CINSim Mini-Howto

Quick Start Guide

Arvid Walter <asas AT cs.tu-berlin.de>
Matthias Kühm <kuehmi AT cs.tu-berlin.de>

Abstract

This document provides installation notes and a short introduction to the use of CINSim, a component-based simulator for interconnection networks. Starting with guidelines to the installation of all relevant packages, an example to the use of the CINSim tool chain will be given. For a more detailed information refer to the CINSim Howto.

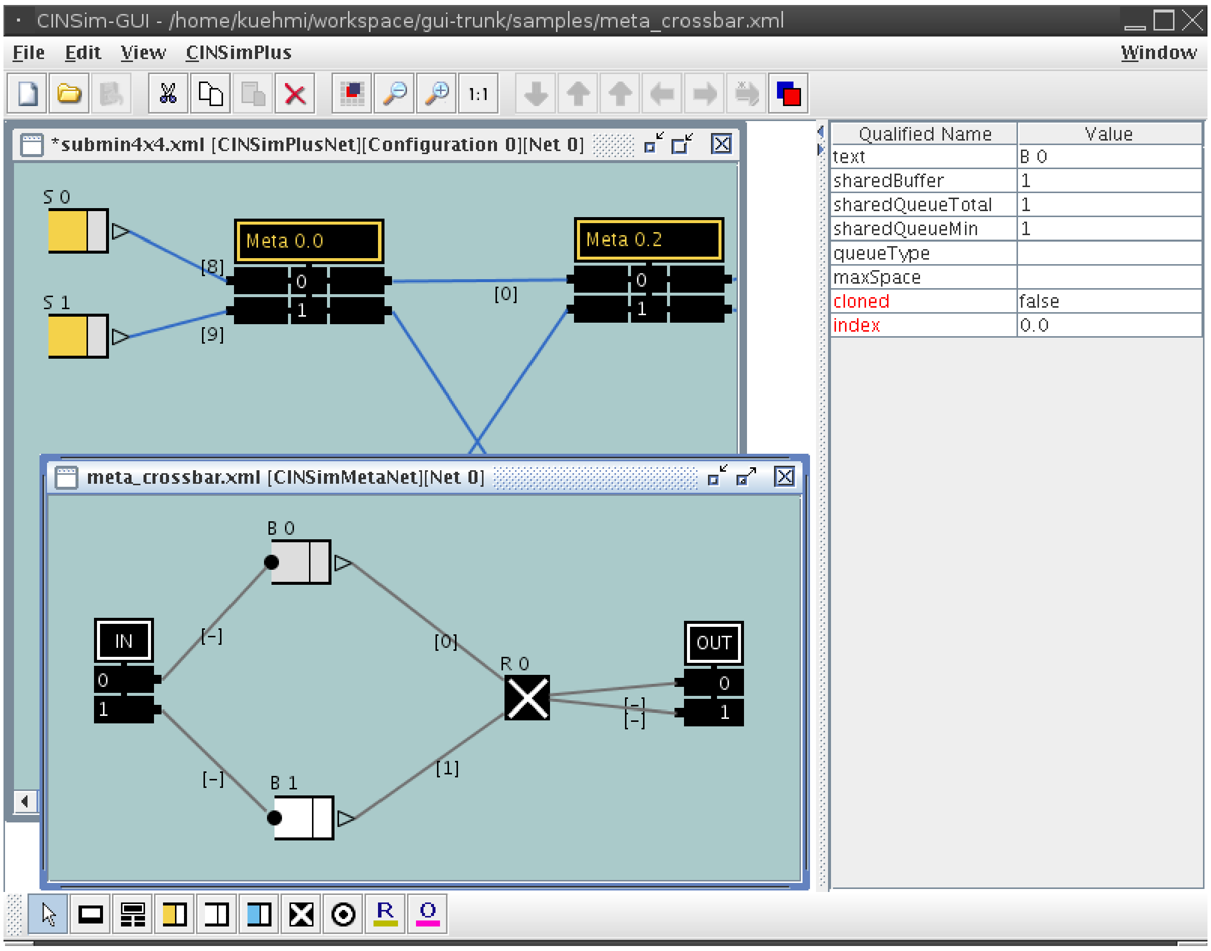
The simulator CINSim is component-based in many aspects. On the one hand, CINSim leaves most decisions related to the design of interconnection networks to be simulated to the user by providing atomic components such as buffers and switches that can be connected in many different ways. Furthermore, the logic of a network can be defined by several independent components including routing, switching, backpressure strategies and handling of redundancy. On the other hand CINSim is not an integrated simulation tool that combines graphical network design, simulation and analyzing in one application. In fact, the use of CINSim invokes several tools, command line and graphical ones.

Table of Contents

1. What Is the CINSim Project?

The CINSim project is located at the Bergische Universität Wuppertal (http://www.uni-wuppertal.de), Lehrstuhl für Automatisierungstechnik/Informatik. The project is targeted on the development of the simulator *CINSim* - **C**omponent-based **I**nterconnection **N**etwork **S**imulator - for Linux environments. In contrast to the former approach *MINSimulate* [11], which was limited to MINs and BMINs, this simulator is designed to simulate various types of network architectures based on atomic components such as switches and buffers. Furthermore, the simulation of dynamically reconfigurable interconnection networks is provided. In this approach, messages remain in the network buffers during a changement of the network behaviour or topology. A weaker approach is static reconfiguration, performed by dropping all remaining messages before switching to a new network configuration.

The simulation is performed by the *simulation core*, to be controlled at the command-line. The core is widely written in C++ and capable of executing performance analysis of regular and irregular interconnection networks with some boundary conditions to be satisfied. To speed up simulation runs, the simulation can be run in parallel on multi-core cpus and computer clusters. The simulation setup, including the network description, must be specified by an XML file based on an XML schema. For this purpose, the CINSim project provides a fully schema-driven editor that visualizes the XML document as shown in Figure 1 to easily describe interconnection networks. The editor is written in JAVA and therefore platform independent.



CINSim-GUI - /home/kuehmi/workspace/gui-trunk/samples/meta_crossbar.xml
File
Edit
View
CINSimPlus
Window
1:1
*submin4x4.xml [CINSimPlusNet][Configuration 0][Net 0]
S 0
S 1
Meta 0.0
Meta 0.2
[8]
[9]
[0]
0
1
meta_crossbar.xml [CINSimMetaNet][Net 0]
B 0
B 1
IN
OUT
R 0
[-]
[0]
[1]
R
O

Qualified Name	Value
text	B 0
sharedBuffer	1
sharedQueueTotal	1
sharedQueueMin	1
queueType	
maxSpace	
cloned	false
index	0.0

Figure 1. Simulation setup

The performance analysis of interconnection networks that can currently be executed using the simulator CINSim includes mean packet delay, mean queue length, mean flit delay, target and source throughput. These properties can be investigated using *steady-state* or *terminating* simulations. Confidence levels and estimated precisions are observed for each measure. If the desired termination criteria are met, the simulation will be stopped. The use of the Message Passing Interface (MPI) also allows to run multiple simulations in parallel on several workstations or on several cores of a multicore processor machine. CINSim sends itself via MPI routines to the desired number of processing nodes. While simulation is performed, the initiating CINSim process collects the intermediate results of all nodes and thus, controls the entire simulation.

The component-based approach of the simulator CINSim leads to a distinction of several network and simulation components, that can be put together in many ways. Network components are used to describe regular or irregular interconnection networks:

- **source buffers** - sources
- **non-shared buffers**
- **routers** - switches
- **target buffers** - destinations

The simulation of an interconnection network invokes several independent simulation components, that can also be set up using the provided XML editor. (Not every paradigm can be used in dynamic reconfiguration mode as well. To see further explanations, read Chapter 8)

- **routing strategies** - bitmask, shortest-path, xy- and west-first routing are currently supported
- **switching techniques** - packet switching including store-and-forward, virtual-/partial-cut-through and wormhole switching is supported
- **backpressure mechanisms** - local or global
- **terminating** and **steady state simulations**
- **scheduling algorithms** - to resolve routing conflicts
- **measure routines** - flit delay, target throughput, source throughput, mean queue length

More information to the work of the CINSim project can be found on the project's homepage (see Section A.1). A good overview to already achieved goals and future work can be found in [12] and [13].

2. Getting the Sources and Libraries

2.1. CINSim

You can download CINSim from the CINSim homepage (Section A.1) using the download section. There you will be asked for registration first. CINSim is free of charge for non-commercial use. Presently, CINSim supports only Linux platforms.

The currently offered way of installing CINSim is to use the tarball archives. You will have to download the archives *cinsim-0.6.tar.gz* and *cinsim-gui-0.6.tar.gz* and execute some installation steps to be explained in Section 3.1.

Release Information

The jar archives of the GUI that come with the tarball archive, are compiled using Sun Java 1.5. We recommend to install and use this version of Java in conjunction with CINSim. There have also been problems reported with some versions of Sun Java 1.5, which led to a crashing GUI. We recommend therefore to use the latest version available (currently 1.5.0_10-b03).

The compiling of the core applications should work on systems providing the GNU C/C++ compiler (>=3.3), GNU Make (>=3.80) and GNU Libtool (1.5.x). This usually implies that proper C/C++ libraries are available, too. Otherwise check for libc and libstdc++.

2.2. MPI

The simulation core of CINSim uses functions of the MPI (*Message-Passing Interface*) specification to execute simulations on multiple hosts. This is an optional feature, that requires an implemention of at least the MPI 1.1 standard to be available.

MPI implementations usually offer special versions of C/C++ compilers to set the required compile flags and add libraries to be linked. The installation routine of CINSim searches for known variants of the compiler commands and libraries. Most Linux distributions offer packaged MPI implementations.

Release Information

CINSim was tested using MPICH 1.2.7 and LAM/MPI 7.1.1 , but should work with other implementations, too. See Section A.2 for more information and download. Independent from the implemenation that you actually use, you will need the library, the development (header) and runtime files (compiler, execution programm). To use MPICH on multicore computers use the MPICH Device: ch_shmem.

2.3. GNU Scientific Library

The statistical evaluation of simulation requires sophisticated mathematical procedures that are povided by the GNU Scientific Library. A packaged version should be available for any Linux distribution, check for *gsl*. Fore more information see Section A.3.

Release Information

CINSIm was tested with several versions >= 1.7 of the library. You will need the library and the development files.

2.4. Xerces-C++

Network descriptions are passed to CINSim by using the XML language and the simulation program needs therefore a validation mechanism to check the contents of input files. For this purpose, CINSim uses Xerces-C++, a validating XML parser library written in a portable subset of C++.

You can download Xerces-C++ from the homepage of the *Apache Software Foundation* (Section A.4) or check the list of packages for your linux distribution.

Release Information

You need at least version 2.4 of Xerces-C++, but versions up to 2.7.0 were tested with CINSim.

Important

Most Linux distributions that provide Xerces-C++ packages distinguish between library and development packages. You should search for libxerces or xerces-c. To compile the CINSim core application from the tarball archive you will need the library and the development files of Xerces-C++. If you have successfully compiled Xerces-C++ from source on your own, you have all you need.

2.5. Bison++

Bison++ is only needed for maintainers use.

Tip

Return to this section if compiling of CINSim fails because of **libmathparser.a**. Actually, bison++ is needed for developers only, because the parts that need bison++ are already compiled, even in the tarball archive.

CINSim makes use of a very simple language to define complex formulas to describe the shape of distributions. Bison++ is able to generate C++ code using a grammar. The intention is to get a C++-class that is able to interpret a formula given to the simulator at runtime.

Release Information

You need to have bison++ version 1.21.9 or later installed if want to compile the formula parser. There is usually no need to do that. You can download a tarball archive (Section A.5) or search for Debian or RPM packages.

3. How to Configure, Compile and Install

The following sections describes how to install CINSim using the tarball archives. It will be assumed that you have already downloaded the related installation packages following Section 2 to a specific directory and all the libraries needed are properly installed.

3.1. CINSim Tarball Archives

The installation of CINSim using the tarball archives should work on almost any somewhat up-to-date Linux platform. Before installing the core application it is necessary to install the GUI.

GUI

Change to the directory that holds the tarball archive *cinsim-gui-0.6.tar.gz* and type

```
tar xzvf cinsim-gui-0.6.tar.gz -C INSTALLDIR
```

This will create the directory *INSTALLDIR/cinsim-gui-0.6*, in the following called *GUIPREFIX*. You will need write access to the installation directory. After extracting the tarball archive change to the directory *GUIPREFIX* and type

```
./install
```

This will create the start script *cinsim-gui* located in *GUIPREFIX/bin*.

Core Applications

Change to the directory that holds the tarball archive *cinsim-0.6.tar.gz* and extract the sources by typing:

```
tar xvzf cinsim-0.6.tar.gz
```

This will create the directory *cinsim-0.6* in the current directory. Change into this to start the installation process. The first step is to create a Makefile working on your system. For this purpose, you have to call the *configure* script invoking some options. The options of interest are given in Table 1 and regard to various installation paths and directives needed during the installation process. If you want to check out all the available options type

```
./configure --help
```

option	description
--prefix=PATH	This parameter sets the installation path. Within the specified directory the sub directories *bin*, *doc*, *include*, *lib* and *share* will be created or used. If the specified directory does not exist, it will be created. Make sure you have write access for using or creating the directory. The default value for *PATH* is */usr/local*.
--with-gui=GUIPREFIX	Use this parameter with the cinsim-gui installation path to create an appropriate configuration file for cinsim.
--with-xerces-prefix=PATH	This option sets the *Xerces-C++* path. Configure will create, compile and run a small sample program to check if Xerces-C++ is running fine. You need to use this option only if the header-files and libraries of Xerces-C++ are unknown to your compiler. In case of this the macro assumes that the missing headers and libraries are to be found in the includes and lib directories within the specified Xerces-C++ folder.

option	description
--with-gsl-prefix=PATH	This option is similar to the one above but can be used to specify the path to the GNU Scientific library if not found by the configure script.
--enable-mrip	This enables to optional MRIP support of CINSim. For this option to take effect you need an MPI implementation installed on your system. Check the output of the configure script, to see whether an MPI implementation was found.
--enable-meshes	CINSim is capable of simulating regular mesh structures under quite restrictive constraints. This has to be seen as an experimental feature, not intended for massive use. If you want to analyze meshes you need to use this option to install the plugins for XY- and West-First-Routing.
--enable-asserts	If you encounter runtime errors, such as segmentation faults, this option can be used to check whether certain assertions are violated. You might use this option for creating error reports.

Table 1. Configure script options

You will most likely want to set the installation path and the location of the GUI. Assuming that you want to install to *PREFIX*, for example */usr/local/cinsim-0.6*, and the GUI has been installed to *GUIPREFIX* type

```
./configure --prefix=PREFIX --with-gui=GUIPREFIX
```

If the configure scripts terminates successfully, a file named *Makefile* will appear in the current directory. Otherwise, if the configuration ends up in error, the last line of output shows an error message. In most cases missing libraries or headers will be reported. You can use these information to install missing packages or specify the correct paths. In case of Xerces-C, for example, the latter can be done using the configure option given in Table 1.

After finishing the configuration process it is time to compile the core applications by typing

```
make
```

This will compile the sources provided by the tarball archive and should work without any problems if the configuration process succeeded. After compiling type

```
make install
```

or

```
make install-strip
```

to finish the installation process. At this point you will need write access to the specified installation directory or it will fail. Calling *make install-strip* removes debug code before installing the application binaries. After successfully finishing the installation process the program binary *cinsim* is located in *PREFIX/bin* .

A final check on the installation can be done using some unit tests. For these tests you need CppUnit installed (library and headers). Check the output of the configure script to see if a proper version was found. Then simply type

```
make check
```

This will compile the test suites, run tests on selected classes and deliver some test statistics, hopefully showing that all tests passed. If not, there is something wrong with compiled instance of CINSim on your system, which may lead to undefined behaviour in simulation runs and useless results.

Setting the paths

Depending on your chosen installation directories it might be necessary to add the paths to the installed binaries manually to your PATH variable for immediate access. This can be done by typing

```
export PATH=PREFIX/bin:GUIPREFIX/bin:$PATH
```

in your shell. The placeholders *PREFIX* and *GUIPREFIX* must be replaced by the installation paths of the core applications and the GUI. However, you will have to repeat this step every time you open a shell to work with CINSim. To avoid this overhead, you can add the same line to the *.bashrc* (if you are using a bash shell) file located in your home directory. Every time you open a shell, the PATH variable will then be properly set.

To check whether the PATH variable is set properly type

```
which cinsim (cinsim-gui)
```

in your shell. This should lead to an output like

```
/usr/local/cinsim-0.6/bin/cinsim
```

which should match with your installation directory.

4. Defining the Simulation Set

4.1. Scenario

This tutorial will discuss a simple multistage interconnection network (MIN) with two stages, four sources and four targets. A *steady-state* simulation sequence with an increasing traffic load up to 100 percent, *bit-mask routing*, *virtual cut-through switching* and *round-robin scheduling* will be defined. The constant size of all packets passed through the network shall be two. In every single clock cycle only a packet fragment (flit) of size one can be transmitted and stored. All buffer elements shall have a capacity of two. The packet delay within the network and the throughput of some of the sources will be analyzed.

Note

If you are familiar with one of the other available tutorials you may skip Section 4.2 and proceed with Section 4.3.

4.2. First Steps

Depending on your installation just type **cinsim-gui** to start the GUI or first move to the installation directory where the start script is located. This command will bring up the initial GUI window. Move your mouse cursor to the menu bar at the top of the window and select the only available entry *File*. The menu shown in Figure 2 appears. Select the first entry to create a new XML file defining a simulation set.

Now a dialog appears, asking you to choose a net class using a combo box (Figure 3). Usually the net class *CINSimPlus* is preselected. Hit *Ok* to proceed. If there is no entry available from the combo box, the button will be deactivated and you will first have to specify a correct schema location path as described in the **User's Guide**. Hitting *Cancel* will abort creating a new file.

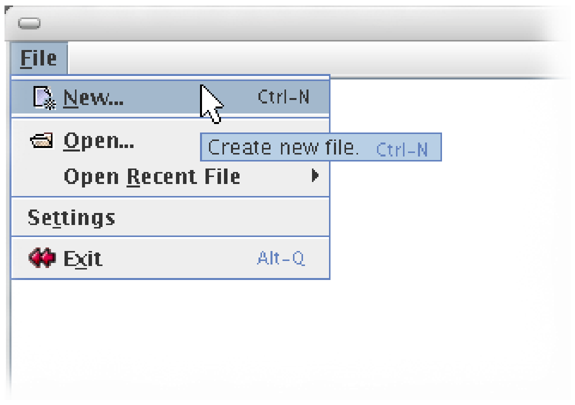


Figure 2. Creating a new file

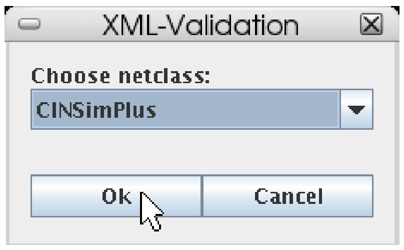
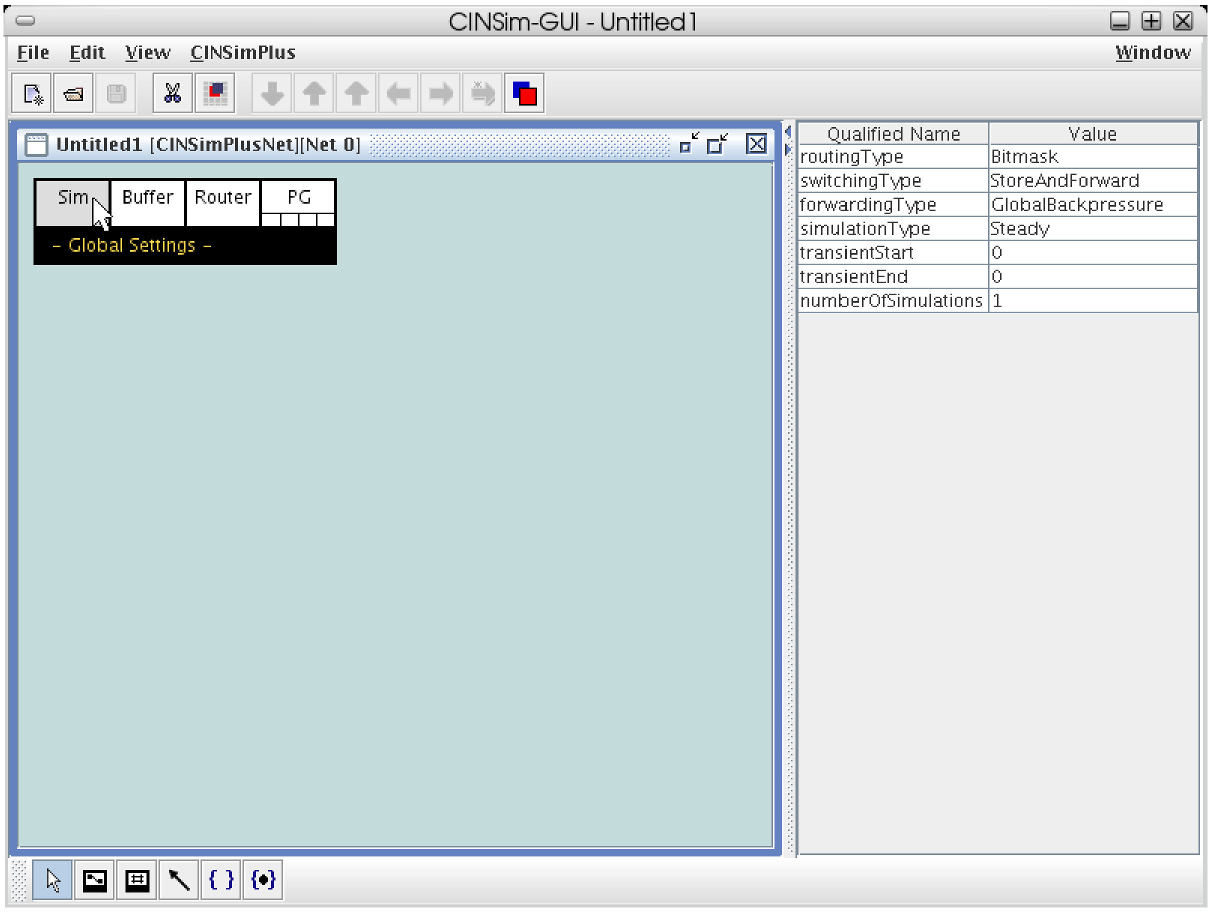


Figure 3. Choosing a net class

The appearance of the editor window changes after choosing the net class CINSimPlus as shown in Figure 4. The menu bar holds new entries and two symbol bars are added. The main part of the window holds a new frame, in the following called editor frame, that shows the newly created simulation set. The selected net class defines a specific view on XML files designed for interconnection networks. An editor frame shows a part of the XML tree that starts from a special element, defining a net. A net can have subelements representing nodes, connectors and labels. Objects shown in the editor frame represent elements in the created XML document. All changes, like adding new objects, will directly affect the underlying XML structure. Selecting an object will bring up a table holding the element's attributes and their current values on the right side of the editor window.


CINSim-GUI - Untitled1
File Edit View CINSimPlus Window
Untitled1 [CINSimPlusNet][Net 0]
Sim
Buffer
Router
PG
- Global Settings -

Qualified Name	Value
routingType	Bitmask
switchingType	StoreAndForward
forwardingType	GlobalBackpressure
simulationType	Steady
transientStart	0
transientEnd	0
numberOfSimulations	1

Figure 4. New simulation set

Tip

You can modify the size of the areas holding the editor frames and the attribute tables by using the divider between them. Use the nested arrows to minimize or maximize one of the areas.

The menu bar at the bottom of the editor window, called insertion tool bar, holds all insertable elements of the currently displayed net element. These elements are specified by the XML schema of the net class CINSimPlus, so insertions will not lead to invalid XML documents. To insert an element to the editor frame, you have to select its symbol from the insertion tool bar. Then move the mouse cursor to the position the new object is to be placed and left-click to insert. As long as the element's symbol is selected, left-clicking within the editor frame will insert new objects. This is called *insertion mode*. To leave this mode, you have to select the first symbol of the insertion tool bar as shown in Figure 5. The editor will return to the normal *selection mode*. This mode allows you to select objects within the editor frame and bring up their attributes. You can select more than one object by holding the ctrl-key or using a selection rectangle: Left-click the editor frame, hold down the button and move the mouse so that all objects to be selected are within the opened rectangle. Releasing the mouse button will select these objects. You can remove an element from a selection by holding the ctrl-key and left-clicking the object to be removed.

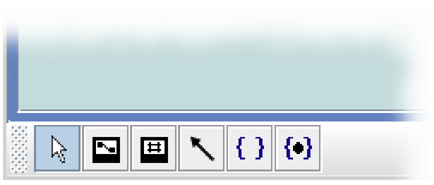


Figure 5. Selection mode

While in selection mode, you can also move objects. Select an object with the left mouse button, hold down the button and move the mouse cursor. Release the mouse button when the desired position is reached. If you select an element from a multiple selection, the whole selection will be moved.

4.3. Defining Variables

There are two different types of variables for CINSim. One type, called *CINSim variable*, is used to vary parameters within a simulation sequence, the other type, called *Akaroa variable*, defines variables observed by Akaroa during simulation runs. Akaroa stops a simulation run when all defined Akaroa variables reached steady states.

For our scenario we will need a CINSim variable to increase the traffic load during the simulation sequence (see Section 4.4) and two Akaroa variables to observe the throughput of the sources and the average delay of packets that passed through the network (see Section 4.6).

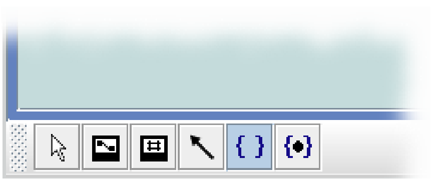


Figure 6. CINSim variable selection

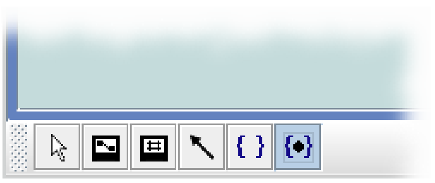
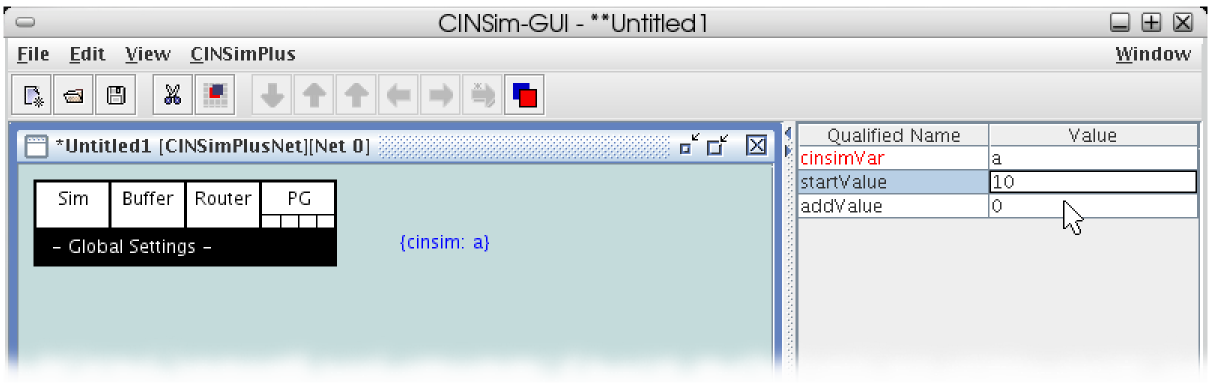


Figure 7. Akaroa variable selection

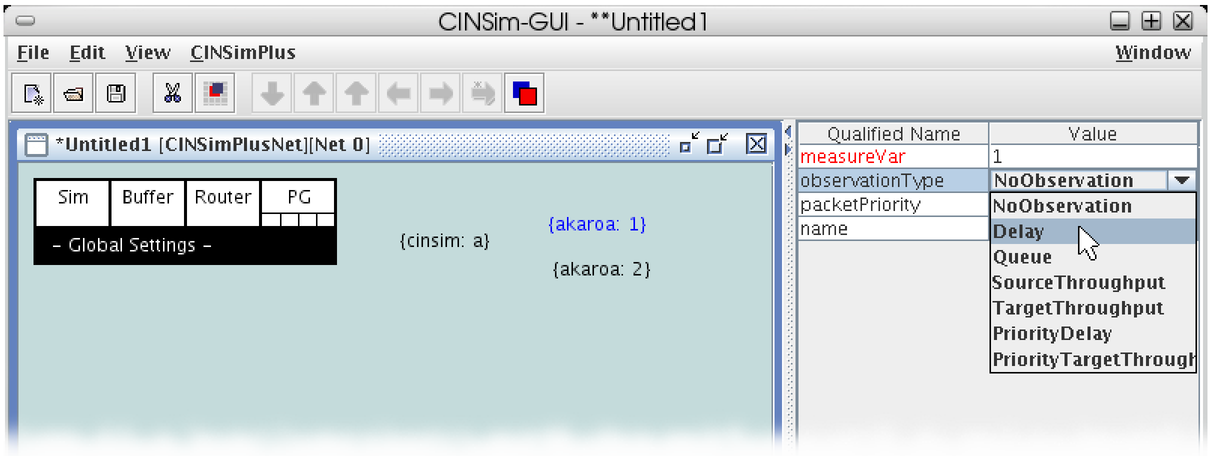
Select the symbol related to a CINSim variable from the insertion tool bar as shown in Figure 6 and place a variable within the editor frame. Then return to the selection mode and select the newly inserted variable description. The attribute editor now shows the attributes belonging to the underlying XML element as shown in Figure 8. The first attribute defines the name of the variable. CINSim variables are automatically named with lower case letters according to the order they are inserted. The attribute value can not be modified. With the remaining attributes you can define a start value and an increment. Only positive integers are allowed. Double-click each value field to bring up an editor prompt and enter the number *10*. This will define the variable *a* to start with the value *10* and each incremental step will raise the current value by *10*.



Qualified Name	Value
cinsimVar	a
startValue	10
addValue	0

Figure 8. CINSim variable setup

Now select the symbol related to an Akaroa variable as shown in Figure 7 and insert two Akaroa variables. After returning to the selection mode, select the first one. Akaroa variables are automatically numbered with integers beginning with number *1*. Now you have to set the type of observation for the selected variable. Clicking the value field next to the attribute *observationType* will turn the value field into a combo box. Select the entry *Delay* as shown in Figure 9. The remaining attributes need no modification. Proceed with the second Akaroa variable and choose *SourceThroughput* instead of Delay.



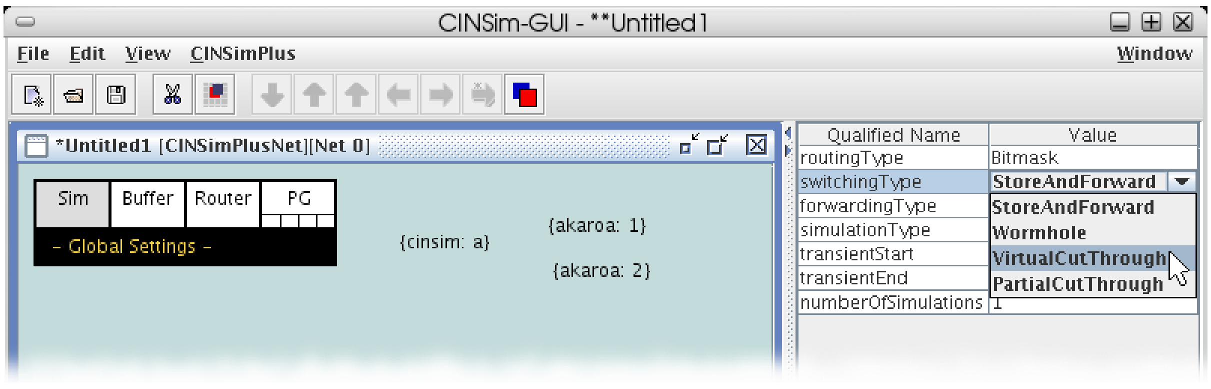
Qualified Name	Value
measureVar	1
observationType	NoObservation
packetPriority	
name	

Figure 9. Akaroa variable setup

4.4. Global Settings

The editor holds a special object in the upper left corner, labeled with *Global Settings*. This menu, strictly speaking its related XML element, is created by default and can neither be moved nor deleted. It is a container for several elements defining basic simulation settings.

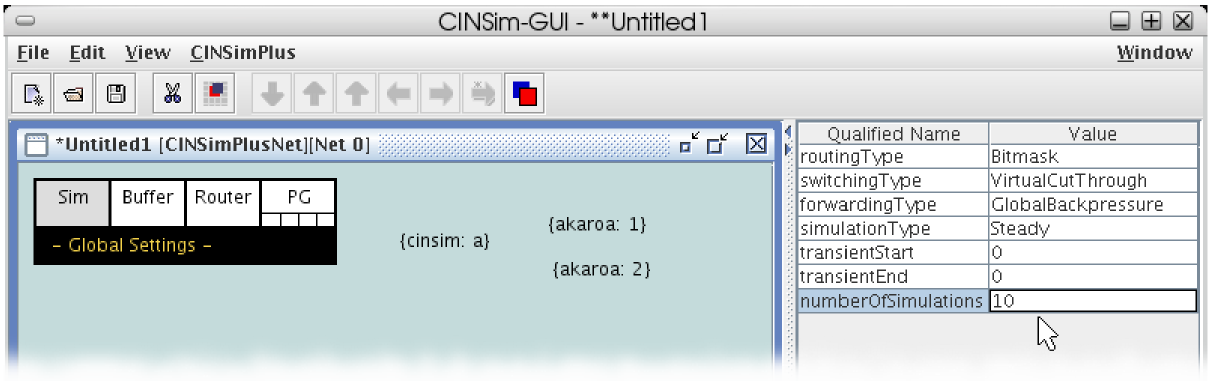
The first field's attributes define general simulation settings. Select the white area labeled with *Sim* by left-clicking. The attribute editor on the right now shows the available attributes. We need to set two of them. Select the value field next to the attribute *switchingType* and choose the entry *VirtualCutThrough* from the appearing combo box as shown in Figure 10. This will set the simulation's switching strategy to *virtual cut-through*. The remaining attributes ending up with *Type* specify a *steady* simulation with *global backpressure* and routing according to the components' *bit masks*.



Qualified Name	Value
routingType	Bitmask
switchingType	StoreAndForward
forwardingType	StoreAndForward
simulationType	Wormhole
transientStart	VirtualCutThrough
transientEnd	PartialCutThrough
numberOfSimulations	1

Figure 10. Setting up switching strategy

The last attribute *numberOfSimulations* is used to define simulation sequences. The default value is *1*, specifying a single simulation run. Setting this value to any other non zero integer defines a sequence. Double-click the value field and enter *10* according to Figure 11. This sets up a simulation sequence with ten simulation runs. The CINSim variable *a* will be increased by 10 after each run, starting with 10 and ending up with 100 (see Section 4.3).

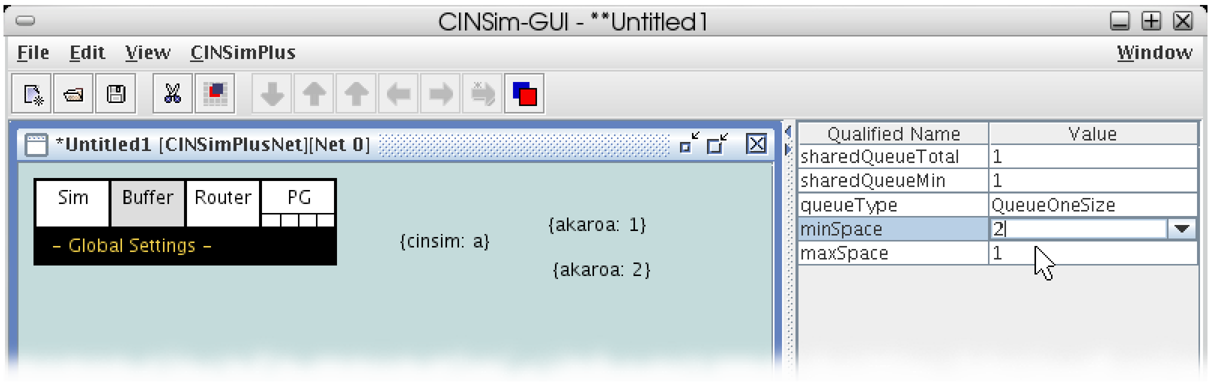


Qualified Name	Value
routingType	Bitmask
switchingType	VirtualCutThrough
forwardingType	GlobalBackpressure
simulationType	Steady
transientStart	0
transientEnd	0
numberOfSimulations	10

Figure 11. Setting up simulation sequence

After setting up the simulation itself, the next steps will set up the network components to be added in Section 4.5. The remaining fields within the settings menu hold attributes defining common properties of all network components to be added. In contrast to the simulation settings, most of these settings can be redefined by the components' attributes.

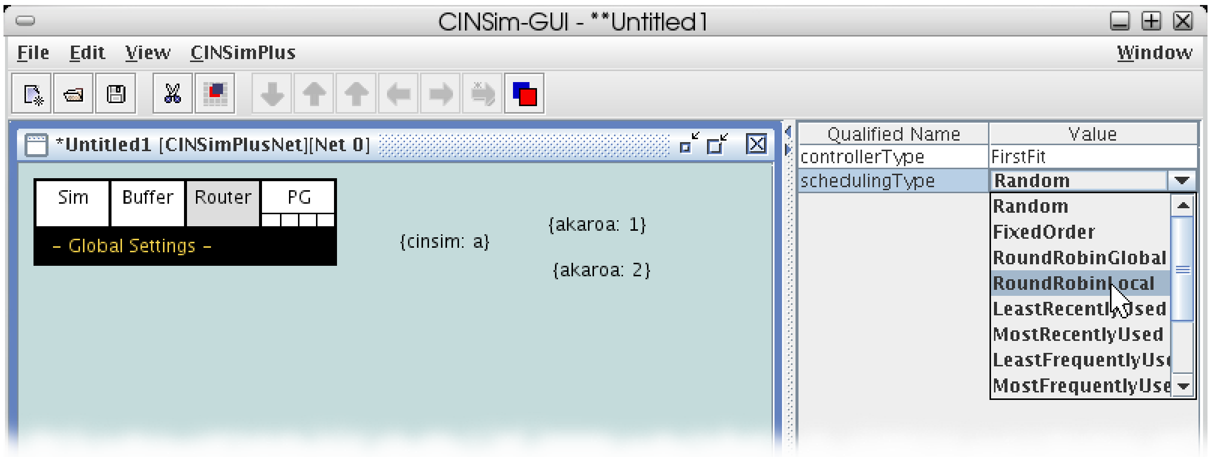
Select the area labeled with *Buffer* to bring up its attributes related to the buffer components of our network. For the given scenario we need to set the space provided by buffers for storing packet fragments. Select the value field next to *minSpace*, ignore the combo box and enter *2* (Figure 12). This sets the lower bound for the available buffer space. The attribute *maxSpace* sets the really available space. Values smaller than the lower bound will be replaced by the lower bound, so leave it unchanged. The lower bound is fixed up here, each buffer component can only redefine the attribute *maxSpace*. All available switching strategies (Figure 10), except *Wormhole*, assume that all buffer components of the network can store generated packets completely. Setting the attribute *minSpace* to a value greater or equal to the size of the generated packets (Figure 14) assures that the assumption will be fulfilled without checking all buffers one by one.



Qualified Name	Value
sharedQueueTotal	1
sharedQueueMin	1
queueType	QueueOneSize
minSpace	2
maxSpace	1

Figure 12. Setting up buffer size

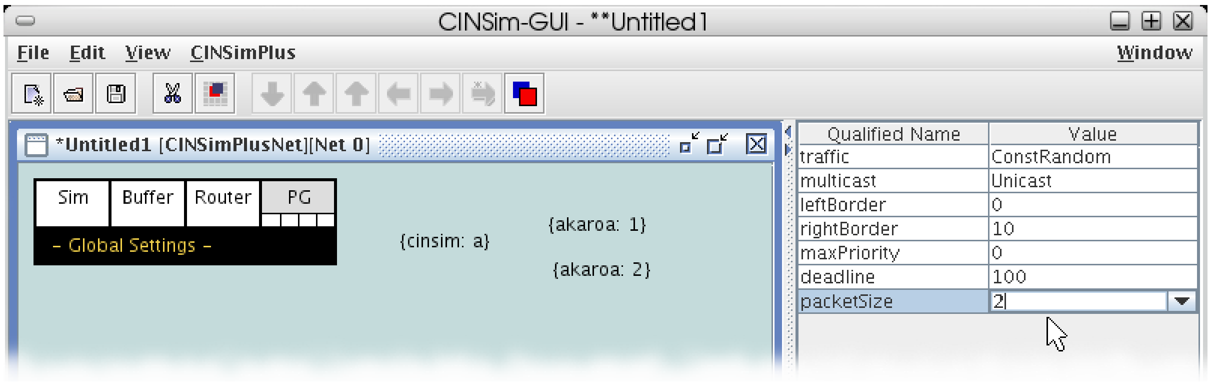
The menu entry labeled with *Router* can be used to set up common properties for router components. In this tutorial we want to set the scheduling strategy to *round-robin*, so select the menu entry by left-clicking. Proceed with selecting the value field of the attribute *schedulingType* and choose *RoundRobinLocal* from the combo box as shown in Figure 13. Ingoing and outgoing wires (inputs and outputs) of router components are numbered in ascending order. *Global* round-robin checks for any given output the available inputs for packets to be transmitted, starting with the **second** input according to the order of the last clock cycle. *Local* round-robin starts with the **next** input following the input actually used in the last tick.



Qualified Name	Value
controllerType	FirstFit
schedulingType	Random

Figure 13. Setting up scheduling type

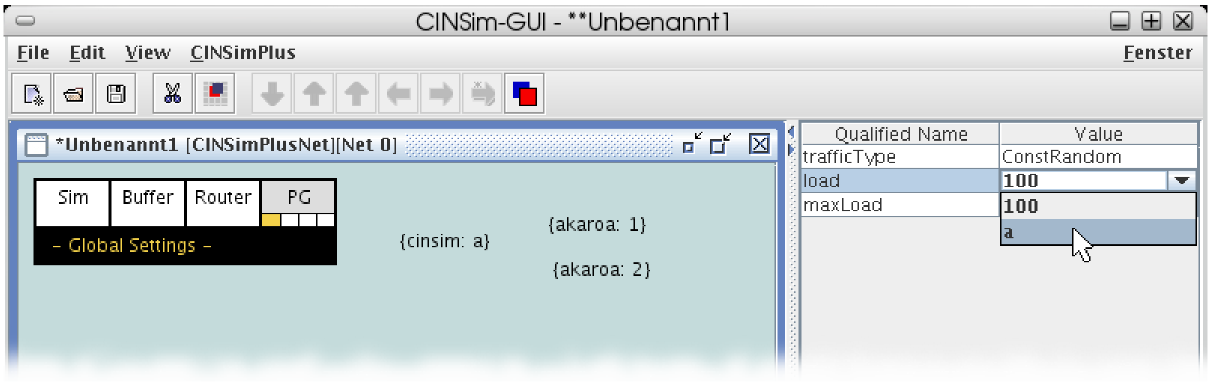
The last menu entry labeled with *PG* holds attributes defining properties of source buffers. We want all the sources to generate packets of size two. Follow Figure 14 and select the area in the editor frame. Then enter *2* within the value field next to the attribute *packetSize*.



Qualified Name	Value
traffic	ConstRandom
multicast	Unicast
leftBorder	0
rightBorder	10
maxPriority	0
deadline	100
packetSize	2

Figure 14. Setting up packet size

In the menu entry dedicated to packet generators, there are four small boxes that again hold several attributes. These attributes can be used to set up different types of traffic distribution strategies, that determine when packets are to be generated. Look at the first attribute in Figure 14. This sets up all packet generators to be inserted to use the *constant-random* distribution. Select the first box as shown in Figure 15. The first attribute *trafficType* is a label, indicating that the remaining attributes can be used to set up the constant-random distribution. We want to increase the generated traffic during our simulation sequence. Select the value field next to the attribute *load*. The displayed combo box holds the current value and all the CINSim variables we defined. Proceed by selecting the variable *a*.


CINSim-GUI - **Unbenannt1
File Edit View CINSimPlus Fenster
*Unbenannt1 [CINSimPlusNet][Net 0]
Sim
Buffer
Router
PG
- Global Settings -
{cinsim: a}
{akaroa: 1}
{akaroa: 2}
100
a

Qualified Name	Value
trafficType	ConstRandom
load	100
maxLoad	

Figure 15. Setting up traffic load

4.5. Net Configuration

The most important element a simulation set must contain is, of course, a network description. CINSim supports static reconfiguration, therefore net descriptions are nested within special elements, called *net configurations*, which can be connected to define reconfiguration steps.

For the given scenario only one net configuration is needed to describe a 4x4 MIN. Select the symbol related to net configurations from the insertion tool bar as shown in Figure 16, place an object within the editor frame and return to the selection mode.

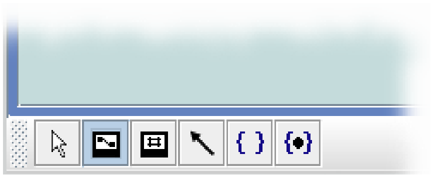
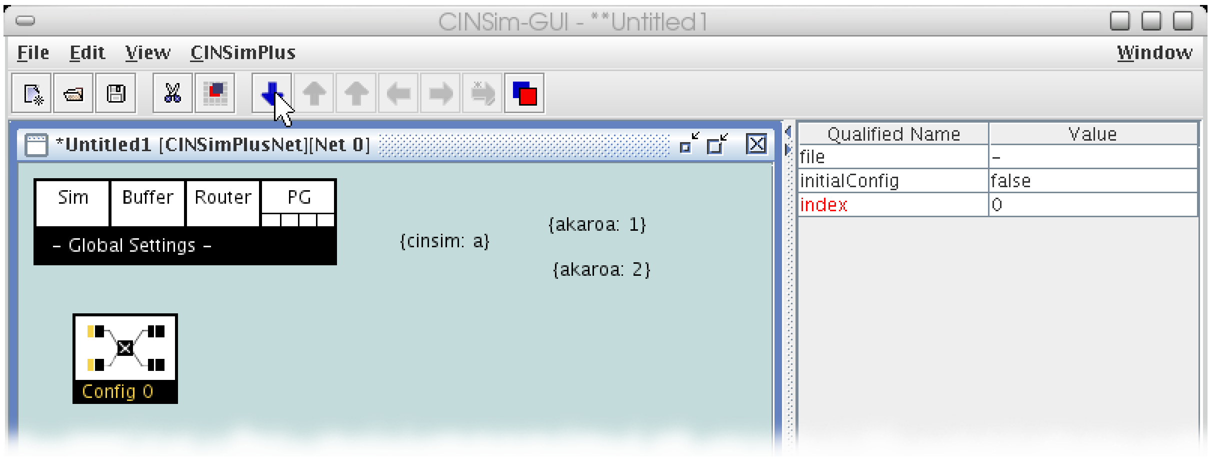


Figure 16. Net configuration selection

We need to descend into the net configuration to edit the nested net description. Selecting the inserted object with the left mouse button will activate the descend button within the symbol bar at the top of the window. Left-click the button (Figure 17) to descend. The editor frame then shows the nested net element and the insertion tool bar the net components that can be inserted.

Note

You can also double-click the inserted object or select the related entry from the menu *CINSim-Plus* to descend into a net configuration.



Qualified Name	Value
file	-
initialConfig	false
index	0

Figure 17. Descending into net configuration

Now we can start to layout our network. Select the symbol related to packet generators from the insertion tool bar as shown in Figure 18. We need four of them, so place four objects one below the other in the left half of the editor frame. We will later need some space on the left to the inserted objects to set up the measurements, so place them not too close to the left border of the editor frame.

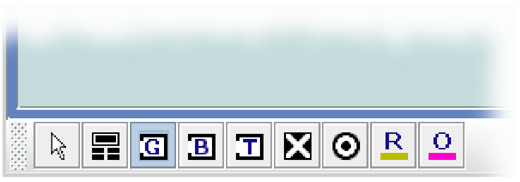


Figure 18. Packet generator selection

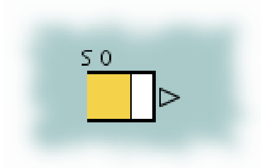


Figure 19. Packet generator object

The two stages of the 4x4 MIN shall consist of four 2x2 crossbars with input buffering. A 2x2 crossbar with buffers is not available from the insertion tool bar and must be defined using basic components: Two buffer components followed up by a router component. We could insert and connect these components straight forward for all stages and proceed, but there is a nice feature called *meta element* that can speed up the use of complex components, especially for larger networks. A meta element can be treated like any other basic component, but contains a nested net description defining its behaviour. Start by selecting a meta element from the insertion tool bar (Figure 20), place one on the right to the packet generators and return to the selection mode.

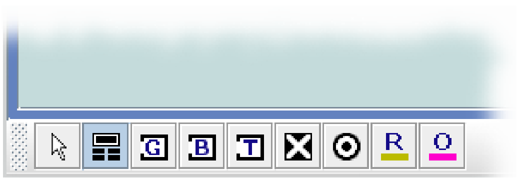


Figure 20. Meta element selection

Now you can select the inserted object and use the highlighted descend button to bring up its nested net description (Figure 21).

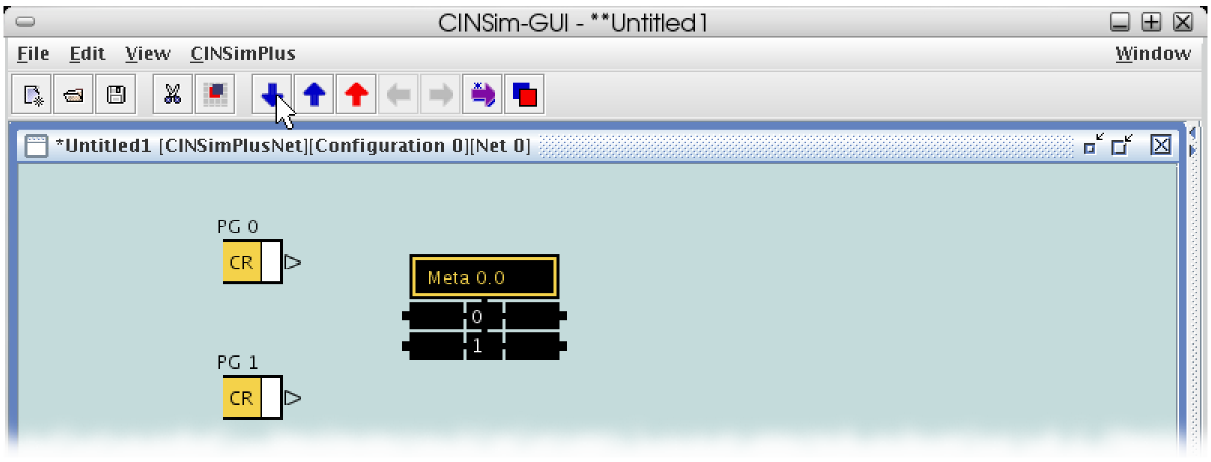


Figure 21. Descending into meta element

The only difference between net descriptions within net configurations and meta elements are the panels, labeled with *IN* and *OUT*, that allow components to be connected to components from outside using the numbered connection points. Figure 22 shows both panels, allowing several ingoing and outgoing connections. You can move the panels to any desired position, but you can not remove them. The maximum number of connections can be set up to 16 by setting the attribute *dimension* of the meta element.

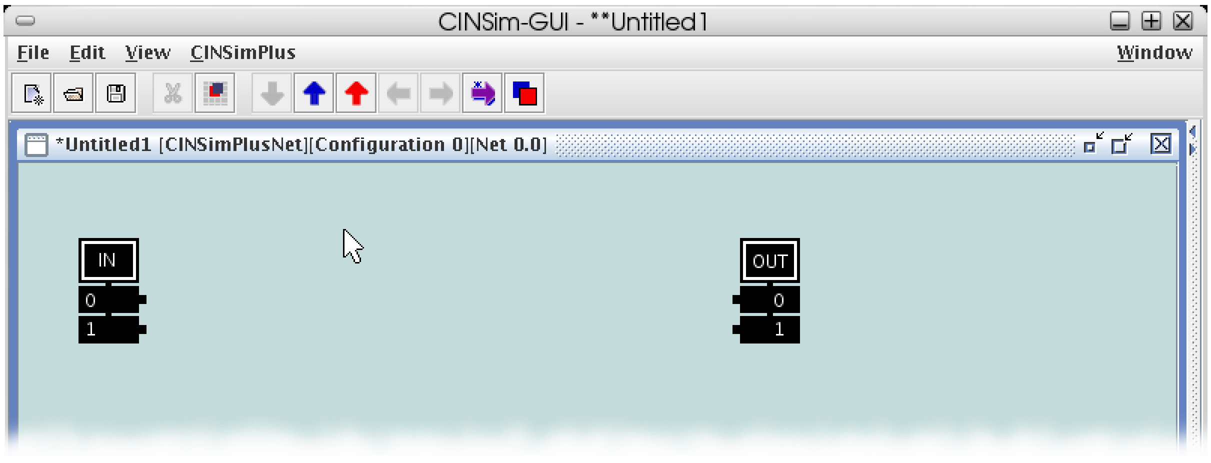


Figure 22. Empty meta net

Within this meta element we now want to describe a buffered crossbar. Select the symbol related to buffers from the insertion tool bar (Figure 23) and place two of them one below the other. Then select the router symbol (Figure 25) and insert a router on the right to the inserted buffers. The next step is to connect the inserted components. A directed connection between net components for sending and receiving packets is called *route*. A route always starts from an output connection point and ends up at an input connection point. A buffer object (Figure 24) has an input connection point on the left (circle) and an output connection point on the right (lace). These symbols are also used for generators (Figure 19) and targets (Figure 33). A router object (Figure 26) combines both types of connection points and can therefore be start and end of a route. Meta elements have their inputs on the left and their outputs on the right. Incoming routes within meta elements start at the IN panel and outgoing routes end up at the OUT panel.

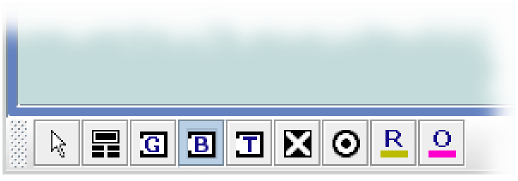


Figure 23. Buffer selection

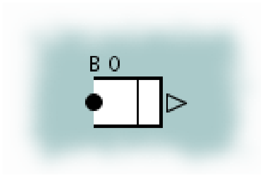


Figure 24. Buffer object

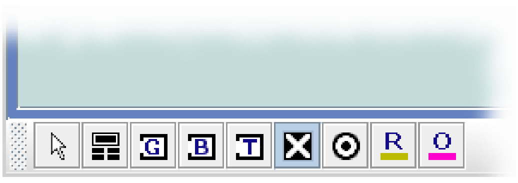


Figure 25. Router selection

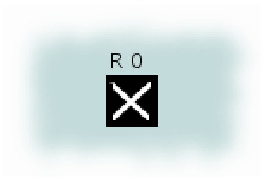


Figure 26. Router object

Select the symbol belonging to routes from the insertion tool bar as shown in Figure 27. Inserting a route starts by selecting the connection point to start (Figure 28). Hold down the left mouse button and move the cursor to the connection point where the route shall end. A thin line will follow the mouse cursor representing the new connection. If not, you missed the start point or the start point is invalid. Releasing the mouse button at the desired position will insert the new route (Figure 29), if the end point is valid. Proceed by connecting the components as shown in Figure 30.

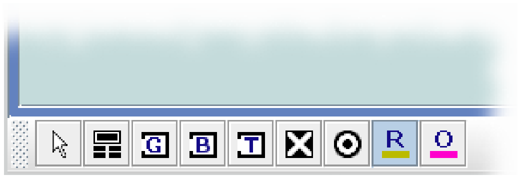


Figure 27. Route selection

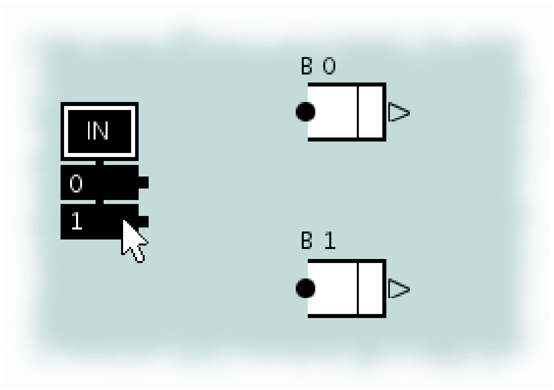


Figure 28. Inserting new route (1)

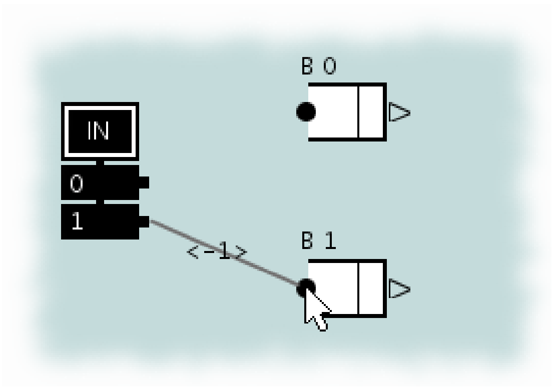


Figure 29. Inserting new route (2)

The inscriptions along the created routes are automatically generated according to the insertion order. Routes connected to the IN and OUT panels will get the inscription *-1* if there is no route connected to the related connection point of the meta element. Otherwise the inscription shows the (full) index of the connected route.

After connecting the components, we can leave the nested net description. Descending into a net configuration or a meta element activates two buttons within the upper symbol bar. These buttons, showing arrows pointing upwards, can be used to ascend from a nested net description. Click the blue one (Figure 30) to ascend from the currently displayed net. The red one can be used to quickly ascend to the top level.

Note

You can also select the related entries from the menu *CINSimPlus* to ascend or use the shortcuts defined there.

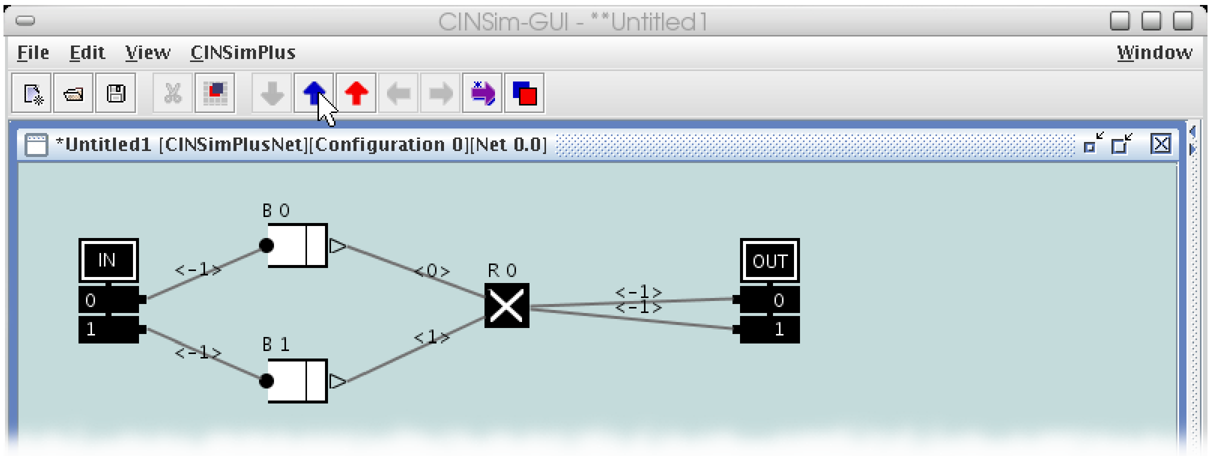

CINSim-GUI - **Untitled1
File Edit View CINSimPlus Window
*Untitled1 [CINSimPlusNet][Configuration 0][Net 0.0]
IN
0
1
B 0
B 1
<-1>
<-1>
<0>
<1>
R 0
<-1>
<-1>
OUT
0
1

Figure 30. Ascending from meta net

Now we can duplicate the created meta element. Mark the object in the editor frame and open the menu *Edit*. This will bring up the entries shown in Figure 31. Select *Copy* to copy the crossbar description. Then use the entry *Paste* to insert three copies. Place them in two columns to describe the two stages of the MIN.

Tip

You can speed up *copy-paste* actions by using the given shortcuts.

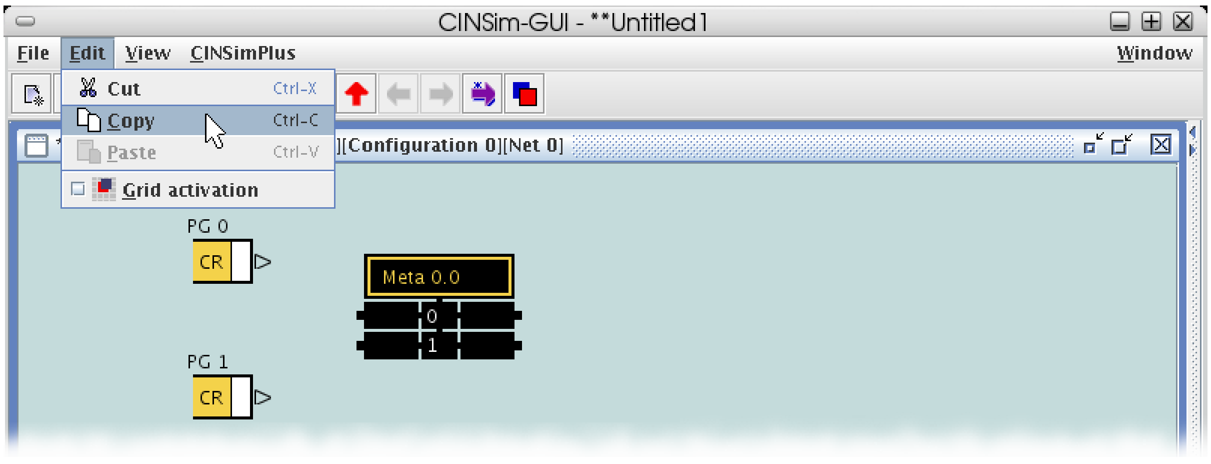


Figure 31. Copying meta element

Targets are special buffer components that just analyze and then erase incoming packet fragments. Select the related symbol from the insertion tool bar (Figure 32) and insert four of them in a column on the right side of the editor frame.

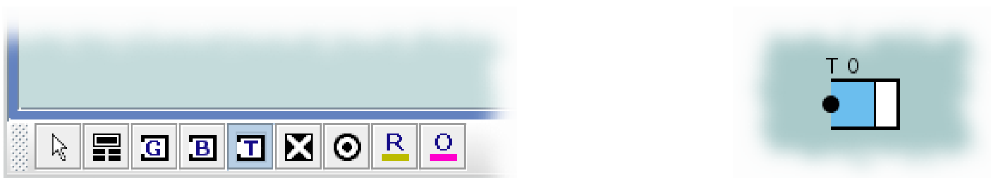


Figure 32. Target selection

Figure 33. Target object

The net configuration is now almost complete. We only need to connect the inserted components using routes as shown in Figure 34.

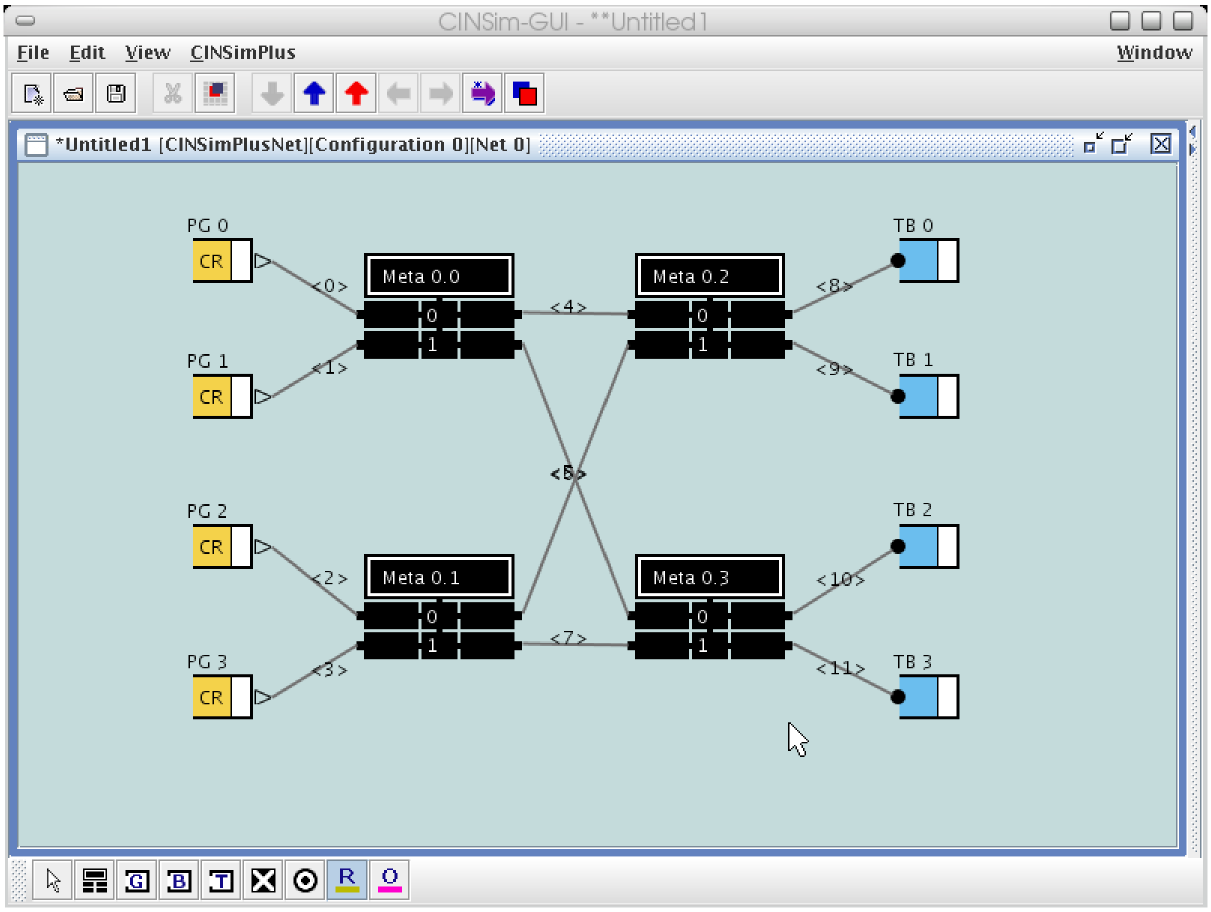


Figure 34. Configuration without observations

4.6. Adding Analyzers

Simulation runs with CINSim will not lead to results unless at least one measurement is defined. The measured values are written to special variables observed by Akaroa. Defining such Akaroa variables (Section 4.3) specifies the type of measurement, but not what components are to be observed.

The insertion tool bar for net descriptions provides a special element, called *analyzer*. An analyzer can be seen as a placeholder for an Akaroa variable. Select its symbol as shown in Figure 35 and place one in the upper right corner of the editor frame. Then return to the selection mode and select the newly inserted object.

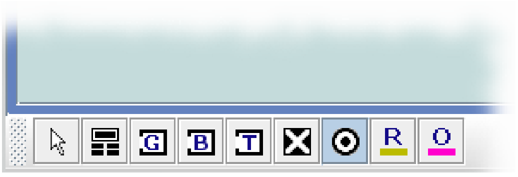
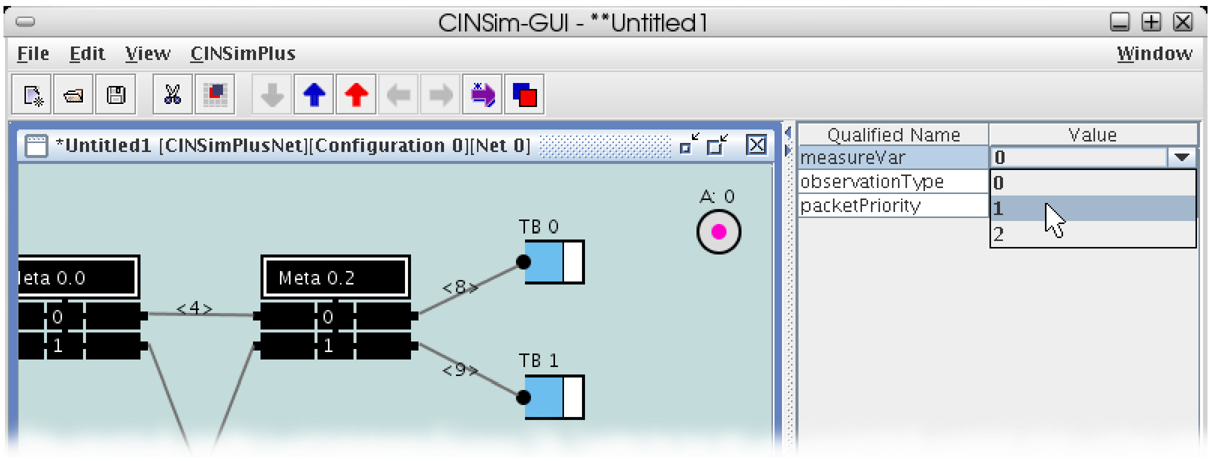


Figure 35. Analyzer selection

The first attribute *measureVar* sets the Akaroa variable to be used. Clicking the value field will bring up a combo box holding all Akaroa variables defined. The number *0* indicates that no variable is selected, therefore no measuring will take place. We want to use variable *1* to analyze the packet delay within the network. Select it from the combo box as shown in Figure 36. The remaining attributes refer to the selected variable and can not be set manually. The label of an analyzer shows the selected Akaroa variable.



Qualified Name	Value
measureVar	0
observationType	0
packetPriority	1
	2

Figure 36. Setting up an analyzer

Now we need to specify the objects to be included in the measurement. For this purpose there are special connectors that bind analyzers to net components, called *observation lines*. Select the related symbol from the insertion tool bar (Figure 37).

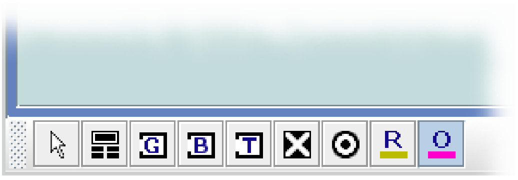


Figure 37. Observation line selection

Every observation line starts at an analyzer object and may end up at packet generators, buffers or targets, depending on the type of measurement. Select the analyzer object and hold down the left mouse button (Figure 38). Move the mouse cursor, followed by a thin line, to one of the target objects and release the mouse button. This will insert a connection inscripted with *Delay* as shown in Figure 39. Proceed by connecting the remaining targets.

Observing all targets will analyze the average delay of all packets passed through the network, while observing a single target would only analyze the packets received by this target.

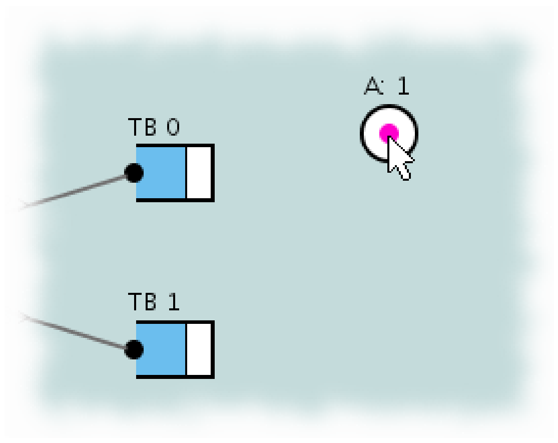


Figure 38. Defining new observation (1)

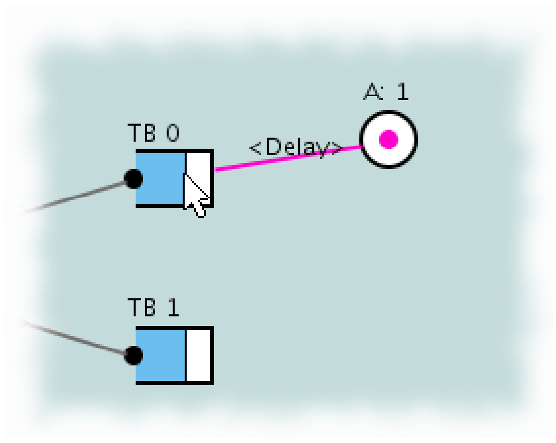


Figure 39. Defining new observation (2)

At last we need to insert an analyzer to observe the throughput of some of the sources. Choose the Akaroa variable *2* to be used and connect the inserted analyzer to the packet generators *1* and *3*. Doing so should lead to a fully set up net configuration similar to Figure 40.

Note

You need to hit the white areas of the packet generators to insert an observation line.

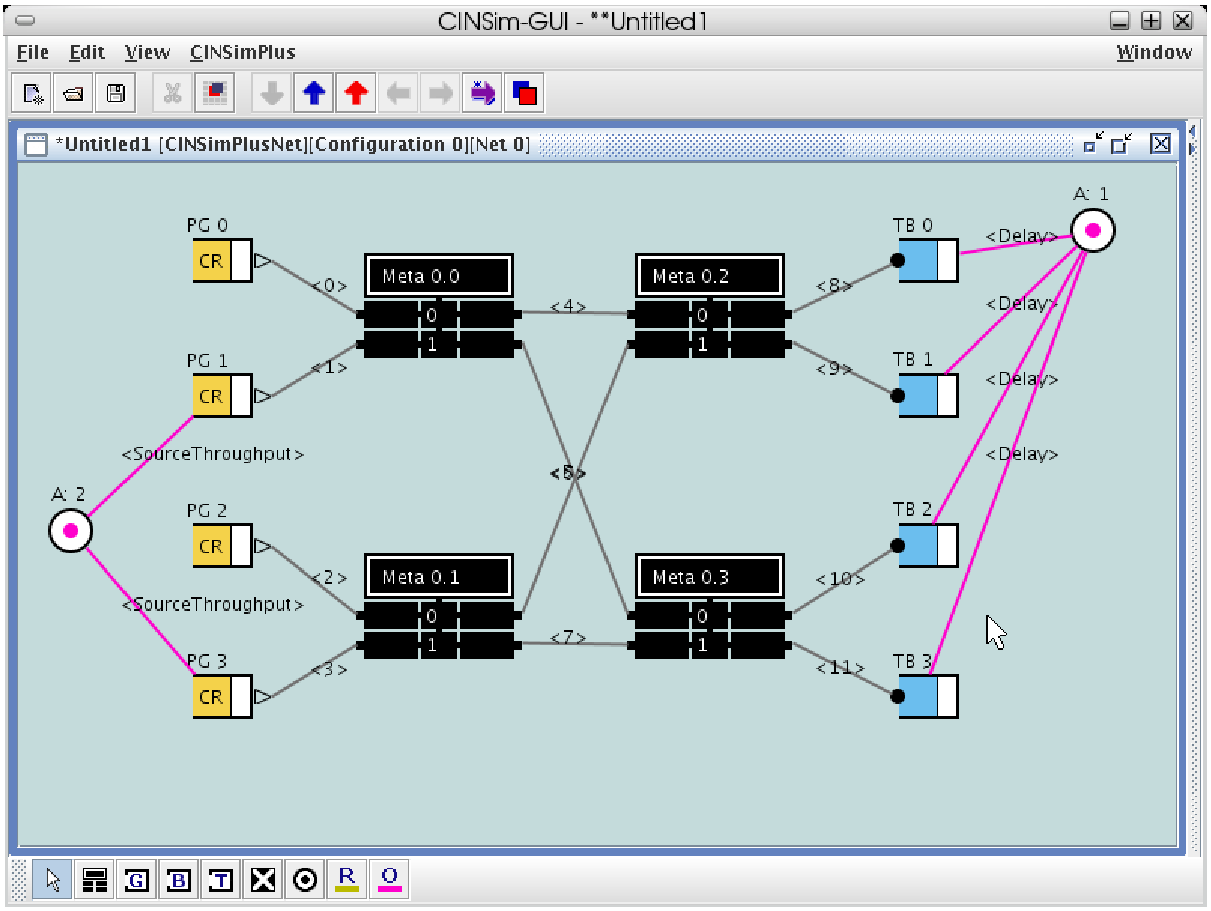


Figure 40. Configuration with observations

4.7. Save and Exit

It is always a good idea to return to the top level when you are finished with setting up the net configurations of the simulation set. Use one of the ascend buttons as shown in Figure 41. Now you can check, whether everything is set up to your liking. At this time, check whether you have already set the reconfiguration attribute "dynamic" as you wished. When you are done, it is time to save the created simulation set.

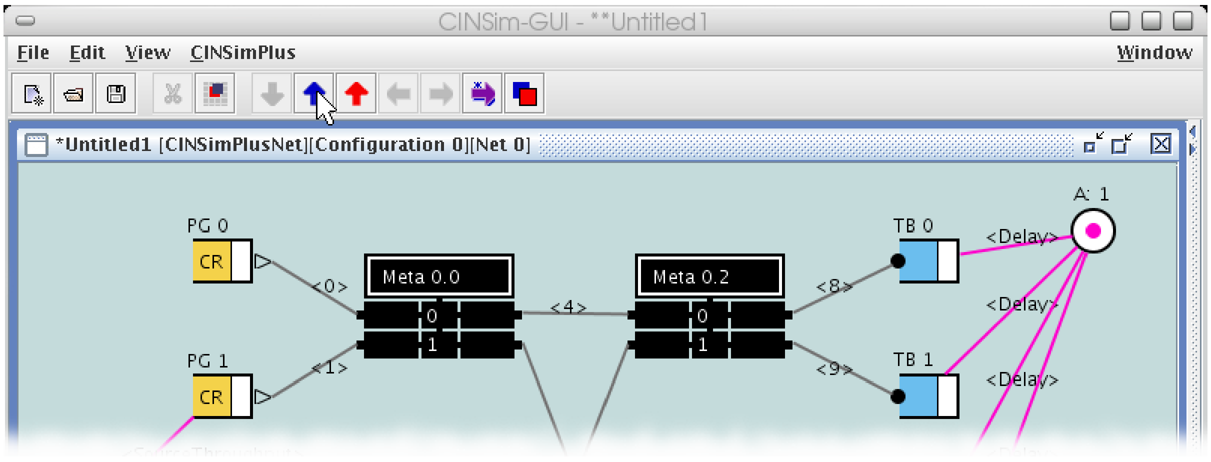


Figure 41. Ascending from net configuration

Select the *File* menu from the menu bar at the top of the editor window and choose the entry *Save as* (Figure 42). This will bring up the file chooser dialog shown in Figure 43. Use the dialog to navigate to a directory your simulation sets are to be stored. Enter a name for the XML file and proceed by hitting *Save*. The file extension *.xml* will be added automatically if omitted.

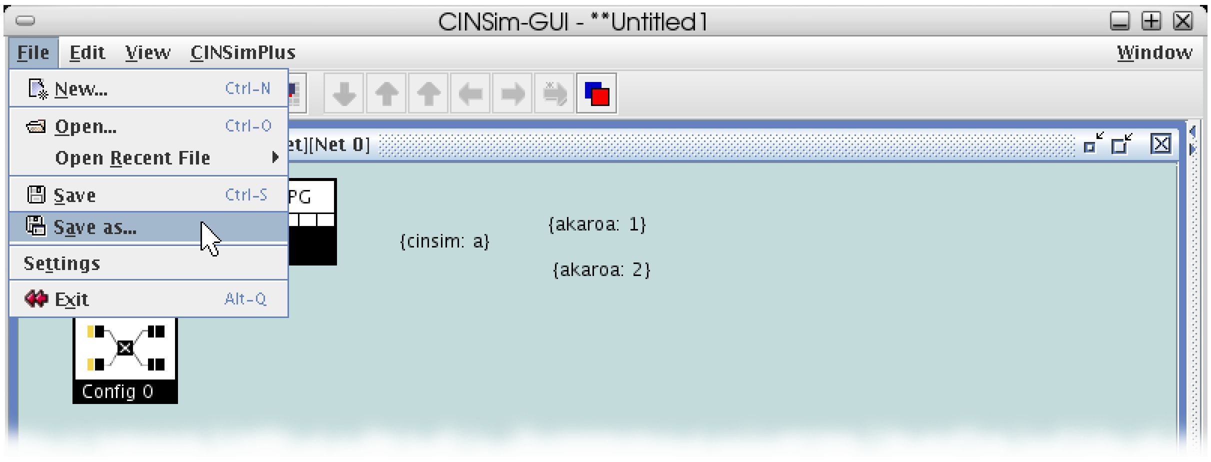
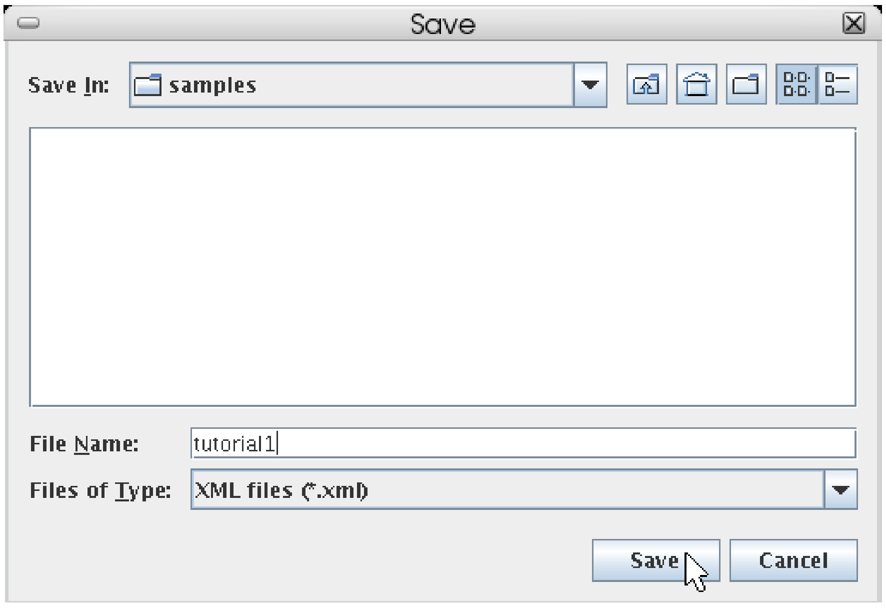


Figure 42. Save as



File Name: tutorial1

Files of Type: XML files (*.xml)

Figure 43. File chooser

The simulation set is now prepared and stored to an XML file. There is nothing left to do. You can use the entry *Exit* from the *File* menu to close the editor. Have fun!

5. Simulation Runs

5.1. Preparations

The generated simulation set *tutorial1.xml* is ready for investigation, but before starting simulation runs using the core applications of CINSim, we first need to set up the environment.

Every simulation run is observed by Akaroa and will be stopped if specified precision and confidence levels are met. We need to tell Akaroa what levels to use, otherwise Akaroa uses its defaults. The common approach for *cinsim* and *csequencer* is to use a configuration file named *Akaroa*. This file must be placed in the directory where the simulation runs are started. A sample configuration file *Akaroa.default* can be found in *PREFIX/ share/cinsim/*, where *PREFIX* specifies the installation directory of the core applications. Copy and rename this file to your simulation directory and edit it according to your wishes. The initial version shows the Akaroa defaults for precision and confidence.

```
MaxSchreubenHeuristicObs = 100000
MaxTransientObs = 100000
Precision = 0.05
Confidence = 0.95
```

The remaining entries usually need no modification. Refer to the Akaroa manual for more information.

Running the *csequencer* involves several tools that come with the Akaroa package. You will basically need to call *akmaster* and *akslave*, but also other applications like *akadd* and *akstat* can be quite useful. In the following we assume that Akaroa is installed properly and the directory holding the binaries is set in your PATH variable. Furthermore, we will need to have gnuplot installed, to create plots from our simulation results.

5.2. Using *cinsim*

The core application *cinsim* is meant to be the simulation client. It parses a given simulation set and starts a single, undistributed simulation run. You should use the client only for testing purposes, for example, to check whether a simulation set is set up correctly. You can not access intermediate results and hitting *ctrl-c* will kill the simulation process without delivering any results.

Before we start the simulation run using *cinsim*, we need to recall what kind of simulation we set up in the simulation set stored to the file *tutorial1.xml*: We set up a steady-state simulation sequence with one CINSim variable, named *a*, to increase the traffic load up to 100 %. As mentioned before, the client *cinsim* can only start single simulation runs and no sequences. For this reason, *cinsim* needs to know how to assign CINSim variables used for simulation sequences. This can be done by using the command line option *-V* followed by a value assignment. To run the simulation with an offered load of 100 % we need to type

```
cinsim tutorial1.xml -V a=100
```

You can get all available command line options by typing

```
cinsim --help
```

Depending on precision and confidence levels and, of course, the complexity of the simulation set, the simulation run might take some time and if the desired levels for at least one observation can not be reached, the simulation will not terminate. For the given simulation set termination should be no problem and calling *cinsim* with the parameters above will produce the following output

```
XML-File parsed, starting simulation...
Param    Estimate       Delta  Conf         Var   Count   Trans
    1     5.33154    0.162297  0.95  0.00470832    1560     260
    2    0.732013   0.0346193  0.95  0.00021423    1818     303
#Results obtained using the Akaroa2(c) automatic parallel simulation manager.
```

The column *Param* refers to the Akaroa variables defined in the simulation set. Variable one is used to observe the flit delay at the targets and variable two to observe the flit throughput at selected sources. *Estimate* gives the mean value and *Delta* the variance for the observation. *Conf* shows the confidence level used as termination criteria. Refer to the Akaroa manual to learn more about the results.

5.3. Using *csequencer*

We strongly recommend to use *csequencer* for your simulations. This application is a wrapper for the simulation client *cinsim* that allows to run simulation sequences and distributed simulations. Furthermore, you can get intermediate results by using the Akaroa application *akstat*. Once the results are to your liking, you can stop a simulation run by hitting *ctrl-c* and the current results will be delivered on your screen.

Before we can call *csequencer* on a simulation set, we need to start an *akmaster* process and an *akslave* client in the background by typing

```
akmaster & akslave &
```

To run the simulation sequence set up in this tutorial we can now type

```
csequencer tutorial1.xml
```

This should produce the output below. The CINSim variable *a* will be evaluated and incremented. The simulation runs ten times, each time with another value for the offered load.

```
Program started...
XML-File parsed, starting simulationsequence...

Repetition: #
Parameter 1 (Delay): Mean = 2.21997 +/- 0.0453146 and 0.0453146
Parameter 2 (SourceThroughput): Mean = 0.180051 +/- 0.00816497 and 0.00816497
Repetition: #
Parameter 1 (Delay): Mean = 2.50735 +/- 0.0697226 and 0.0697226
Parameter 2 (SourceThroughput): Mean = 0.345543 +/- 0.0163168 and 0.0163168
Repetition: #
Parameter 1 (Delay): Mean = 3 +/- 0.131934 and 0.131934
Parameter 2 (SourceThroughput): Mean = 0.454264 +/- 0.0146227 and 0.0146227
Repetition: #
Parameter 1 (Delay): Mean = 3.43636 +/- 0.0919607 and 0.0919607
Parameter 2 (SourceThroughput): Mean = 0.547146 +/- 0.0221494 and 0.0221494
Repetition: #
Parameter 1 (Delay): Mean = 3.74579 +/- 0.161267 and 0.161267
Parameter 2 (SourceThroughput): Mean = 0.599783 +/- 0.0209639 and 0.0209639
Repetition: #
Parameter 1 (Delay): Mean = 4.20252 +/- 0.148466 and 0.148466
Parameter 2 (SourceThroughput): Mean = 0.644014 +/- 0.0235073 and 0.0235073
Repetition: #
Parameter 1 (Delay): Mean = 4.50283 +/- 0.127849 and 0.127849
Parameter 2 (SourceThroughput): Mean = 0.680336 +/- 0.0160676 and 0.0160676
Repetition: #
Parameter 1 (Delay): Mean = 4.58715 +/- 0.14501 and 0.14501
Parameter 2 (SourceThroughput): Mean = 0.702878 +/- 0.0326623 and 0.0326623
Repetition: #
Parameter 1 (Delay): Mean = 4.8672 +/- 0.207348 and 0.207348
Parameter 2 (SourceThroughput): Mean = 0.719444 +/- 0.0242733 and 0.0242733
Repetition: #
Parameter 1 (Delay): Mean = 5.26231 +/- 0.242706 and 0.242706
Parameter 2 (SourceThroughput): Mean = 0.757435 +/- 0.0304655 and 0.0304655
```

The output of the *csequencer* application shows the estimated mean, the upper and lower errors (delta) and the type of observation for every Akaroa variable defined in a simulation set. In case of terminating simulations, values for each observed clock cycle will be displayed.

Tip

You can change the precision and confidence level by using command line options instead of editing the Akaroa configuration file. The option *-P* sets the precision and *-C* the confidence level. The number format follows the configuration file example in Section 5.1. Run *csequencer* with the option *--help* to get all options.

The use of the *csequencer* application allows you to access intermediate results during simulation runs. This might be desirable when running simulations with high precision and confidence levels and therefore long runtimes. On the other hand, a simulation run does not necessarily terminate, so you will need intermediate results to guess when to stop it. To access results of running simulations you have to make use of *akstat* or *akgui*. Both applications come with the Akaroa package. We will shortly discuss the use of *akstat* in the following. For more information refer to the Akaroa manual.

Whenever you run a simulation using the *csequencer* you can open a new terminal and type

```
akstat -G
```

This should produce an output like the one below, showing an example simulation with three parameters.

```
SID PAR❶ REQP CONF❷       MEAN❸   PREC❹    VARIANCE      OBS    TRANS  CHKPTS  CP/MIN LAST CHKPT
  1    1 0.010 0.99           0     nan           0        0        0       0       0
  1    2 0.010 0.99     4.06521 0.00230 1.08896e-05   100230      736     133       0 Thu Sep 15 10:33:15 2005
  1    3 0.010 0.99    0.507638 0.00442 6.27132e-07   208467      750     276       0 Thu Sep 15 10:33:29 2005
```

❶ This column shows the observed Akaroa parameters. The numbers are the same as in the *csequencer* output and refer to the Akaroa variables defined in a simulation set.
❷ The confidence levels used for each parameter of a simulation.
❸ This column holds the current mean values for all observed parameters.
❹ For every observed parameter the precision according to the current mean value will be calculated. The simulation will be stopped if all of these values fall below the specified precision for the simulation. *nan* (not a number) and *inf* (infinite) indicate that there are not enough values to calculate a mean or all of the observed values equal to zero. In both cases, the precision and variance for a parameter can not be determined. As long as any parameter's current precision exceeds the desired precision or is set to *nan* or *inf*, the simulation run will not terminate.

Note

akstat does not provide a real-time view on simulation results. You will have to call *akstat* again, each time you want the latest values.

5.4. Distributed Simulations

The *csequencer* application makes it possible to distribute simulations over several hosts for performance optimization. The previous section already introduced the usage of the *csequencer* to run undistributed simulations or simulation sequences. To run simulations distributed is actually not much more complicated. Again, we need one *akmaster* process but we need to start more than one *akslave* client, each one on a different host. So we start by typing

```
akmaster &
```

This will start the *akmaster* process and the file *.akmaster* will be created in your home directory. This file will be used by each *akslave* client to identify the *akmaster* process. Now we can start a client on the current host by typing

```
akslave &
```

We then need to start a client on every further host to be invoked, too. The hosts must share the same file system, for example via *NFS*, so that all clients can access the *.akmaster* file. If only the home directory is shared, you have to make sure that CINSim is installed and available on every single host. Alternatively, you can install CINSim one time in your home. To start the clients you need to log in on each host, most likely using *ssh*, and call *akslave*. You can speed up this procedure by using the public key authentication feature of *ssh* (SSH protocol version 2). Refer to the man pages for more information. Once set up, we can simply type

```
ssh -f HOSTNAME akslave
```

to start a client. No password authentication is needed and no remote shell will be started. After starting the clients we can call *akstat* to get an overview.

```
kuehmi@iscream:~$ akstat
SLAVE  HOST                              PID  ENGINES
    1  uscream                          4969        0
    2  pontiac                         14856        0
    3  iscream                         14223        0
   SID PARMS  ENGS        RANDOM FLAGS COMMAND
```

The example above shows three clients available for distributed simulation runs. You can only start one client on each host.

A distributed simulation can now be started by specifying the number of hosts to be invoked. For this purpose, the command line option *-H* (or *--host*) of the *csequencer* application must be used. To use three clients to run the simulation set created in this tutorial we need to type

```
csequencer tutorial1.xml -H 3
```

to start over. To check whether everything works fine we can use the *akstat* application again.

```
kuehmi@iscream:~$ akstat
SLAVE  HOST                              PID  ENGINES
    1  uscream                          4969        1 ❶
    2  pontiac                         14856        1
    3  iscream                         14223        1
   SID PARMS  ENGS        RANDOM FLAGS COMMAND
    22     2     3 ❷14281:0000078897
...
```

❶ There is one simulation engine running on host *uscream*.
❷ Overall there are 3 engines running, distributed over three host.

Note

If the specified number of hosts exceeds the number of available hosts with running clients, multiple simulation engines will be started on some hosts.

5.5. Post-Processing Using *gnuplot*

The core applications *cinsim* and *csequencer* of CINSim produce textual output. However, in some cases a graphical representation is desirable, for example, when investigating transient behaviour or running simulation sequences. In the following we will give some exemplary instructions how to visualize simulation results using *gnuplot*. Future versions of CINSim will produce gnuplot-ready data files automatically.

This tutorial discusses a simulation sequence consisting of ten simulation runs, each one with a different traffic load. One might be interested in a graph showing the evolution of the observed network properties, the flit-based delay at the targets and the flit-based source throughput. We need to start by redirecting the *csequencer* output to a file. We are only interested in the values for the observed parameters, so we will ignore additional information.

```
csequencer tutorial1.xml | grep "^Parameter" > tutorial1.data
```

This command will run the simulation and create the data file *sequence* holding the simulation results. Now it is time for *awk*, *sed* and finally *gnuplot*. It is hardly possible to explain in few words how these tools work, so the following script *seqtoplot.sh* is meant to be an example. It takes our data file, creates a plot and additionally two data files, one for each observation.

```
#!/bin/sh
#
# seqtoplot.sh

pwd=$(pwd)

if [ ! -f "$1" ]; then
echo "Cannot open file $1"
exit
fi

pnr=`awk 'max $2 { max = $2 } END { print max }' < $1`;\
let i=$pnr-1;
while [ "$i" != "-1" ]; do
echo "writing $pwd/$(basename $1)_$i";
let line=($i+$pnr-1)%$pnr+1
name=`sed -n "$line p" < $1 |awk '{print $3}'|
sed -e "s#(\(.*\)).*#\1#g"`
awk ' NR%'$pnr'=='$i' \
{ print (NR+('$pnr'-'$i')%'$pnr')/'$pnr' " " $6 } ' < $1\
> $pwd/$(basename $1)-$i ;

# generate an appropriate gnuplot line
newline="\"$1_$i\" with lines lw 2 title \"$name\""
if [ "$plotline" != "" ]; then
plotline=$plotline,$newline;
else
plotline=$newline;
fi
let i=i-1;
done

plotfile=$pwd/$(basename $1).ps

gnuplot << EOF
set title "sequenced simulation"
set terminal postscript color
set output "$plotfile"
plot $plotline
EOF

echo "writing $plotfile"
```

Running the script with our data file *tutorial1.data* creates the plot *tutorial1.data.ps* and two data files *tutorial1.data-0, tutorial1.data-1* holding the results for each observation.

```
sh ./seqtoplot.sh tutorial1.data
```

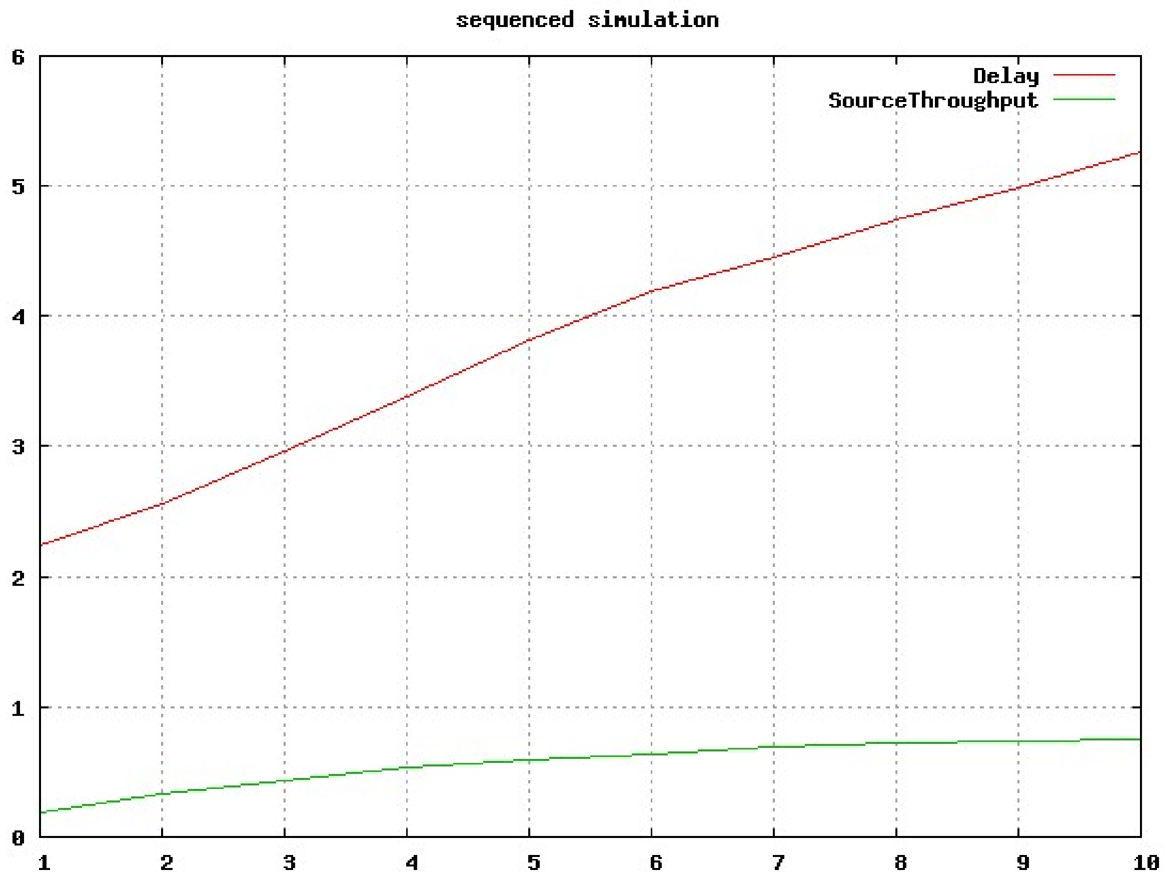


Figure 44. Visualized results using gnuplot

A. Sources of Information

A.1. CINSim

1. The CINSim Homepage (http://www.lfa.uni-wuppertal.de/forschung/simulator-cinsim-engl)
2. Online Tutorials (URL)
3. The CINSim Mailinglist < `cinsim AT uni-wuppertal.de` >

A.2. MPI Implementations

1. The MPICH Homepage (http://www-unix.mcs.anl.gov/mpi/mpich1/)
2. The MPICH2 Homepage (http://www-unix.mcs.anl.gov/mpi/mpich2/)
3. LAM/MPI Parallel Computing (http://www.lam-mpi.org/)

A.3. GNU Scientific Libraries

1. GSL - GNU Scientific Library (http://www.gnu.org/software/gsl/)

A.4. Xerces-C++

1. The Xerces-C Homepage (http://xml.apache.org/xerces-c/index.html)

A.5. Bison++

1. Tutorial by Mario Konrad (http://www.mario-konrad.ch/index.php?page=20024)
2. Tarball download (ftp://ftp.tu-darmstadt.de/pub/programming/languages/C++/tools/flex++bison++/LATEST)

A.6. XSLT Processors

1. The xsltproc tool (http://xmlsoft.org/XSLT/xsltproc2.html)
2. Saxon (http://saxon.sourceforge.net/)
3. Xalan (http://xalan.apache.org/)

Bibliography

[1] H. Akimaru and K. Kawashima. *Teletraffic - Theory and Applications*. 2nd Edition. Springer-Verlag London. 1999.

[2] J.E. Flood. *Telecommunications Switching, Traffic and Networks*. 1998. Prentice-Hall.

[3] Glen Kramer. *On generating self-similar traffic using pseudo-Pareto distribution*. Technical Brief, Network Research Lab, Department of Computer Science, University of California. 2000.

[4] Matthias Kühm. *Stochastische Auswertung und Beschleunigung von Netzwerksimulationen (in German)*. Master's thesis, Technische Universität Berlin. 2006.

[5] A.M. Law and W.D. Kelton. *Simulation Modeling and Analysis*. 3rd Edition. McGraw-Hill. 2000.

[6] N. McKeown, A. Mekkittikul, V. Anantharam, and J Walrand. *Achieving 100 % Throughput in an Input-Queued Switch*. August 1999. IEEE Trans. Commun.. Vol. 47. No. 8. pp. 1260-1267.

[7] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*. University of Tennessee. Knoxville, Tennessee. June 1995.

[8] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*. University of Tennessee. Knoxville, Tennessee. July 1997.

[9] Krysztof Pawlikowski, Victor W.C. Yau, and Don McNickle. *Distributed Stochastic Discrete-Event Simulation in Parallel Time Streams*. Proceedings of the 1994 Winter Simulation Conference. December 1994. pp. 723-730.

[10] Hans-Christian Rahloff. *Simulative Leistungsbewertung von Scheduling-Verfahren in unregelmäßigen mehrstufigen Verbindungsnetzen. Diplomarbeit (in German)*. Master's thesis, Technische Universität Berlin. 2004.

[11] Dietmar Tutsch and Marcus Brenner . *MINSimulate - A Multistage Interconnection Network Simulator* . 17th European Simulation Multiconference: Foundations for Successful Modelling & Simulation (ESM'03) . Nottingham, UK . 2003 . pp. 211-216 .

[12] Dietmar Tutsch, Daniel Lüdtke, Arvid Walter, and Matthias Kühm. *CINSim - A Component-Based Interconnection Network Simulator for Modeling Dynamic Reconfiguration*. Proceedings of the 12th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA 2005). Riga, Latvia. June 2005. pp. 132-137.

[13] Dietmar Tutsch, Daniel Lüdtke, and Matthias Kühm. *Investigating Dynamic Reconfiguration of Network Architectures with CINSim* . Proceedings of the 13th Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems 2006 (MMB 2006). Nürnberg, Germany. March 2006. pp. 445-448.

[14] Dietmar Tutsch. *Performance Analysis of Network Architectures*. 1st Edition. Springer Verlag Berlin. 2006.

[15] Li Zhou. *Simulation der Rekonfigurierung mehrstufiger Verbindungsnetze (in German)*. Master's thesis, Technische Universität Berlin. 200.

[16] Christian Zimmermann. *Simulative Leistungsbewertung und Entwicklung rekonfigurationsfester Routingverfahren für mehrstufige Verbindungsnetzwerke (in German)*. Master's thesis, Eidgenössische Technische Hochschule Zürich. 2006.