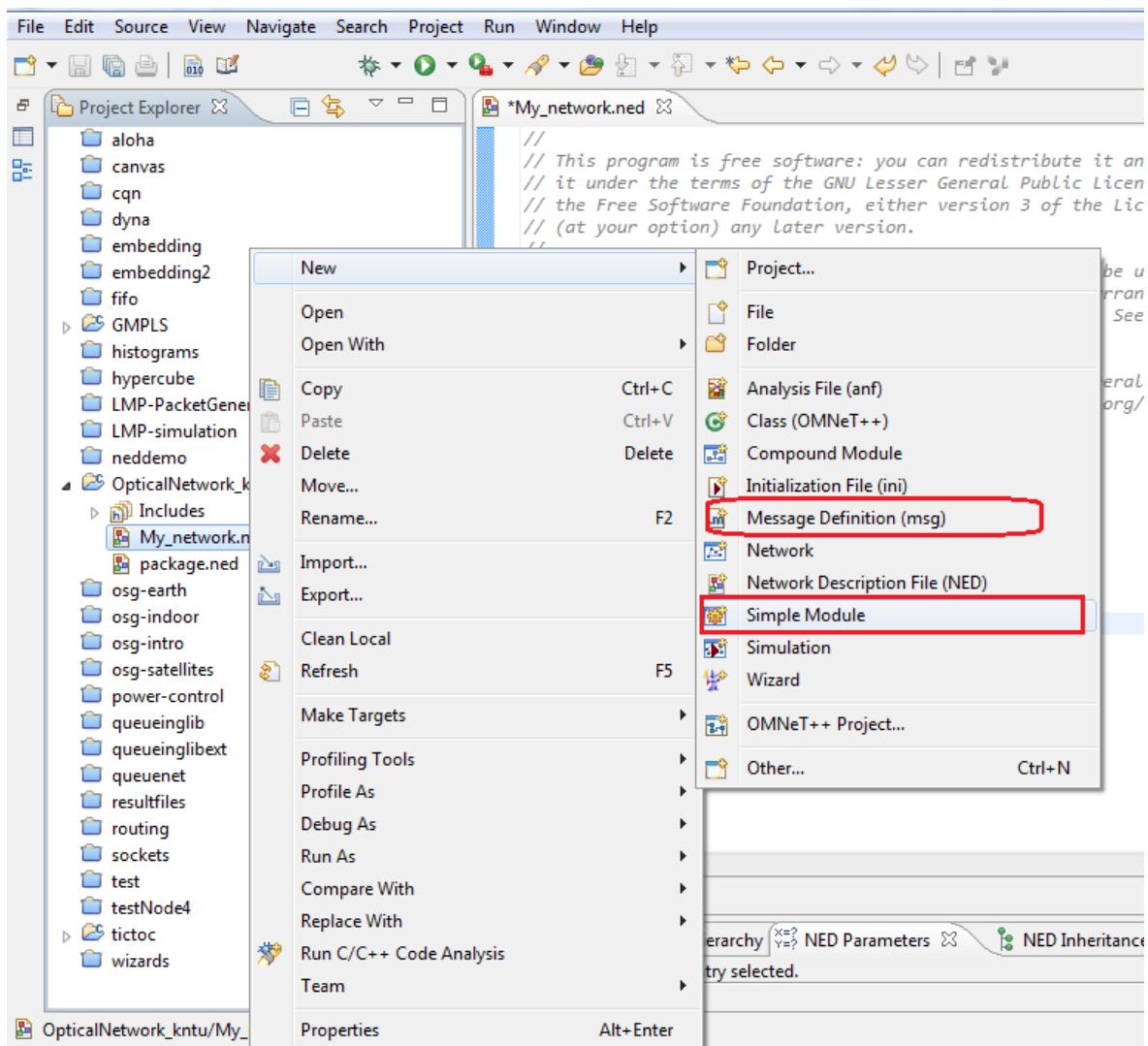


شکل ۳-۳ نحوه ایجاد یک پروژه (ج)



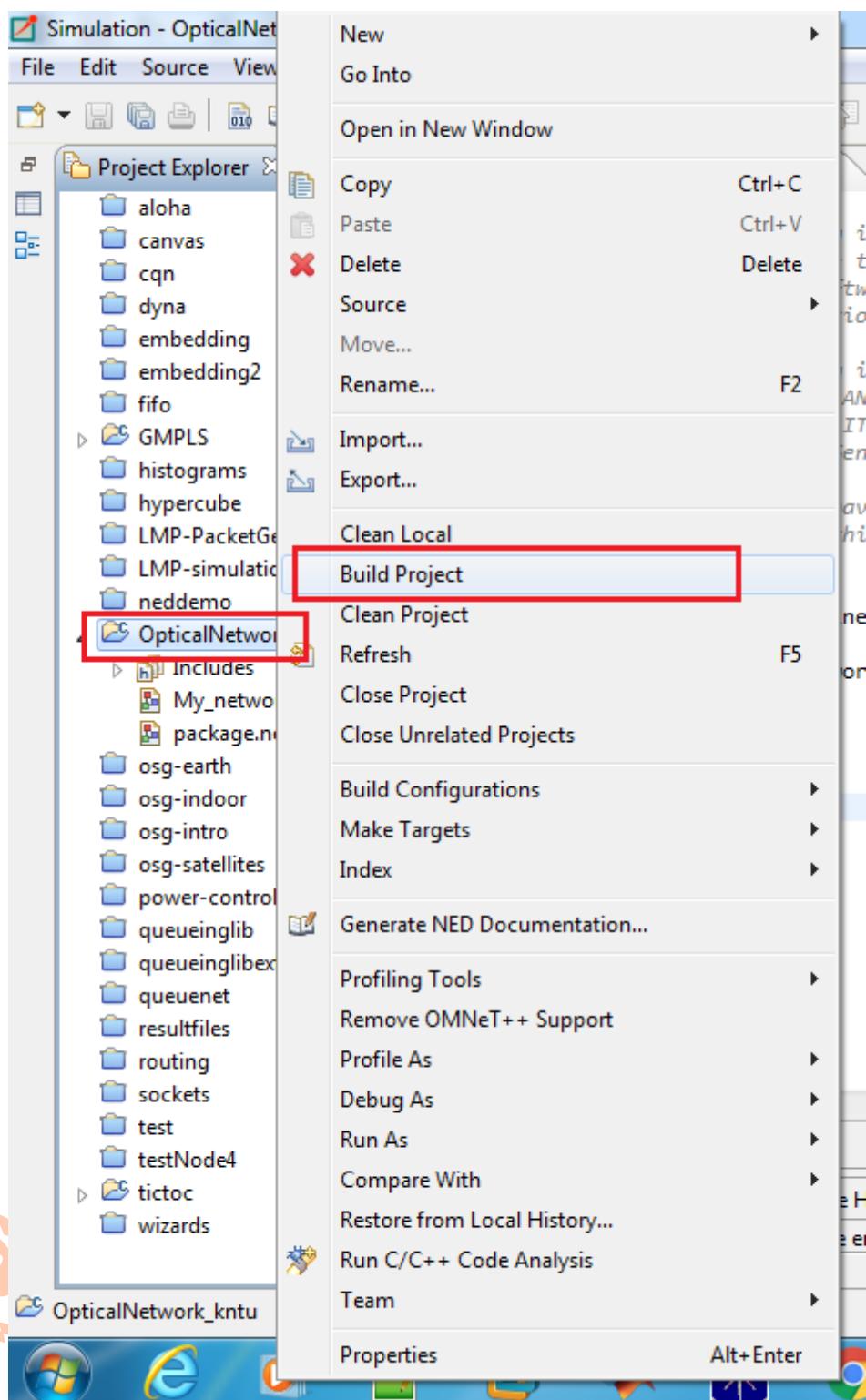
شکل ۴-۳ نحوه ایجاد یک پروژه (د)

پس از تعریف ned. و انتخاب اسم برای آن نیاز به تعریف simple Module (یا Compound Module) دارید که در فصل قبل به تفصیل بررسی شدند. تعریف msg. هم به همین صورت خواهد بود.



شکل ۵-۳ نحوه ایجاد یک پروژه (۵)

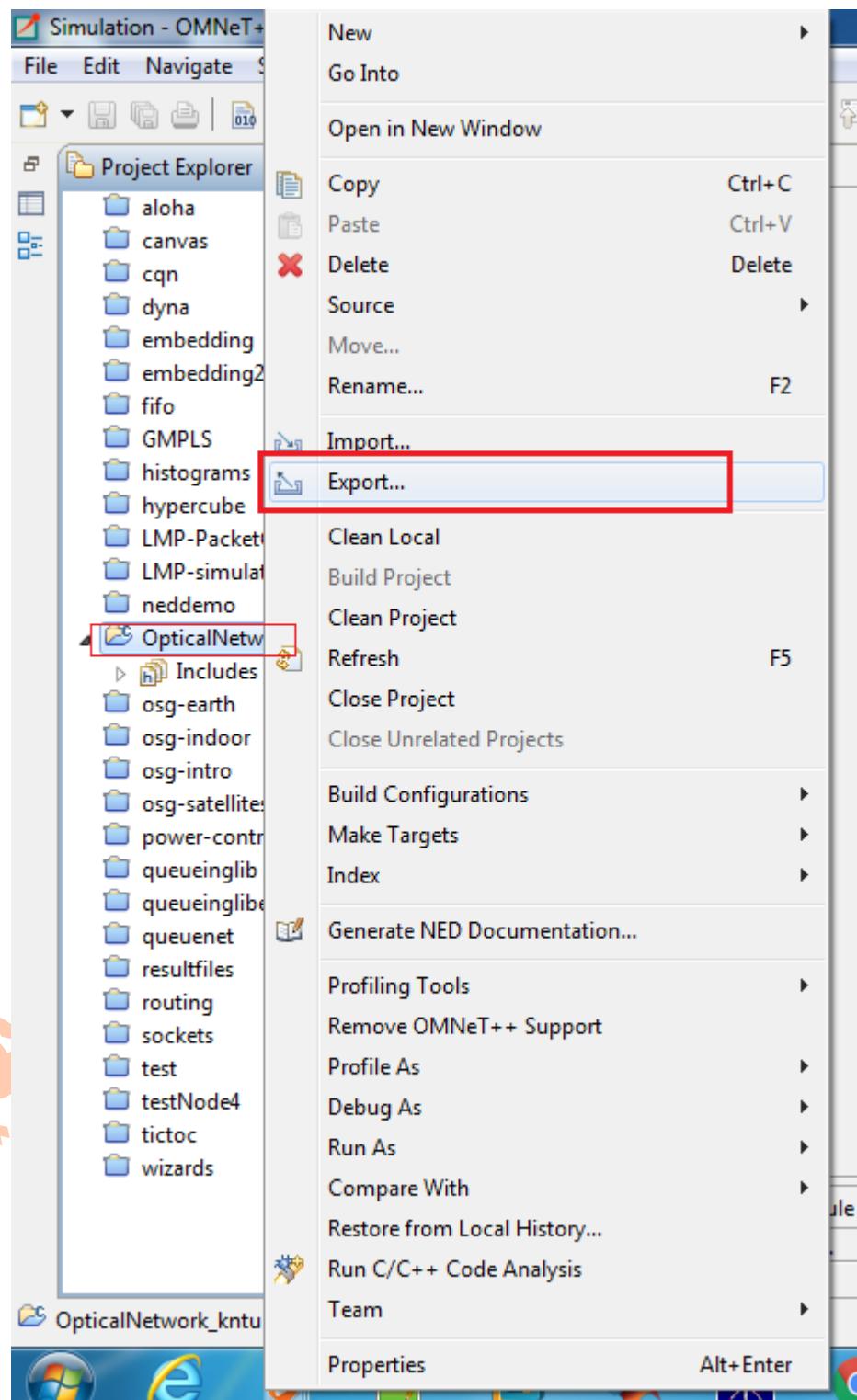
نکته قابل توجه این است که در هر بار تغییر در قسمت‌های شبکه پس از ذخیره سازی باید کل پروژه تعریف شده را Build کرد که در زیر نحوه انجام دادن این کار را مشاهده می‌کنید.



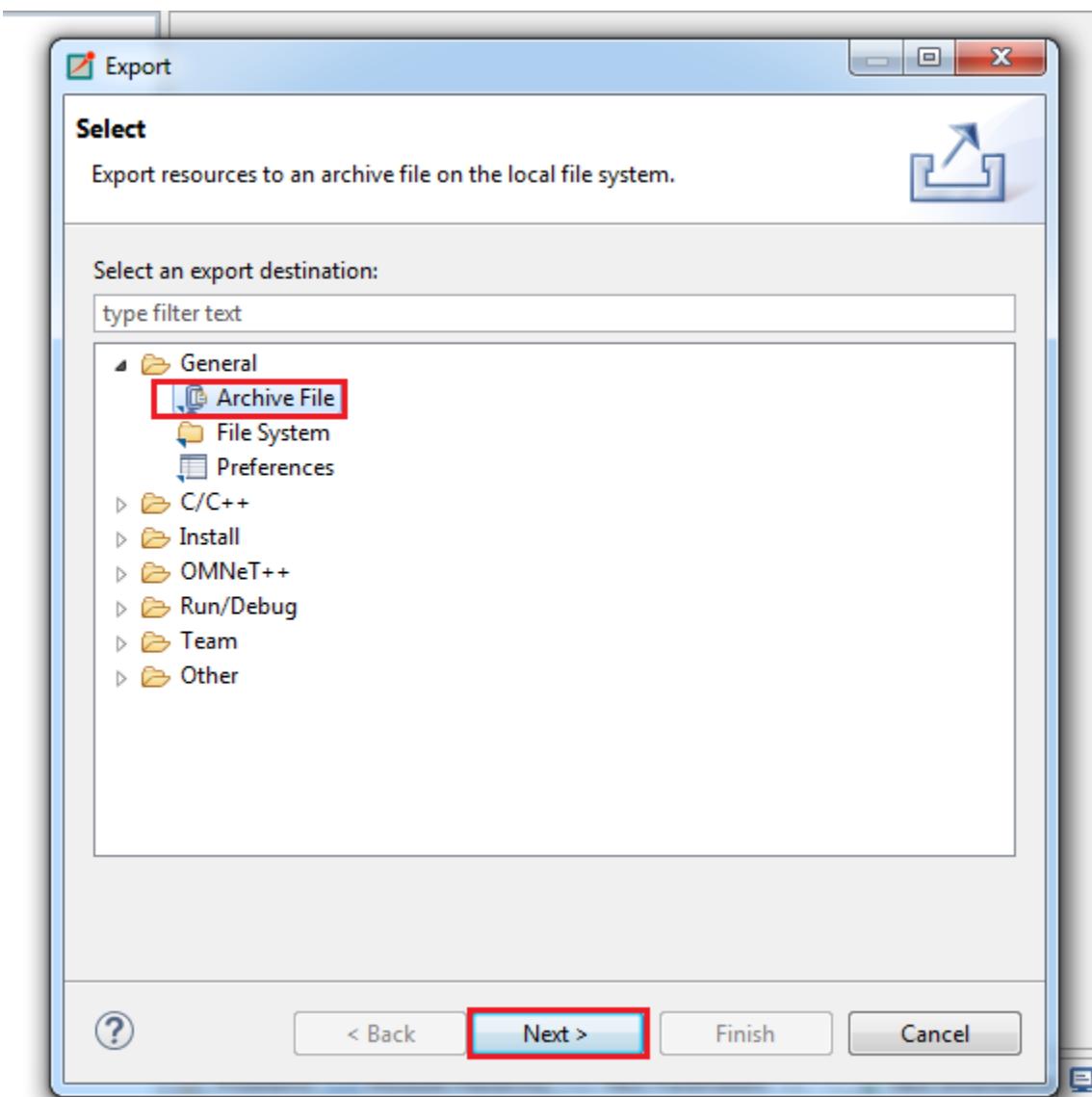
شکل ۶-۳ نحوه Build کردن یک پروژه

۳-۳- نحوه Export کردن یک پروژه

در این بخش نحوه Export کردن یک پروژه توضیح داده شده است.

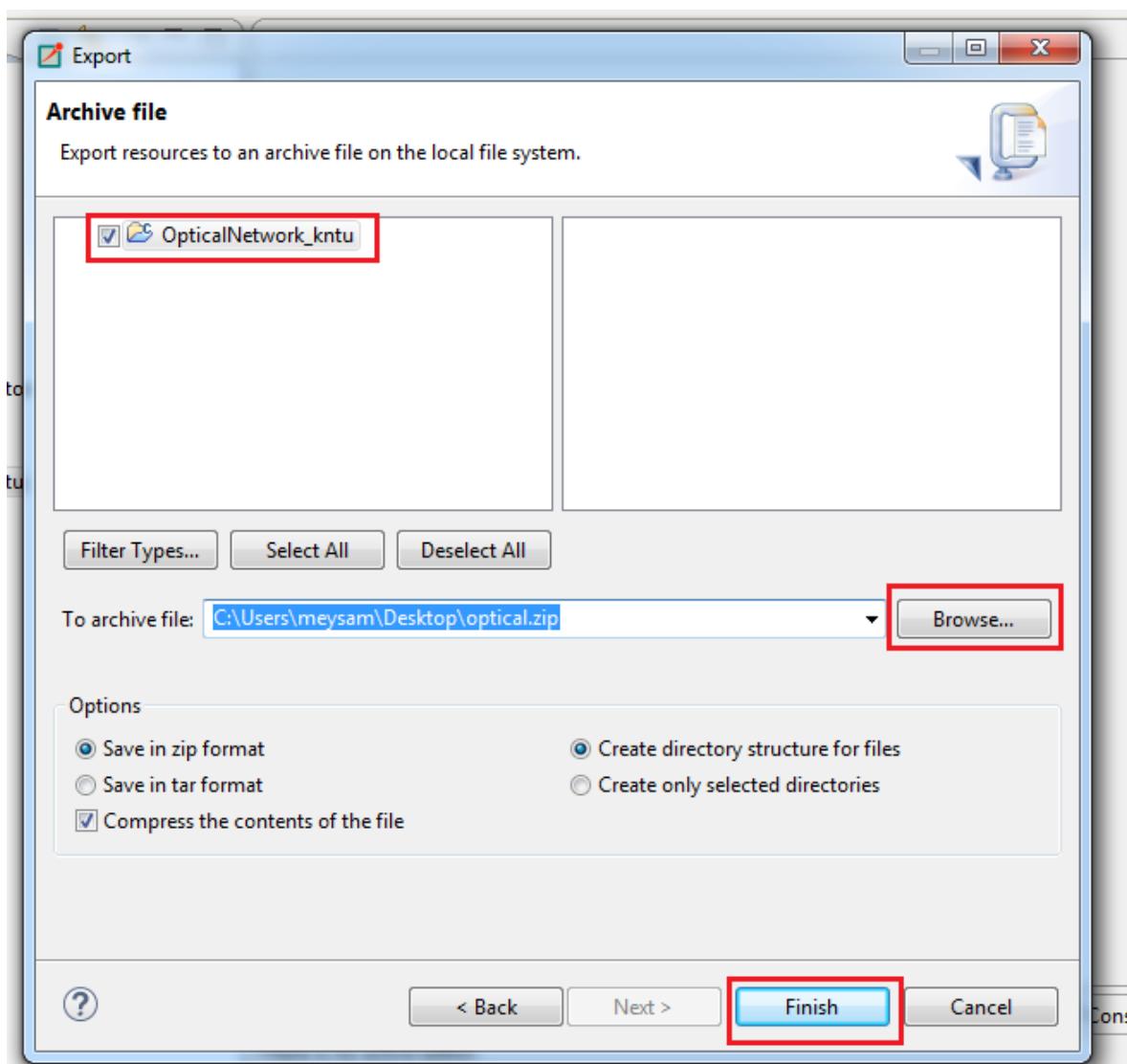


شکل ۷-۳- نحوه Export کردن یک پروژه (الف)



شکل ۸-۳ نحوه Export کردن یک پروژه (ب)

در قسمت Browse آدرس محلی را که می خواهید پروژه در آن ذخیر شود انتخاب کنید.



شکل ۹-۳ نحوه Export کردن یک پروژه (ج)

پروژه در محل تعیین شده با فرمت zip ذخیره می‌شود.