UNIVERSITY OF CAGLIARI

DIEE - Department of Electrical and Electronic Engineering

OMNeT++
Esercitazione TicToc

Dott. Ing. Cristina Desogus

Introduzione

Per avviare OMNeT++ è necessario:

- Aprire il terminale tramite mingwenv
- Digitare sul terminale **omnetpp**

Si aprirà il programma e chiederà di selezionare il workspace.

Confermare senza cambiare il workspace.

TicToc

Creiamo un nuovo progetto:

- Clic destro -> New -> Project
- Selezionare OMNeT++ -> OMNeT++ Project -> Next
- Project Name:
  - TICTOC1

Si creerà un progetto con all'interno:

- gli includes;
- la cartella simulations che conterrà i file .ned e i .ini;
- la cartella src che conterrà i file .h e .cc.

Creiamo nella cartella simulation: TICTOC1.ned

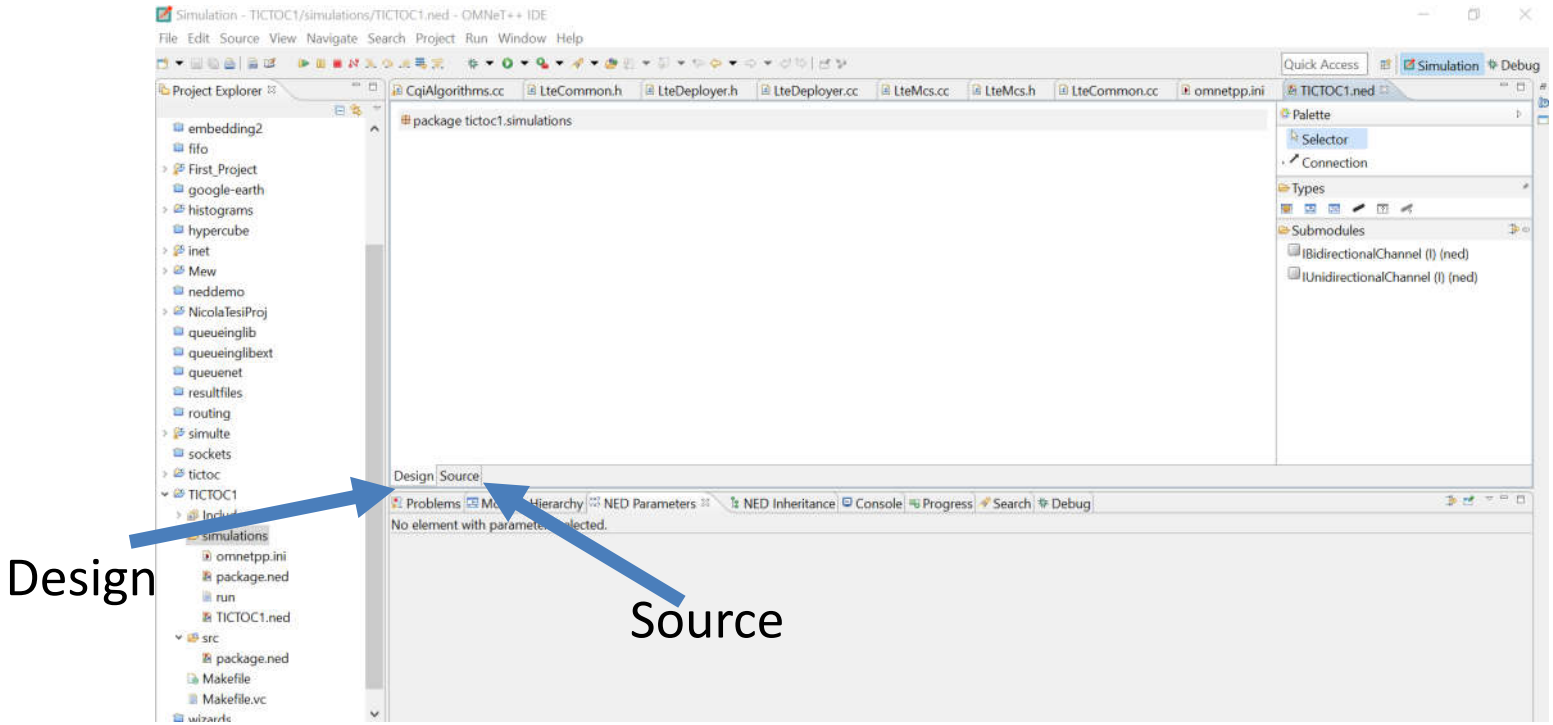
TicToc

Creiamo nella cartella simulation TICTOC1.ned:

- Clic destro nella cartella simulation -> New -> Network Description File
- File Name: TICTOC1.ned
- Empty NED file -> Finish

Si aprirà la pagina di editor di TICTOC1.ned

TicToc



MCLab

TicToc

Scriviamo nel file .ned

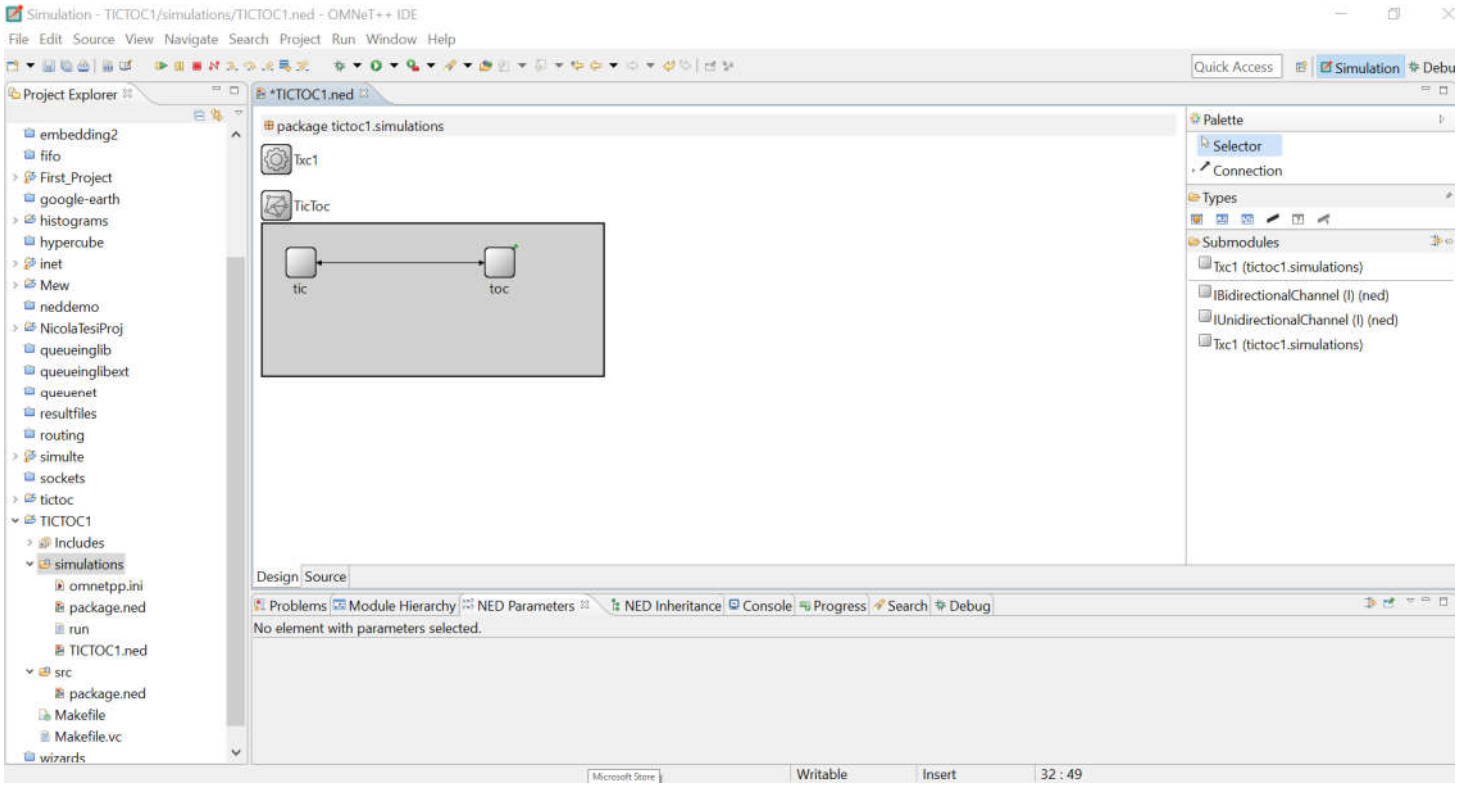
```
simple Txc1
{
    gates:
        input in;
        output out;
}
```

MCLab

TicToc

```
network Tictoc
{
        submodules:
            tic: Txc1;
            toc: Txc1;
        connections:
            tic.out --> { delay = 100ms; } --> toc.in;
            tic.in <-- { delay = 100ms; } <-- toc.out;
}
```

TicToc


Simulation - TICTOC1/simulations/TICTOC1.ned - OMNeT++ IDE
File Edit Source View Navigate Search Project Run Window Help
Quick Access
Simulation
Project Explorer
*TICTOC1.ned
package tictoc1.simulations
Txc1
TicToc
tic
toc
Palette
Selector
Connection
Types
Submodules
Txc1 (tictoc1.simulations)
IBidirectionalChannel (I) (ned)
IUnidirectionalChannel (I) (ned)
Txc1 (tictoc1.simulations)
embedding2
fifo
First_Project
google-earth
histograms
hypercube
inet
Mew
neddemo
NicolaTesiProj
queueinglib
queueinglibext
queuenet
resultfiles
routing
simulte
sockets
tictoc
TICTOC1
Includes
simulations
omnetpp.ini
package.ned
run
TICTOC1.ned
src
package.ned
Makefile
Makefile.vc
wizards
Design
Source
Problems
Module Hierarchy
NED Parameters
NED Inheritance
Console
Progress
Search
Debug
No element with parameters selected.
Writable
Insert
32 : 49

TicToc

Creiamo un file .h dentro la cartella src. Come nel caso precedente:

- Clic destro -> New -> Souce File.
- Chiamiamolo come il simple module, nel nostro esempio Txc1.h

```
#include <string.h>
#include <omnetpp.h>
class Txc1 : public cSimpleModule
{
        protected:
                virtual void initialize() override;
                virtual void handleMessage(cMessage *msg) override;
};
Define_Module(Txc1);
```

TicToc

Creiamo il file .cc con lo stesso nome dato al .h e inseriamo il riferimento al precedente file dentro un include.

#include <Txc1.h>

Per adesso conterrà solo i due metodi dichiarati nel file .h

Initialize() e handleMessage()

MCLab

TicToc

Initialize() viene chiamato all'inizio della simulazione, la cosa importante su cui focalizzarci è la creazione di un oggetto di tipo cMessage che contiene la stringa tictocMsg (che identifica il messaggio) e l'invio del messaggio sul gate out

```
void Txc1::initialize()
{
 if (strcmp("tic", getName()) == 0) {
        cMessage *msg = new cMessage("tictocMsg");
        send(msg, "out");
        }
}
```

MCLab

TicToc

Il metodo handleMessage () viene chiamato ogni volta che arriva un messaggio, in questo caso il nostro modulo, trasmette attraverso il gate out il messaggio.

```
void Txc1::handleMessage(cMessage *msg)
{
        send(msg, "out");
}
```

Poiché sia il nodo `tic ‘che il nodo` toc' fanno la stessa cosa, il messaggio rimbalzerà tra i due nodi.

TicToc

Affinché la nostra simulazione possa partire è necessaria la presenza di un file di inizializzazione.

In generale è possibile creare il file .ini come nei casi precedenti. Nel nostro caso il file è stato creato in maniera automatica all'interno della cartella simulations:

- omnetpp.ini

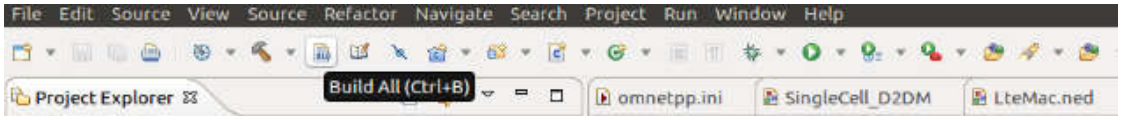
[General]

network = TicToc

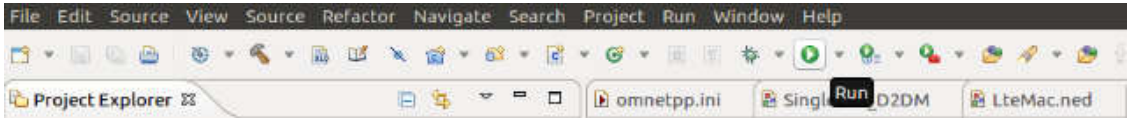
TicToc

Adesso è possibile compilare e infine lanciare il run.

Per effettuare il build:



Per effettuare il run:



TicToc

Su questa base è possibile modificare i parameters che ci permettano di gestire meglio le nostre simulazioni aggiungendo funzionalità.

In questo esempio, vogliamo che la nostra simulazione si blocchi dopo 10 steps e cancelli il messaggio alla fine degli steps.

TicToc

Nel file .h aggiungiamo una variabile.

private:

int counter;

Nel .cc. (dentro il blocco ***initialize***)

counter = 10;

WATCH(counter);

MCLab

TicToc

Dentro handleMessage

```
void Txc1::handleMessage(cMessage *msg)
{
      counter--;
      if (counter == 0) {
            delete msg;
            }
      else {
             send(msg, "out");
             }
}
```

MCLab

TicToc

I parameter possono essere definiti anche all'esterno dei file .cc

A seconda dell'utilizzo che essi hanno possono essere definiti anche nei **file .ned e nei file .ini**

MCLab

TicToc

Dentro .ned, quando definiamo il nostro simple module Txc1, aggiungiamo:

parameters:

int limit = default(2);

Dentro .ini

****.limit = 5**

Dentro .cc verrà infine utilizzato come:

counter = par("limit");

TicToc

Nell'esempio precedente, tic e toc inviavano il messaggio immediatamente.

Ci sono dei casi in cui invece è necessaria una temporalizzazione del messaggio.

In Omnet++ tale temporizzazione è ottenuta dal modulo inviando un messaggio a se stesso.

Tali messaggi sono chiamati self-message (sono a tutti gli effetti messaggi ordinari).

TicToc

Aggiungiamo alla classe nel file .h due variabili di tipo cMessage * (event e tictocMsg)

```
#include <stdio.h>
private:
        cMessage *event;
        cMessage *tictocMsg;
```

TicToc

Aggiungiamo anche il costruttore e il distruttore nel file .h

```
public:
    Txc1();
     virtual ~Txc1();
```

MCLab

TicToc

Nel file .cc:

```
Txc1::Txc1()
{
        event = tictocMsg = nullptr;
}
Txc1::~Txc1()
{
        cancelAndDelete(event);
        delete tictocMsg;
}
```

MCLab

TicToc

Invieremo gli self-message con la funzione scheduleAt (), specificando quando dovrebbe essere consegnato al modulo.

```
void Txc1::initialize()
{
      event = new cMessage("event");
      tictocMsg = nullptr;
      if (strcmp("tic", getName()) == 0) {
            tictocMsg = new cMessage("tictocMsg");
            scheduleAt(5.0, event);
            }
}
```

MCLab

TicToc

In handleMessage () ora dobbiamo distinguere se un nuovo messaggio è arrivato tramite il gate di input oppure è un self-message.

Nel secondo caso, il self-message è tornato indietro, quindi il timer è scaduto e deve essere rinviato.

MCLab

TicToc

```
void Txc1::handleMessage(cMessage *msg)
{
      if (msg == event) {
            send(tictocMsg, "out");
            tictocMsg = nullptr;
            }
      else {
            tictocMsg = msg;
            scheduleAt(simTime()+1.0, event);
            }
}
```

TicToc

Prima di lanciare la simulazione aggiungiamo il tempo di simulazione.

Per farlo aggiungiamo nel file .ini:

- sim-time-limit= 10s

MCLab

TicToc

Esercizio:

Realizzare una semplice rete utilizzando 5 nodi di tipo Txc1

TicToc

```
simple Txc1
{
    parameters:
        int limit = default(2);
    gates:
        input in[];
        output out[];
}

network TicToc
{
    submodules:
        tic[6]: Txc1;
    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++;

}
```

TicToc

Finora non si è data importanza al canale considerandolo identico per tutte le connessioni.

Nello specifico abbiamo considerato sempre lo stesso ritardo ma in realtà è possibile creare diversi tipi di canale.

Nel file .ned, all'interno della sessione network inseriamo:

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

MCLab

TicToc

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++;
```

MCLab

TicToc

Consideriamo ora la sottoclasse cMessage

Finora abbiamo utilizzato un messaggio standard che conteneva informazioni non settate da noi ma potrebbe essere utile avere delle informazioni aggiuntive

Ad esempio il destinatario e la sorgente del messaggio -> per fare questo creiamo un file .msg

TicToc

Creiamo il file dentro la cartella src, selezioniamo Message Definition, empty file.

- tictoc.msg

TicToc

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

Abbiamo aggiunto la destinazione, la sorgente e il numero di hop che esso compie come variabili del messaggio.

MCLab

TicToc

Andiamo ad utilizzare il nostro messaggio all'interno della simulazione

Per far questo è necessario creare una classe di tipo message, un processo molto lungo e laborioso ma OMNeT++ crea per noi la classe tictoc_m.h con all'interno tutti i metodi di get e set per tutte le variabili contenute al suo interno.

MCLab

TicToc

```
#include "tictoc_m.h"
class Txc1 : public cSimpleModule
{
    protected:
        virtual TicTocMsg *generateMessage();
        virtual void forwardMessage(TicTocMsg *msg);
        virtual void initialize() override;
        virtual void handleMessage(cMessage *msg) override;
};
```

MCLab

TicToc

```
void Txc1::initialize()
{
    if (getIndex() == 0) {
        TicTocMsg *msg = generateMessage();
        scheduleAt(0.0, msg);
        }
}
```

MCLab

TicToc

```
void Txc1::handleMessage(cMessage *msg)
{
        TicTocMsg *ttmsg = check_and_cast<TicTocMsg *>(msg);
        if (ttmsg->getDestination() == getIndex()) {
                bubble("ARRIVED, starting new one!");
                delete ttmsg;
                TicTocMsg *newmsg = generateMessage();
                forwardMessage(newmsg);
        }
        else {
                  forwardMessage(ttmsg);
        }
}
```

MCLab

TicToc

```
TicTocMsg *Txc1::generateMessage()
{
        int src = getIndex();
        int n = getVectorSize();
        int dest = intuniform(0, n-2);
        if (dest >= src)
                dest++;
        char msgname[20];
        sprintf(msgname, "tic-%d-to-%d", src, dest);
        TicTocMsg *msg = new TicTocMsg(msgname);
        msg->setSource(src);
        msg->setDestination(dest);
        return msg;
}
```

MCLab

TicToc

```
void Txc1::forwardMessage(TicTocMsg *msg)
{
    msg->setHopCount(msg->getHopCount()+1);
    int n = gateSize("out");
    int k = intuniform(0, n-1);
    send(msg, "out", k);
}
```

TicToc

Con OMNeT++ è possibile ottenere delle informazioni dettagliate circa i messaggi che vengono scambiati configurando nel file .ini:

record-eventlog = true

Permetterà di visualizzare successivamente il file di log

TicToc

Ad esempio possiamo essere interessati al conteggio degli hop di ogni messaggio al momento dell'arrivo

Per fare questo memorizziamo in un vettore di uscita con coppie di valori (time, value) per ogni nodo

Possiamo quindi calcolare: deviazione standard, media, valore minimo, massimo per ogni nodo, e scriverli in un file alla fine della simulazione

Utilizzeremo strumenti offerti da Omnet ++ IDE per analizzare i file di output

MCLab

TicToc

```
class Txc1 : public cSimpleModule
{
    private:
        long numSent;
        long numReceived;
        cLongHistogram hopCountStats;
        cOutVector hopCountVector;
    protected:
        virtual TicTocMsg *generateMessage();
        virtual void forwardMessage(TicTocMsg *msg);
        virtual void initialize() override;
        virtual void handleMessage(cMessage *msg) override;
        virtual void finish() override;
};
```

MCLab

TicToc

```
void Txc1::initialize()
{
        numSent = 0;
        numReceived = 0;
        WATCH(numSent);
        WATCH(numReceived);
        hopCountStats.setName("hopCountStats");
        hopCountStats.setRangeAutoUpper(0, 10, 1.5);
        hopCountVector.setName("HopCount");
         if (getIndex() == 0) {
                TicTocMsg *msg = generateMessage();
                scheduleAt(0.0, msg);
                }
}
```

MCLab

TicToc

```
void Txc1::handleMessage(cMessage *msg)
{
        TicTocMsg *ttmsg = check_and_cast<TicTocMsg *>(msg);
        if (ttmsg->getDestination() == getIndex()) {
                int hopcount = ttmsg->getHopCount();
                bubble("ARRIVED, starting new one!");
                numReceived++;
                hopCountVector.record(hopcount);
                hopCountStats.collect(hopcount);
                delete ttmsg;
                TicTocMsg *newmsg = generateMessage();
                forwardMessage(newmsg);
                numSent++;
        }
        else {
                  forwardMessage(ttmsg);
        }
}
```

MCLab

TicToc

```
TicTocMsg *Txc1::generateMessage()
{
        int src = getIndex();
        int n = getVectorSize();
        int dest = intuniform(0, n-2);
        if (dest >= src)
                dest++;
        char msgname[20];
        sprintf(msgname, "tic-%d-to-%d", src, dest);
        TicTocMsg *msg = new TicTocMsg(msgname);
        msg->setSource(src);
        msg->setDestination(dest);
        return msg;
}
```

MCLab

TicToc

```
void Txc1::forwardMessage(TicTocMsg *msg)
{
      msg->setHopCount(msg->getHopCount()+1);
      int n = gateSize("out");
      int k = intuniform(0, n-1);
       send(msg, "out", k);
}
```

MCLab

TicToc

```
void Txc1::finish()
{
    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");
}
```

EV serve per la registrazione delle informazioni con il livello di log di default.

TicToc

I file sono memorizzati nella sottodirectory results.

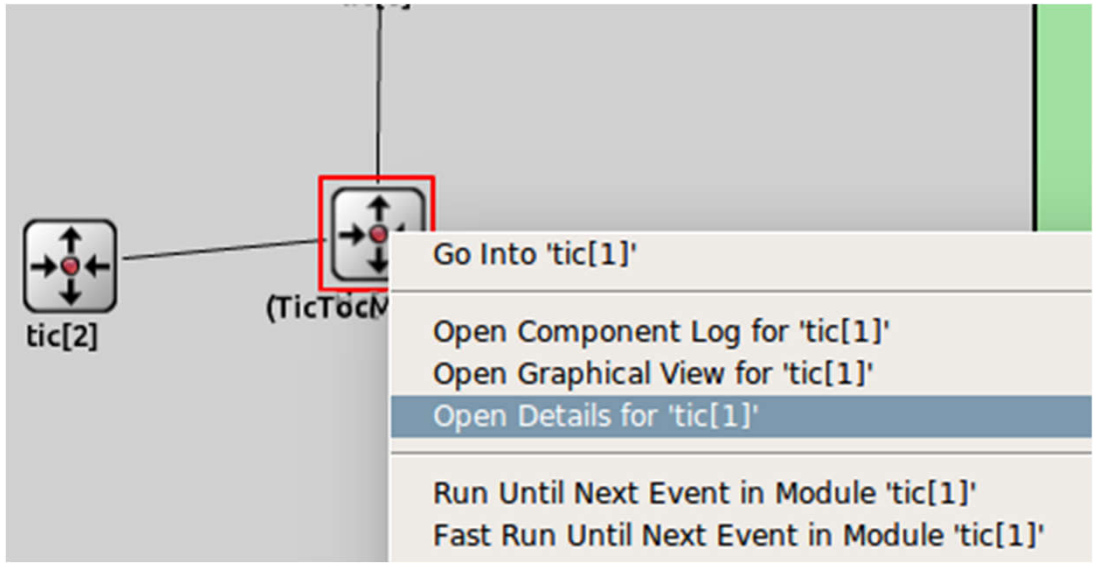
È inoltre possibile visualizzare i dati durante la simulazione.

Per fare questo, fare clic destro su un modulo, e scegliere Open Details.

Nella pagina del sommario del modulo troverete gli hopCountStats e gli oggetti hopCountVector. Per aprirgli, fate clic destro sul cLongHistogram hopCountStats o cOutVector hopcount, e fare clic su Open Graphical View.

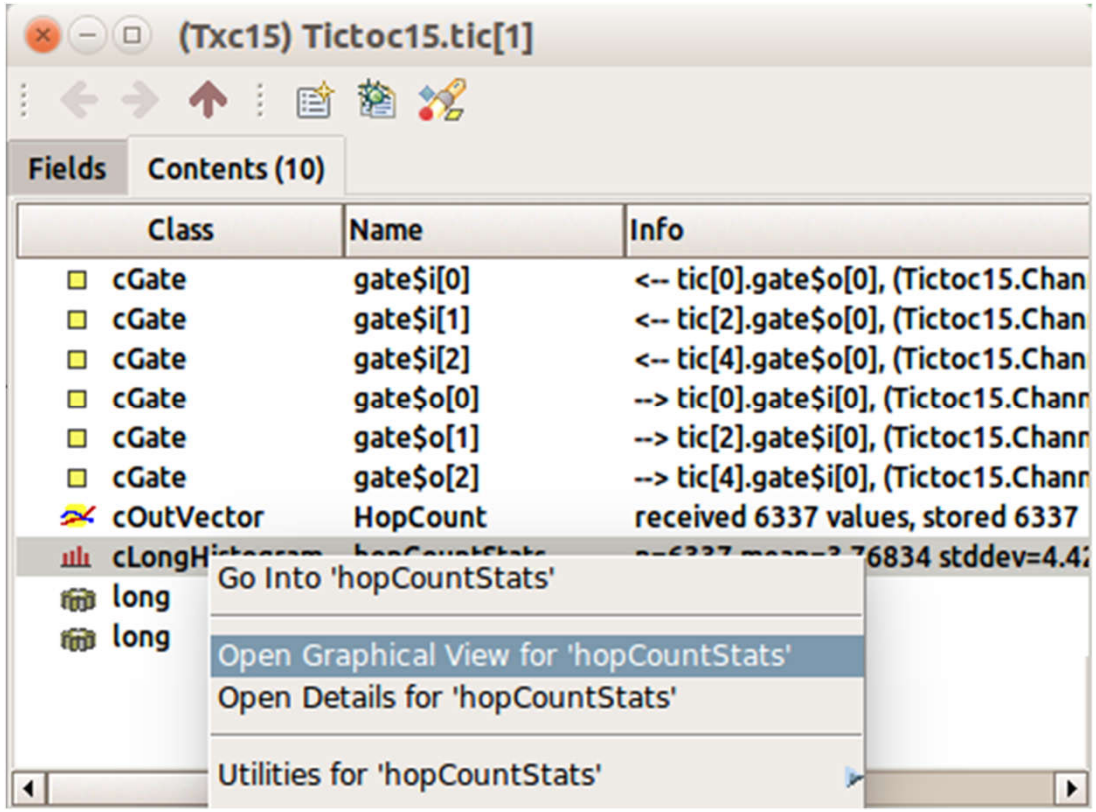
MCLab

TicToc



MCLab

TicToc



TicToc

Non sempre è possibile prevedere in anticipo quali dati e quali statistiche saranno utili al fine della nostra simulazione

Per questo OMNeT++ fornisce un ulteriore meccanismo per effettuare il record dei values e degli eventi che si sono verificati durante la simulazione

Ogni modello può emettere "segnali" che possono trasportare informazioni (values) o un oggetto.

TicToc

Si può decidere quali segnali emettere, quali dati allegare ad ognuno di loro e quando emetterli

A questi segnali è possibile collegare dei ***listener*** in grado di elaborare o registrare questi dati-> in tal modo il codice del modello non deve contenere alcun codice specifico per la raccolta delle statistiche e possiamo liberamente aggiungere ulteriori statistiche senza metter mano al codice C ++

TicToc

Per prima cosa definiamo il nostro segnale dentro il file .h

private:

simsignal_t arrivalSignal;

TicToc

È necessario registrare tutti i segnali prima di utilizzarli, quindi dentro initialize():

arrivalSignal = registerSignal("arrival");

Ora è possibile emettere il nostro segnale nel momento in cui il messaggio arriva al nodo di destinazione

MCLab

TicToc

```
void Txc1::handleMessage(cMessage *msg)
{
        TicTocMsg *ttmsg = check_and_cast<TicTocMsg *>(msg);
        if (ttmsg->getDestination() == getIndex()) {
                int hopcount = ttmsg->getHopCount();
                emit(arrivalSignal, hopcount);
                EV << "Message " << ttmsg << " arrived after " << hopcount << " hops.\n";
                bubble("ARRIVED, starting new one!");
                delete ttmsg;
                EV << "Generating another message: ";
                TicTocMsg *newmsg = generateMessage();
                EV << newmsg << endl;
                forwardMessage(newmsg);
        }
        else {
                  forwardMessage(ttmsg);
        }
}
```

TicToc

Definiamo il segnale emesso anche nel file .ned:

```
parameters:
    @signal[arrival](type="long");
    @statistic[hopCount](title="hop count"; source="arrival";
     record=vector,stats; interpolationmode=none);
```

TicToc

Per finire modifichiamo il file .ini aggiungendo

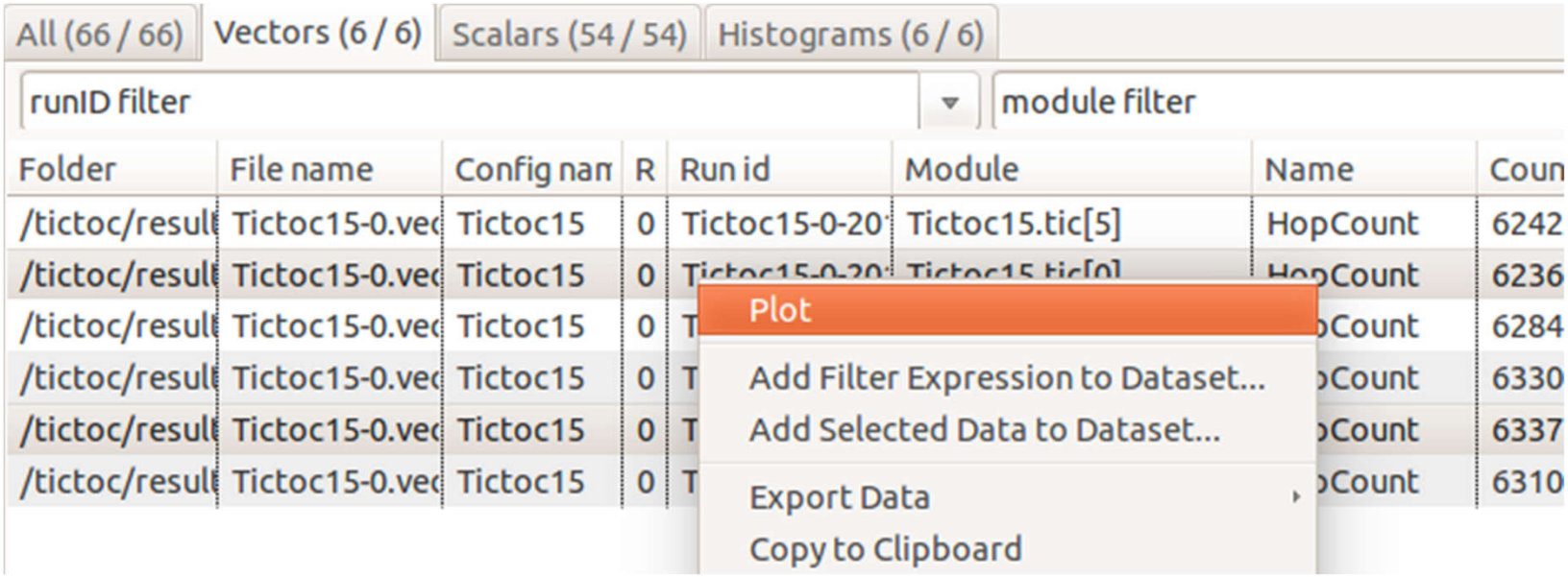
****.tic[1].hopCount.result-recording-modes = +histogram**

****.tic[0..2].hopCount.result-recording-modes = -vector**

I vectors contengono i record dei dati in funzione del tempo, mentre gli scalari tipicamente registrano valori aggregati al termine della simulazione. Per aprire lo strumento di analisi results, fare doppio clic su entrambi i file .sca o .vec nella OMNet ++ IDE.

MCLab

TicToc



UNIVERSITY OF CAGLIARI

DIEE - Department of Electrical and Electronic Engineering

Grazie per l'attenzione

Il tutorial e la documentazione di OMNeT++ si possono trovare sul sito:

www.omnetpp.org