OMNet++ Tutorial

Chang-Gun Lee (cglee@snu.ac.kr)

Assistant Professor

The School of Computer Science and Engineering

Seoul National University

Why OMNet++

- SMPL is not modular and so not extendable
- We need a OO-based simulation design
- OMNet++ supports that

What we need

- Linux (e.g., RedHat 8.0) installed with full options
- Tcl/Tk (version 8.4.14)
- BLT (version 2.4z)
- OMNet++ (version 3.3)

Step 1: Tcl/Tk install (1)

1. Check if you already have one
   - Usually /usr/bin (tslsh, wish), /usr/include (tcl.h), /usr/lib (libtclxx/so)
   - If the current version is 8.3, remove all of them (/usr/bin/wish*, /usr/bin/tclsh*, usr/include/tcl*.h, /usr/include/tk*.h, /usr/lib/libtcl*)
2. Download “tcl8.4.14-src.tar.gz” and “tk8.4.14-src.tar.gz” from http://www.tcl.tk/software/tcltk/
3. Decompress them
   - tar zxvf tcl8.4.14-src.tar.gz
   - tar zxvf tk8.4.14-src.tar.gz
4. Configure tcl and tk
   - configure tcl
     - cd tcl8.4.14/unix
     - ./configure --enable-gcc --enable-shared --prefix=/usr/X11R6 --exec-prefix=/usr/X11R6
   - configure tk
     - cd tk8.4.14/unix
     - ./configure --enable-gcc --enable-shared --with-tcl=/home/yourHome/tcl8.4.14/unix \\
       --prefix=/usr/X11R6 --exec-prefix=/usr/X11R6

Step 1: Tcl/Tk install (2)

5. Make
   - cd tcl8.4.14/unix
   - make
   - cd tk8.4.14/unix
   - make
6. Test
   - cd tcl8.4.14/unix
   - make test
   - cd tk8.4.14/unix
   - make test
7. Install (for this, you should be the super user)
   - cd tcl8.4.14/unix
   - make install (this will copy necessary files to /usr/X11R6/(lib, include, man/mann, bin)
   - cd tk8.4.14/unix
   - make install (this will copy necessary files to /usr/X11R6/(lib, include, man/mann, bin)

See http://www.tcl.tk/doc/howto/compile.html for more details.

Step 2: BLT install

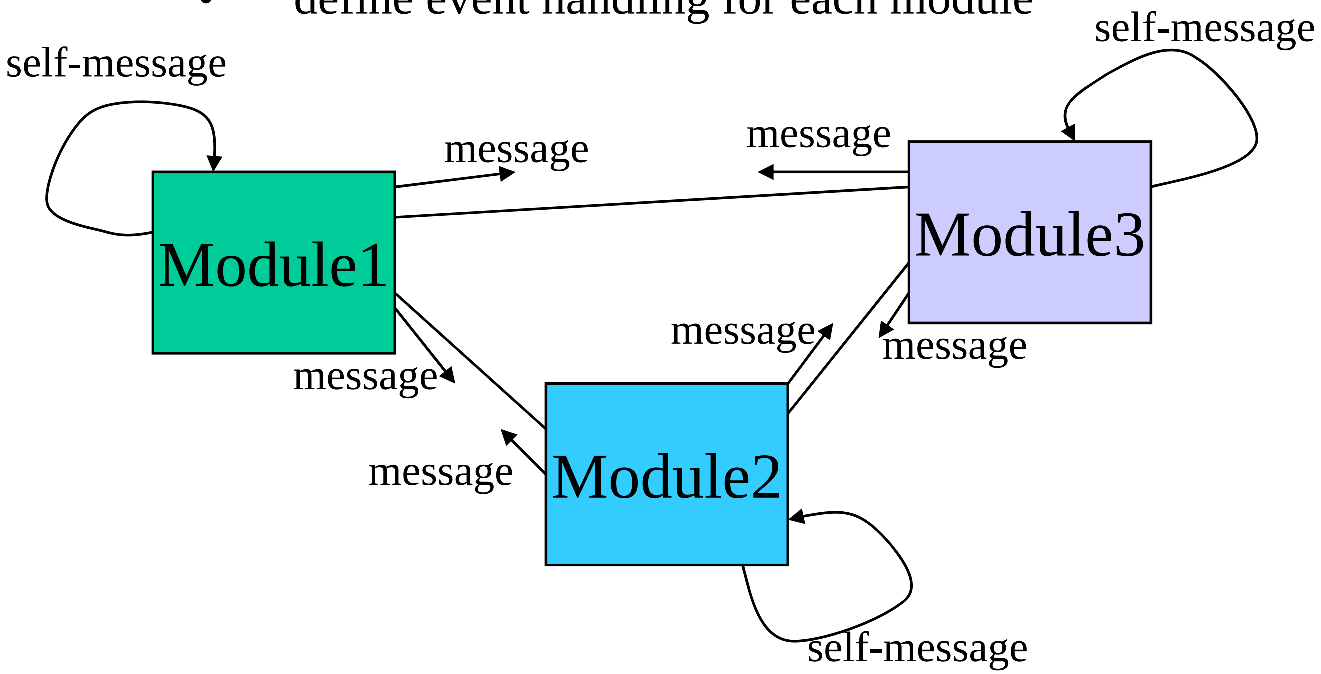
1. Download “BLT2.4z.tar.gz” from http://sourceforge.net/projects/blt
2. Decompress
   - tar zxvf BLT2.4z.tar.gz
3. Configure
   - cd blt2.4z
   - ./configure --with-tcl=/usr/X11R6 --prefix=/usr/X11R6 --with-cc=gcc
4. Make
   - cd blt2.4z
   - make
5. Test
   - cd demos
   - ./graph1.tcl (or “../src/bltwish ./graph1.tcl”)
6. Install (be the super user first)
   - cd blt2.4z
   - make install

Step 3: OMNet++ install

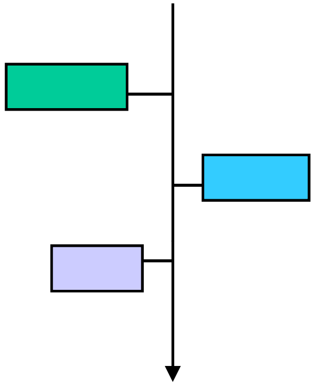
1. Download “omnetpp-3.3-src.tgz” from http://www.omnetpp.org
2. Decompress
   - tar zxvf omnetpp-3.3-src.tgz
3. Environment variable setting (add the followings in .bashrc)
   - export PATH=$PATH:~/omnetpp-3.3/bin
   - export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/omnetpp-3.3/lib
   - source .bashrc
4. Configure (edit configure.user as needed)
   - Uncomment the line of TK_CFLAGS and change the line to
     - TK_CFLAGS=“-I/user/X11R6/include”
   - cd omnetpp-3.3
   - ./configure
5. Make
   - cd omnetpp-3.3
   - make
   - Note: if you encounter an error message of “isdigit undefined in vectortilereader.cc”, add <ctype.h> into the include file list
6. Test
   - cd omnetpp-3.3/samples/dyna
   - ./dyna
   - Note: if you encounter “Cannot load libtcl8.4.so”, add /usr/X11R6/lib to the LD_LIBRARY_PATH variable by editing .bashr

OMNet++ Overview

- Object-Oriented simulation tool
  - System is defined by “a connection of modules”
  - Each module handles “messages” (events) according to the system specification
  - Only thing we have to do is
    - define modules
    - define interconnections of the modules
    - define events for each module
    - define event handling for each module



OMNet++ kernel manages a global message (event) list



TicToc Tutorial (1)

- Two nodes (tic and toc) ping-pong a message
- What we program
  - tictoc1.ned: define modules and their connections
  - txc1.cc: define the operations for events and messages
  - omnetpp.ini: specify simulation parameters, networks, etc.
- What we have to do
  - opp_makemake (This will automatically creat Makefile)
  - make
  - ./tictoc
- Keywords to learn
  - simple module (gates)
  - compound module (submodules and connections)
  - network
  - message (=event)

TicToc1 Codes

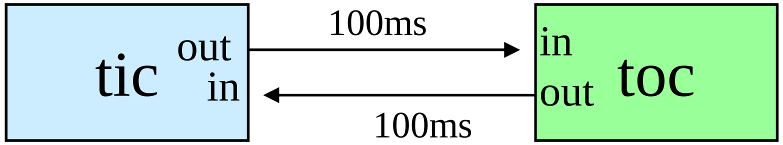
```
simple Txc1
     gates:
                  in: in;
                  out: out;
endsimple



module Tictoc1
    submodules:
                  tic: Txc1;
                  toc: Txc1;
    connections:
                  tic.out --> delay 100ms --> toc.in;
                  tic.in <-- delay 100ms <-- toc.out;
endmodule

network tictoc1: Tictoc1
endnetwork
```

tictoc.ned



TicToc1 Codes

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

class Txc1: public cSimpleModule
{
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Txc1);

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

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

txc1.cc

```
[General]
preload-ned-files=*.ned
network=tictoc1

[Parameters]

tictoc4.toc.limit = 5
tictoc6.tic.delayTime = exponential(3)
tictoc6.toc.delayTime = truncnormal(3,1)
```

omnetpp.ini

TicToc Tutorial (2)

- Beautify modules and add debugging output
- What to learn
  - Module output (right-click module icon and choose “Module output”) to see a separate window showing only the outputs of a specific module

TicToc2 Codes

```
simple Txc2
   gates:
                  in: in;
                  out: out;
endsimple

module Tictoc2
   submodules:
       tic: Txc2;
          display: "i=block/process,cyan";
       toc: Txc2;
          display: "i=block/process,gold";
   connections:
       tic.out --> delay 100ms --> toc.in;
       tic.in <-- delay 100ms <-- toc.out;
endmodule

network tictoc2: Tictoc2
endnetwork
```

tictoc2.ned

```
class Txc2: public cSimpleModule
{
   protected:
        virtual void initialize();
        virtual void handleMessage(cMessage *msg);
};

Define_Module(Txc2);

void Txc2::initialize()
{
   if(strcmp("tic", name())==0)
   {
       cMessage *msg = new cMessage("tictocMsg");
       ev << "Sending initial message\n";
       send(msg, "out");
   }
}

void Txc2::handleMessage(cMessage *msg)
{
   ev << "Receiving message `" << msg->name()
<< "', sending it out again\n";
   send(msg, "out");
}
```

txc2.cc

TicToc Tutorial (3)

- "State Variables" of a module
  - Let's add a counter that keeps the number of msg exchanges
  - Delete the message after 10 exchanges
- What to learn
  - WATCH(counter): this makes it possible to see the counter value (state variable) in Tkenv.
    - Double-click on tic's icon, then choose the Contents page from the inspector window that pops up

TicToc3 Codes

```
class Txc3: public cSimpleModule
{
  private:
    int counter;
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Txc3);

void Txc3::initialize()
{
  counter = 10;
  WATCH(counter);
  if(strcmp("tic", name())==0)
  {
    cMessage *msg = new cMessage("tictocMsg");
    ev << "Sending initial message\n";
    send(msg, "out");
  }
}
```

txc3.cc

```
void Txc3::handleMessage(cMessage *msg)
{
  counter--;
  if (counter==0){
        ev << name() <<"'s counter reached
zero, deleting message\n";
        delete msg;
  }
  else{
        ev << name() << "'s counter is "
<< counter << ", sending back message\n";
        send(msg, "out");
  }
}
```

TicToc Tutorial (4)

- “Simulation Parameters”
  - Let’s give input parameter “limit” as the limit of message exchanges (instead of 10)
  - Delete the message after “limit” exchanges
- What to learn
  - Parameter declaration in Module definition of *.ned file
  - Two ways to give the input parameter
    - When instantiating the module object in *.ned file
    - By giving the value in omnetpp.ini file
  - How to make a module to read the input parameter?
    - counter = par(“limit”);

TicToc4 Codes

```
simple Txc4
    parameters:
        limit: numeric const;
    gates:
        in: in;
        out: out;
endsimple

module Tictoc4
    submodules:
        tic: Txc4;
             parameters:
                  limit = 8;
            display: "i=block/process,cyan";
        toc: Txc4;
            display: "i=block/process,gold";
    connections:
        tic.out --> delay 100ms --> toc.in;
        tic.in <-- delay 100ms <-- toc.out;
endmodule

network tictoc4 : Tictoc4
endnetwork
```

```
[General]
network=tictoc1

[Parameters]

tictoc4.toc.limit = 5
```

```
class Txc4: public cSimpleModule
{
   private:
      int counter;
   protected:
      virtual void initialize();
      virtual void handleMessage(cMessage *msg);
};

Define_Module(Txc4);

void Txc4::initialize()
{
   counter = par(“limit”);
   WATCH(counter);

   …. same as before
}
```

TicToc Tutorial (5)

- Modeling processing delay (different events)
  - Let’s hold a message for 1 simulated sec to model the processing delay of a message
- What to learn
  - How to schedule an event at a specific time
    - scheduleAt(simTime()+1.0, event);
  - How to differentiate two events (self-message and actual message reception)

TicToc5 Codes

```
class Txc5: public cSimpleModule
{
   private:
      cMessage *event;
      cMessage *tictocMsg;
  public:
      Txc5();
      virtual ~Txc5();
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Txc5);
Txc5::Txc5()
{
     event = tictocMsg = NULL;
}
Txc5::~Txc5()
{
     cancelAndDelete(event);
     delete tictocMsg;
}
```

```
void Txc5::initialize()
{
   event = new cMessage("event");
   if(strcmp("tic", name())==0){
       ev << "Scheduling first send to t=5.0s\n";
       tictocMsg = new cMessage("tictocMsg");
       scheduleAt(5.0, event);
   }
}
void Txc5::handleMessage(cMessage *msg)
{
   if (msg==event){
       ev << "Wait period is over, sending back
message\n";
       send(tictocMsg, "out");
       tictocMsg = NULL;
   }
   else{
       ev << "Message arrived, starting to wait 1
sec...\n";
       tictocMsg = msg;
       scheduleAt(simTime()+1.0, event);
   }
}
```

TicToc Tutorial (6)

- Random numbers and parameters
  - Let’s hold a message for “random” sec (instead of 1 sec) to model the “random” processing delay of a message
  - Let’s probabilistically “lose” the packet
- What to learn
  - How to get a random parameter
  - How to use a random number generation function to model the probabilistic loss

TicToc6 Codes

```
simple Txc6
   parameters:
      delayTime: numeric;
   gates:
      in: in;
      out: out;
endsimple
```

```
[Parameters]
tictoc6.tic.delayTime = exponential(3)
tictoc6.toc.delayTime = truncnormal(3,1)
```

```
void Txc6::handleMessage(cMessage *msg)
{
  if (msg==event){
      ev << "Wait period is over, sending back
message\n";
      send(tictocMsg, "out");
      tictocMsg = NULL;
  }
  else{
      if (uniform(0,1) < 0.1){
              ev << "\"Losing\" message\n";
              delete msg;
      }
      else{
          double delay = par("delayTime");
          ev << "Message arrived, starting to wait
" << delay << " secs...\n";
          tictocMsg = msg;
          scheduleAt(simTime()+delay, event);
      }
  }
}
```

TicToc Tutorial (7)

- Timeout and cancelling timer
  - Let’s simulate “stop and wait” and “new message sending” if the message is lost.
  - Tic sends the message and Toc acknowledge (Separate simple modules for Tic and Toc)
  - Tic starts “timer” whenever it sends the message. When the timer expires before receiving ACK from Toc, Tic sends another one.
  - Toc sends back the message as ACK. But, it will drop the message with a small probability.
  - (No processing delay “delayTime” for the simplicity)
- What to learn
  - How to model the timeout event
  - How to cancel the already scheduled timeout event (why?)
    - cancelEvent(timeoutEvent)
  - bubble(“xxxx”)

TicToc7 Codes

```
class Tic7: public cSimpleModule
{
  private:
      double timeout;
      cMessage *timeoutEvent;
  public:
     Tic7();
     virtual ~Tic7();
  ……
};
Define_Module(Tic7);
Tic7::Tic7(){
  timeoutEvent = NULL;
}
Tic7::~Tic7(){
  cancelAndDelete(timeoutEvent);
}

void Tic7::initialize()
{
  timeout = 1.0;
  timeoutEvent = new cMessage("timeoutEvent");
  ev << "Sending initial message\n";
  cMessage *msg = new cMessage("tictocMsg");
  send(msg, "out");
  scheduleAt(simTime()+timeout, timeoutEvent);
}
```

```
void Tic7::handleMessage(cMessage *msg)
{
  if (msg==timeoutEvent)
 {
    ev << "Timeout expired, resending message
and restarting timer\n";
    cMessage *msg=new cMessage("tictocMsg");
    send(msg, "out");
    scheduleAt(simTime()+timeout, timeoutEvent);
 }
 else // ACK message arrived
 {
    ev << "Timer cancelled.\n";
    cancelEvent(timeoutEvent);
    delete msg;

   cMessage *msg=new cMessage("tictocMsg");
   send(msg,"out");
   scheduleAt(simTime()+timeout, timeoutEvent);
 }
}
```

Toc7 is same as before except **bubble(“msg lost”)**

TicToc Tutorial (8)

- Keeping the message copy and retransmitting the copy until acknowledged
  - Let’s keep the copy of the original message
  - Retransmit the same copy (not the new one) if timer expires
  - Remove the copy when the acknowledgement is received.
  - After that, send a new message with increased sequence number
- What to learn
  - How to duplicate the original message
    - msg->dup();

TicToc8 Codes

```
class Tic8: public cSimpleModule
{
  private:
    double timeout;
    cMessage *timeoutEvent;
    int seq;
    cMessage *message;
  public:
    Tic8();
    virtual ~Tic8();
  protected:
    virtual cMessage *generateNewMessage();
    virtual void sendCopyOf(cMessage *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

void Tic8::initialize()
{
  seq = 0;
  timeout = 1.0;
  timeoutEvent = new cMessage("timeoutEvent");
  ev << "Sending initial message\n";
  message = generateNewMessage();
  sendCopyOf(message);
  scheduleAt(simTime()+timeout, timeoutEvent);
}
```

```
void Tic8::handleMessage(cMessage *msg)
{
  if (msg==timeoutEvent){ // timeout expired
     sendCopyOf(message);
     scheduleAt(simTime()+timeout, timeoutEvent);
  }
  else{  // ACK message arrived
     delete msg; // delete ACK
     cancelEvent(timeoutEvent);
     delete message; // delete kept message
     message = generateNewMessage();
     sendCopyOf(message);
     scheduleAt(simTime()+timeout, timeoutEvent);
  }
}
cMessage *Tic8::generateNewMessage()
{
  char msgname[20];
  sprintf(msgname, "tic-%d", ++seq);
  cMessage *msg = new cMessage(msgname);
  return msg;
}
void Tic8::sendCopyOf(cMessage *msg)
{
  cMessage *copy = (cMessage *) msg->dup();
  send(copy, "out");
}
```

TicToc Tutorial (9)

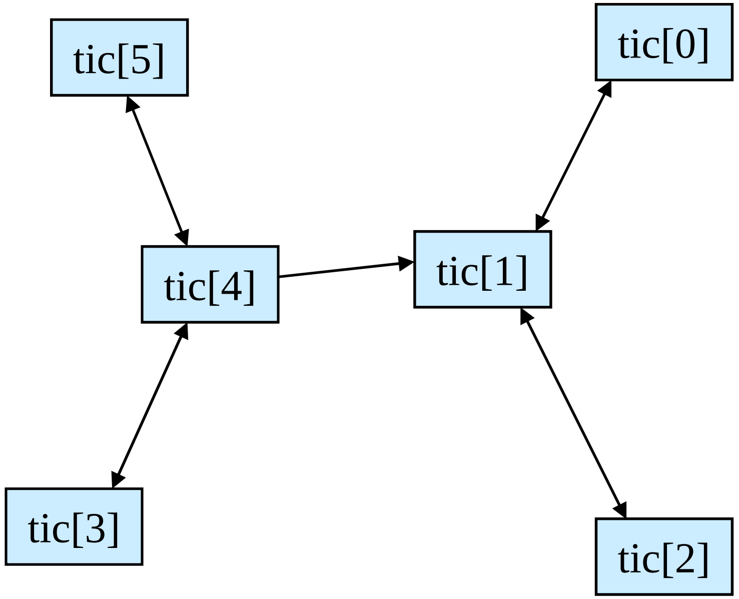
- More than two nodes and random routing
  - Let’s make a network of 6 nodes
  - One node (say tic0) sends a message destined to another node (tic3)
  - If a node receives this message
    - If the node is tic3: it is the final destination. Delete the message. No more event to handle
    - If not, it forwards the message to the randomly chosen out gate.
- What to learn
  - How to model the array of submodules in *.ned file
  - How to model the array of gates in *.ned file (We do not know how many neighbors will be connected. So, the number of gates should be generic.)
  - How to connect the arrayed gates

TicToc9 Codes

```
simple Txc9
  gates:
    in: in[];
    out: out[];
endsimple

module Tictoc9
 submodules:
   tic: Txc9[6];
     display: "i=block/process";
   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++;
endmodule

network tictoc9 : Tictoc9
endnetwork
```



TicToc9 Codes

```
class Txc9: public cSimpleModule
{
  protected:
    virtual void forwardMessage(cMessage *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Txc9);

void Txc9::initialize()
{
  if (index()==0)
  {
      char msgname[20];
      sprintf(msgname, "tic-%d", index());
      cMessage *msg = new cMessage(msgname);
      scheduleAt(0.0, msg);
  }
}
```

```
void Txc9::handleMessage(cMessage *msg)
{
  if (index()==3)
  {
      ev << "Message " << msg << " arrived.\n";
      delete msg;
  }
  else // intermediate node
  {
      forwardMessage(msg);
  }
}

void Txc9::forwardMessage(cMessage *msg)
{
  int n= gate("out")->size();
  int k= intuniform(0, n-1);
  ev << "Forwarding message " << msg << " on
port out[" << k << "]\n";
  send(msg, "out", k);
}
```

TicToc Tutorial (10)

- Carrying info in message (event): Define our own message class
  - Let’s draw a random destination for each message
  - Let’s add the destination address to the message
  - If the final destination node receives the message, it will create and send another message with randomly drawn destination
- What to learn
  - How to define our own message class in xxx.msg file
  - How xxx.msg file is coverted to xxx_m.h and xxx_h.cc files by “opp_msgc”
  - How our “handleMessage” can access the fields of our new message class

TicToc10 Codes

```
message TicTocMsg10
{
   fields:
       int source;
       int destination;
       int hopCount = 0;
}
```

tictoc10.msg

```
#include "tictoc10_m.h"

class Txc10: public cSimpleModule
{
  protected:
      virtual TicTocMsg10 *generateMessage();
      …………
};

Define_Module(Txc10);

void Txc10::initialize()
{
  if (index()==0)
  {
      TicTocMsg10 *msg = generateMessage();
      scheduleAt(0.0, msg);
  }
}
```

```
void Txc10::handleMessage(cMessage *msg)
{
  TicTocMsg10 *ttmsg = check_and_cast<TicTocMsg10
*>(msg);
  if (ttmsg->getDestination()==index() ){ \\ arrived
      ev << "Message " << ttmsg << " arrived after "
<< ttmsg->getHopCount() << " hops.\n";
      delete ttmsg;
      TicTocMsg10 *newmsg = generateMessage();
      forwardMessage(newmsg);
   }
  else{
      forwardMessage(ttmsg);
  }
}

TicTocMsg10 *Txc10::generateMessage()
{
  int src = index();   int n = size();
  int dest = intuniform(0, n-2);
  if (dest>=src) dest++;
  char msgname[20];
  sprintf(msgname, "tic-%d-to-%d", src, dest);
  TicTocMsg10 *msg = new TicTocMsg10 (msgname);
  msg->setSource(src);
  msg->setDestination(dest);
  return msg;
}
```

TicToc Tutorial (11)

- Displaying the number of packets sent/received
  - Let’s keep the number of packets sent/received for each node
- What to learn
  - How to view the state variables of a module
    - Inspect menu (Find/inspect object dailog)
  - How to show the state variables on the GUI network diagram
    - ev.isGUI()
    - displayString().setTagArg

TicToc11 Codes

```
class Txc11: public cSimpleModule
{
  private:
    long numSent;
    long numReceived;
  …
}

void Txc11::initialize()
{
  numSent = 0;
  numReceived = 0;
  WATCH(numSent);
  WATCH(numReceived);

  if (index()==0)
  {
     TicTocMsg11 *msg = generateMessage();
     scheduleAt(0.0, msg);
  }
}
```

```
void Txc11::handleMessage(cMessage *msg)
{
  TicTocMsg11 *ttmsg = check_and_cast<TicTocMsg11
*>(msg);
  if (ttmsg->getDestination()==index()){
     int hopcount = ttmsg->getHopCount();
     numReceived++;
    delete ttmsg;
    TicTocMsg11 *newmsg = generateMessage();
    forwardMessage(newmsg);
    numSent++;

    if (ev.isGUI())
                 updateDisplay();
  }
  else {
     forwardMessage(ttmsg);
  }
}

void Txc11::updateDisplay()
{
  char buf[40];
  sprintf(buf, "rcvd: %ld sent: %ld",
numReceived, numSent);
  displayString().setTagArg("t",0,buf);
}
```

TicToc Tutorial (12)

- Collect statistics
  - Let’s collect hopCount statistics of messages
- What to learn
  - How to add and use “output vector” object (creating omnetpp.vec file)
    - to view, click module to see module inspector’s Contents page
  - How to add and use “histogram” object (creating omnetpp.sca file)
    - to view, click module to see module inspector’s Contents page
  - How to explicitly call finish() of all modules to flush out the histogram scalar data to omnetpp.sca
    - Simulate|Call finish menu
  - How to offline visualize omnetpp.vec file using the “plove” tool
  - How to offline visualize omnetpp.sca file using the “scalars” tool

TicToc12 Codes

```
class Txc12: public cSimpleModule
{
  private:
    ….
    cLongHistogram hopCountStats;
    cOutVector hopCountVector;

  protected:
     …
     virtual void finish();
};

Define_Module(Txc12);


void Txc12::initialize()
{
   …..
  hopCountStats.setName("hopCountStats");
  hopCountStats.setRangeAutoUpper(0, 10, 1.5);
  hopCountVector.setName("HopCount");

  if (index()==0){
     TicTocMsg12 *msg = generateMessage();
    scheduleAt(0.0, msg);
  }
}
```

```
void Txc12::handleMessage(cMessage *msg)
{
if (ttmsg->getDestination()==index()){
      int hopcount = ttmsg->getHopCount();
      numReceived++;
      hopCountVector.record(hopcount);
      hopCountStats.collect(hopcount);
      delete ttmsg;
      TicTocMsg12 *newmsg = generateMessage();
      forwardMessage(newmsg);
      numSent++;
       ….
  }
  else {
      forwardMessage(ttmsg);
  }
}

void Txc12::finish()
{
  ev << "Hop count, min " << hopCountStats.min();
  ev << "Hop count, max " << hopCountStats.max();
  ev << "Hop count, mean " << hopCountStats.mean();
  ev << "Hop count, stddev " << hopCountStats.stddev();
  recordScalar("#sent", numSent);
  recordScalar("#received", numReceived);
  hopCountStats.recordScalar("hop count");
}
```

TicToc Tutorial (13)

- Better routing (pre-built routing table for each node)
  - Let’s specify the routing table using the parameters in omnetpp.ini
- What to learn
  - Format of routing table for each node
    - (dest, outgate) pairs
  - How to read the routing parameters into module’s local memory
  - For the message destination, how to get the forwarding out gate from the routing table

TicToc12 Codes

```
simple Txc13
    parameters:
                route0: numeric,
                route1: numeric,
                route2: numeric,
                route3: numeric,
                route4: numeric,
                route5: numeric;
    gates:
        in: in[];
        out: out[];
endsimple
```

tictoc13.ned

```
[Parameters]
tictoc13.tic[0].route0 = 0
tictoc13.tic[0].route1 = 0
tictoc13.tic[0].route2 = 0
tictoc13.tic[0].route3 = 0
tictoc13.tic[0].route4 = 0
tictoc13.tic[0].route5 = 0

tictoc13.tic[1].route0 = 0
tictoc13.tic[1].route1 = 0
tictoc13.tic[1].route2 = 1
tictoc13.tic[1].route3 = 2
tictoc13.tic[1].route4 = 2
tictoc13.tic[1].route5 = 2

tictoc13.tic[2].route0 = 0
tictoc13.tic[2].route1 = 0
tictoc13.tic[2].route2 = 0
tictoc13.tic[2].route3 = 0
tictoc13.tic[2].route4 = 0
tictoc13.tic[2].route5 = 0

tictoc13.tic[3].route0 = 0
tictoc13.tic[3].route1 = 0
tictoc13.tic[3].route2 = 0
tictoc13.tic[3].route3 = 0
tictoc13.tic[3].route4 = 0
tictoc13.tic[3].route5 = 0
```

```
tictoc13.tic[4].route0 = 0
tictoc13.tic[4].route1 = 0
tictoc13.tic[4].route2 = 0
tictoc13.tic[4].route3 = 1
tictoc13.tic[4].route4 = 0
tictoc13.tic[4].route5 = 2

tictoc13.tic[5].route0 = 0
tictoc13.tic[5].route1 = 0
tictoc13.tic[5].route2 = 0
tictoc13.tic[5].route3 = 0
tictoc13.tic[5].route4 = 0
tictoc13.tic[5].route5 = 0
```

omnetpp.ini

TicToc13 Codes

```
class Txc13 : public cSimpleModule
{
  protected:
    long numSent;
    long numReceived;
    cLongHistogram hopCountStats;
    cOutVector hopCountVector;
    int rt[6];
    …
}

Define_Module(Txc13);

void Txc13::initialize()
{
  …
  // Initialize the routing table
  rt[0] = par("route0");
  rt[1] = par("route1");
  rt[2] = par("route2");
  rt[3] = par("route3");
  rt[4] = par("route4");
  rt[5] = par("route5");
…
}
```

```
void Txc13::forwardMessage(TicTocMsg13 *msg)
{
  // Increment hop count.
  msg->setHopCount(msg->getHopCount()+1);

  // Gate selection from routing table.
  TicTocMsg13 *ttmsg = check_and_cast<TicTocMsg13
*>(msg);
  int k = rt[ttmsg->getDestination()];

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