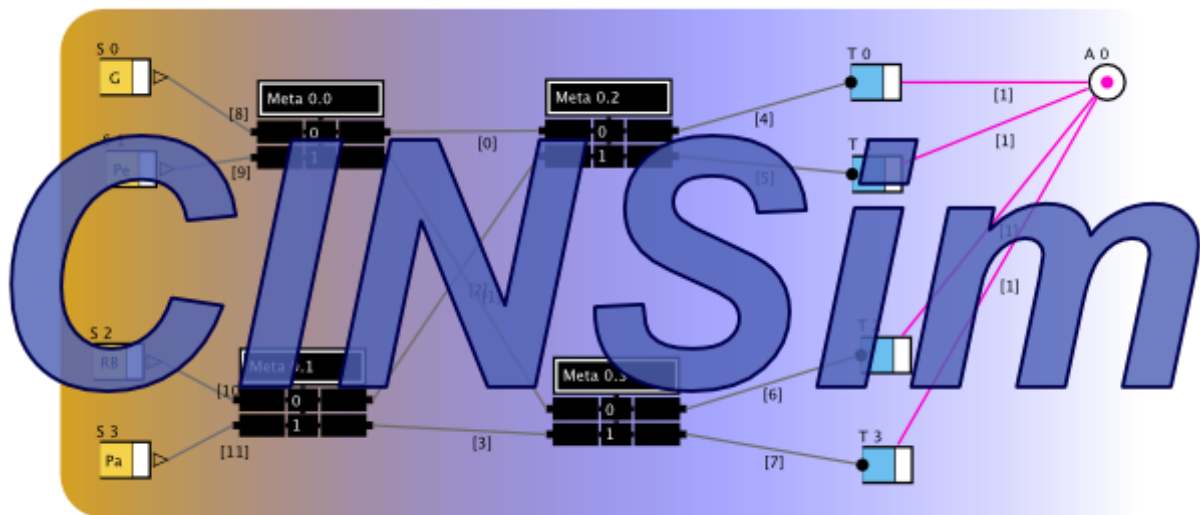


CINSim Handbook

Installation and User's Guide



Arvid Walter
Matthias Kühm
Dietmar Tutsch
Daniel Lüdtkke
Christian Zimmermann
<http://pdv.cs.tu-berlin.de/cinsim/>

CINSim Handbook: Installation and User's Guide

by Arvid Walter, Matthias Kühm, Dietmar Tutsch, Daniel Lüdtke, and Christian Zimmermann
Copyright © 2004-2007 Technische Universität Berlin, Real-Time Systems and Robotics

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, crucial steps for designing, simulating and analyzing networks with CINSim will be discussed.

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

I. Introduction	10
1. About this Document	12
1.1. What Is the CINSim Project?	12
1.2. Who Should Be Reading this Guide	14
2. Basics	15
2.1. Component-Based Interconnection Networks	15
2.2. Stochastic Simulations	16
2.3. Reconfiguration of Interconnection Networks	17
II. Architecture and Tools	18
3. Introduction to the CINSim Tool Chain	20
3.1. Simulation Core	20
3.1.1. Distributed Simulation Runs	20
3.1.2. Simulation Sequences	20
3.2. Graphical User Interface	21
3.3. Gnuplot	22
4. Installation	23
4.1. Getting the Sources and Libraries	23
4.1.1. CINSim	23
4.1.2. MPI	23
4.1.3. GNU Scientific Library	23
4.1.4. Xerces-C++	24
4.1.5. Bison++	24
4.2. Upgrading from CINSim 0.4 to 0.6	25
4.3. How to Configure, Compile and Install	26
4.3.1. CINSim Tarball Archives	26
4.4. Converting your network models from 0.4 to 0.6	29
5. Simulation Setup with <i>cinsim-gui</i>	30
5.1. Editor Framework	30
5.1.1. General Information	30
5.1.2. Getting Started	30
5.1.3. Editing Net Class Files	32
5.2. Net Class Extension CINSimPlus	39
5.2.1. Hierarchical Structure	39
5.2.2. Local/Global Settings Menu	42
5.2.3. Network Components	47
5.2.4. Using parameters and formulas	50
5.2.5. Defining Measurements	52
5.2.6. Deriving Net Descriptions	54
5.2.7. Reconfiguration Steps	57
5.2.8. Specialities of Dynamic Reconfiguration	60
6. Simulation Runs with <i>cinsim</i>	61
6.1. The Basics	61
6.2. Using <i>cinsim</i>	62
6.2.1. Command Line Options	62
6.2.2. Simulation Sequences	64
6.2.3. Distributed Simulation	64
6.2.4. Intermediate Results	65
6.2.5. Gnuplot Support	65
6.3. Configuration	67
III. Specifications	68
7. Simulation Components	70
7.1. The Coroutine	70
7.1.1. Local Backpressure	70
7.1.2. Global Backpressure	71
7.1.3. Alternatives	71
7.2. Routing Strategies	72
7.2.1. Bitmask Routing	72

7.2.2. Shortest-Path Routing	72
7.2.3. XY Routing	73
7.2.4. West-First Routing	73
7.3. Switching Techniques	74
7.3.1. Store-And-Forward Switching	74
7.3.2. Virtual Cut-Through Switching	75
7.3.3. Partial Cut-Through Switching	75
7.3.4. Wormhole Switching	75
7.4. Steady-State vs. Terminating Simulations	76
8. Network Components	77
8.1. Source Buffer Component	77
8.1.1. Constant-Size Packets	77
8.1.2. Distribution in Time	77
8.1.3. Cumulative Distribution Functions	79
8.2. Buffer Component	82
8.2.1. Queue Types	82
8.3. Target Buffer Component	83
8.4. Router Component	84
8.4.1. Arbiter	84
8.4.2. Arbiter Controller	88
8.5. Analyser Component	89
8.5.1. Observation Types	89
8.5.2. Measurement Statistics Types	90
9. Reconfiguration	91
A. Distribution Functions	93
A.1. Syntax Description	93
B. Sources of Information	95
B.1. CINSim	95
B.2. MPI Implementations	95
B.3. GNU Scientific Libraries	95
B.4. Xerces-C++	95
B.5. Bison++	95
B.6. XSLT Processors	95
Glossary	97
Bibliography	104
Index	105

List of Figures

1.1. Simulation setup	12
2.1. 3D cube	15
2.2. 2D mesh	15
2.3. 16x16 min	15
5.1. Initial editor window	31
5.2. Editor settings	31
5.3. Choosing a net class	32
5.4. Choosing a file to open	32
5.5. Editor window with new file	33
5.6. Editor framework - File menu	33
5.7. Editor framework - Edit menu	33
5.8. Editor framework - View menu	34
5.9. Editor framework - Window menu	34
5.10. Inserting a new node	35
5.11. Inserting a connection (1)	36
5.12. Inserting a connection (2)	36
5.13. Default right-click menu	37
5.14. Using handles to redirect connectors	37
5.15. Attribute with enumeration type	38
5.16. Attribute with built-in or pattern based type	38
5.17. Descending into network configuration	40
5.18. Changing the dimension of a meta element	41
5.19. CINSimMetaNet - I/O panels	41
5.20. Changing the dimension of a meta element	42
5.21. Simulation setup	43
5.22. Source buffer element	47
5.23. Buffer element	48
5.24. Target element	48
5.25. Router element	48
5.26. A small network example	48
5.27. Meta element	49
5.28. I/O Panels	49
5.29. Connecting mesh nodes	49
5.30. Valid layout of a mesh node	50
5.31. Setting up a CINSim parameter	50
5.32. Using a CINSim parameter	51
5.33. Setting up a formula (1)	51
5.34. Setting up a formula (2)	51
5.35. Using a formula	52
5.36. Measurement variable setup (1)	53
5.37. Measurement variable setup (2)	53
5.38. Analyser setup	54
5.39. Observation	54
5.40. New nested net element from draft (1)	55
5.41. New nested net element from draft (2)	55
5.42. New document from draft (1)	56
5.43. New document from draft (2)	56
5.44. Inserting net element from external document	56
5.45. Reconfigured network	57
5.46. Single reconfiguration step	58
5.47. Setting for static reconfiguration	58
5.48. Setting for dynamic reconfiguration	58
5.49. Reconfiguration steps with loop	59
5.50. Switching to the next network configuration	59
5.51. Mapping to a component of previous network configuration	60
7.1. Global backpressure routing oriented on targets	71
7.2. Global backpressure routing, deadlock case, oriented on targets and buffers	71
8.1. ON/OFF Model	77

8.2. Bursty traffic	79
8.3. Non-normalized sample multicast distribution function $f(n)$	80
8.4. Routing matrix with conflict	84
8.5. Routing matrix after scheduling	84
8.6. GRR (1)	85
8.7. GRR (2)	86
8.8. GRR (3)	86
8.9. GRR (4)	87
9.1. Simulation example	92

List of Tables

4.1. Configure script options	27
5.1. Editor framework - Tool bar	34
5.2. Net class CINSimPlus - Tool bar	39
5.3. CINSimPlusNet - Insertion tool bar	40
5.4. CINSimNet/CINSimMetaNet - Insertion tool bar	41
5.5. MeshNet - Insertion tool bar	42
5.6. Valid attribute values for simulation setup	43
5.7. Valid attribute values for buffer setup	44
5.8. Valid attribute values for router setup	44
5.9. Valid attribute values for source buffer setup	45
5.10. Valid attribute values for geometric distribution setup	46
5.11. Valid attribute values for periodic distribution setup	46
5.12. Valid attribute values for random-burst distribution setup	46
5.13. Valid attribute values for pareto distribution setup	47
5.14. Valid attribute values for reconfiguration setup	58
7.1. Value descriptions for attribute <i>forwardingType</i>	70
7.2. Value descriptions for attribute <i>routingType</i>	72
7.3. Value descriptions for attribute <i>switchingType</i>	74
7.4. Value descriptions for attribute <i>simulationType</i>	76
8.1. Geometric traffic attributes	78
8.2. Periodic traffic attributes	78
8.3. Pareto traffic attributes	78
8.4. Bursty traffic attributes	79
A.1. Functions	93
A.2. Standard operators	94
A.3. Constants	94
A.4. Variables	94

List of Examples

6.1. Customized configuration file	67
--	----

Part I. Introduction

Table of Contents

1. About this Document	12
1.1. What Is the CINSim Project?	12
1.2. Who Should Be Reading this Guide	14
2. Basics	15
2.1. Component-Based Interconnection Networks	15
2.2. Stochastic Simulations	16
2.3. Reconfiguration of Interconnection Networks	17

Chapter 1. About this Document

1.1. What Is the CINSim Project?

The CINSim project is located at the [Technische Universität Berlin](http://www.tu-berlin.de) (<http://www.tu-berlin.de>), Real-Time Systems and Robotics Group. The project is targeted on the development of the simulator *CINSim* - Component-based Interconnection Network Simulator - 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 change 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 1.1 to easily describe interconnection networks. The editor is written in JAVA and therefore platform independent.

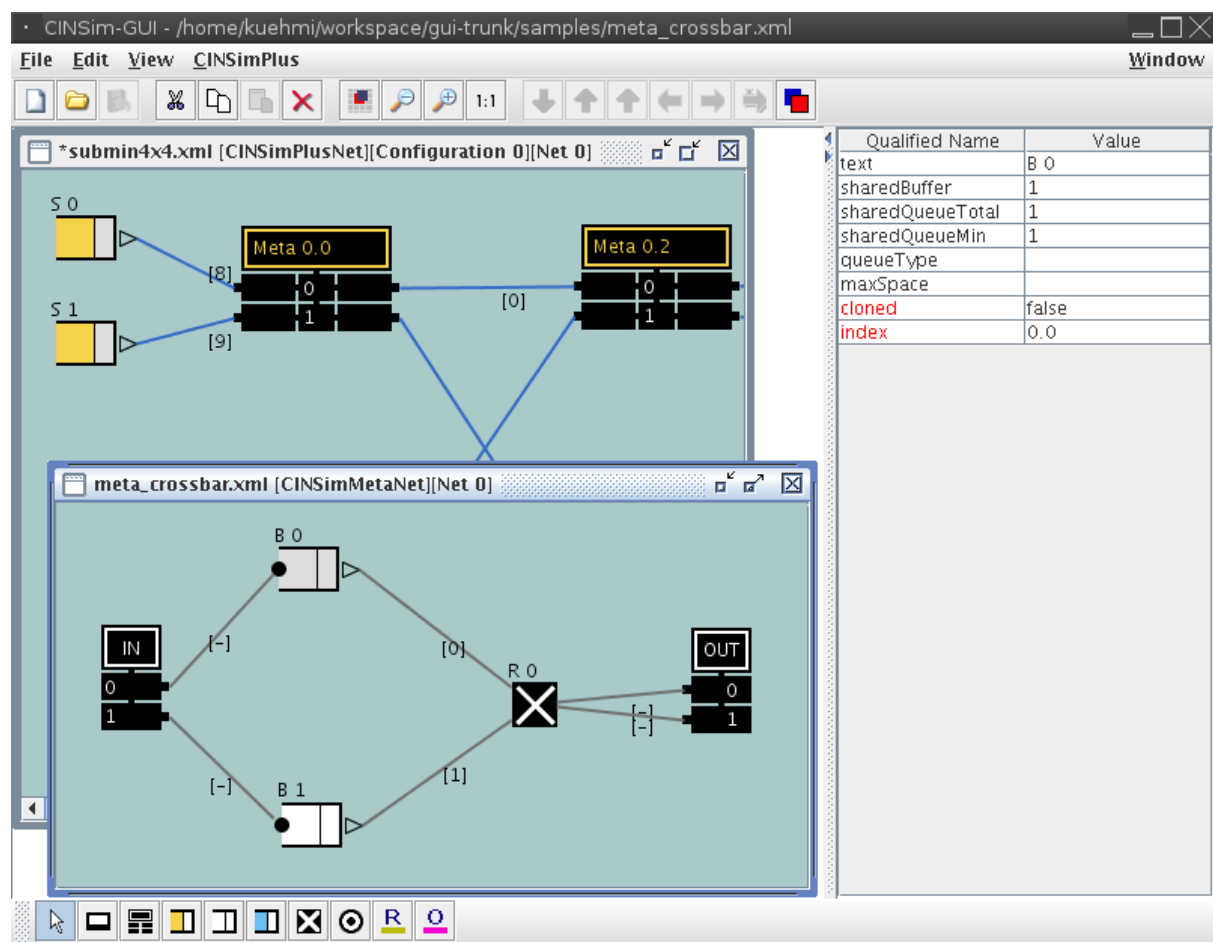


Figure 1.1. Simulation setup

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 prop-

erties 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 B.1](#)). A good overview to already achieved goals and future work can be found in [\[12\]](#) and [\[13\]](#).

1.2. Who Should Be Reading this Guide

This document is targeted to people who want to use the simulator CINSim for performance analysis of interconnection networks where analytical methods fail. That is usually the case if there are too many variations of possible network states for proper calculations. This document provides detailed descriptions to most aspects crucial for successfully using CINSim.

This part of the document continues with some background information to the idea of simulating interconnection networks for performance analysis. By no means, these information are all-embracing, especially if one is completely unexperienced with this subject. However, it might help to classify the simulator CINSim within the wide range of available simulation tools and, hopefully, it motivates the reader to deal with this subject. A selection of further reading can be found in the [Bibliography](#).

[Part II](#) introduces the CINSim tool chain and gives detailed information on how to successfully configure and install all related packages. Furthermore, the basic use of the graphical user interface *cinsim-gui* and the command-line simulator core *cinsim* will be discussed. If you are already experienced with these tools or you are looking for a more exemplary introduction, you might also have a look at the *quickguide* or the available *online tutorials*. See [Appendix B](#) for more information.

Finally, [Part III](#) provides detailed information on how the realised concepts work. This is intended for those people, who have already run some simulations using CINSim and now want to know more about the meanings of the various settings that can be made. It is usually a good idea to read this part before intensively using CINSim, because it may help to understand the calculated results.

Chapter 2. Basics

2.1. Component-Based Interconnection Networks

In telecommunication, interconnection is the physical linking of two or more carriers. Communication takes place across a more or less complex network. The main purpose is to connect multiple computers or processors using adaptive, cheap and efficient communication structures. It also plays a central role in determining the performance of multicomputer, multiprocessor, and multicore systems.

Depending on the application, a variety of topologies have been developed and investigated, such as meshes (Figure 2.2), tori, cubes (Figure 2.1), hypercubes, butterfly networks (Figure 2.3), Clos networks, trees, rings and many more [14]. At the same time of that investigation, buses, as an alternative structure, long the mainstay of system interconnect, were and are unable to keep up with increasing performance requirements. Interconnection networks offer an attractive solution and are becoming pervasive in digital systems. A well-designed interconnection network makes efficient use of scarce communication resources providing low-latency communication between carriers with a minimum of cost and energy. Years ago only used in high-end supercomputers and telecom switches, interconnection networks are now found in systems of all sizes and all types. They are used in large supercomputers, but in small embedded systems-on-a-chip (SoC) as well, also in applications including inter-processor communication, processor-memory interconnects to name only some among innumerable ones. One can differ between two types of networks - such ones routing packets and such ones routing entire messages. Indeed, as system complexity and integration continues to increase, it is more efficient to route packets. Using a packet routing interconnection network allows scarce bandwidth to be shared so it can be used efficiently with a high duty factor. Another advantage is that interconnection networks also enforce regular, structured use of communication resources, making systems easier to design, use and, of course, easier to analyse.

The basic principles of interconnection networks are relatively simple and it is easy to design an interconnection network that meets all of the requirements for a given application. However, there are disadvantages such as deadlocks or bottlenecks. Fortunately, these are usually design shortcomings.

A component-based network is divided into components such as routers, buffers etc. which are always connected via wires or any similar units. A component might also be a small network itself again.

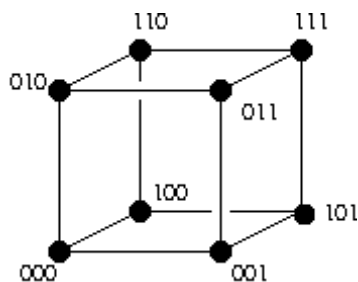


Figure 2.1. 3D cube

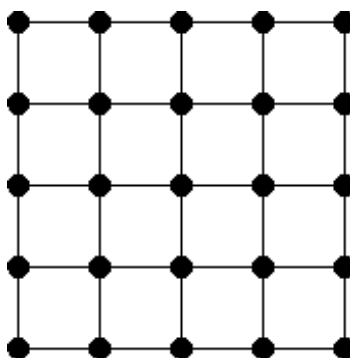


Figure 2.2. 2D mesh

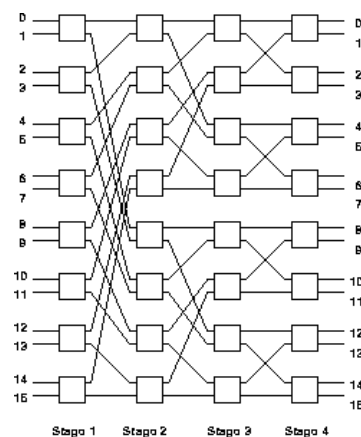


Figure 2.3. 16x16 min

2.2. Stochastic Simulations

The mathematical analysis of complex systems can become extremely difficult, because of the fact that most standard modelling techniques analyse each component individually and do not necessarily consider the relationships between the components within the system. It is also hardly possible to cover all states of a complex system for proper calculations. Simulation is an approach which can be used to model complex and large stochastic systems for performance analysis and forecasting purposes and is usually the most common quantitative modelling technique. The choice of simulation as the way of modelling is usually driven by the advantage that it is less restrictive than many other modelling techniques, because other methodologies often impose material, mathematical or performance restrictions on investigated processes, which sometimes can restrict the result of the analysis. Roughly speaking, a simulation is a much more straightforward approach than mathematical analysis because it usually imitates the investigated process rather than depending on assumptions and exact information on the whole system. However, the mapping from real-world applications to a simulation set may cause inaccuracies that must be taken care of and the time expenditure for simulation runs rises strongly with the complexity of the investigated issue.

Following [5] [1] [2], one can differ between two types of simulation used to model telecommunications networks, *discrete* and *continuous* simulations. Discrete simulations, also known as *discrete event simulations*, are used for *event-based dynamic stochastic systems* [1]. That means, the system contains a discrete number of states, and is modelled using a set of variables. An event is represented by the change of one or more of these variables and is also reflected in a change of the system's state. As the system is dynamic, it is constantly changing, and there is a factor of randomness in the system. Representing *discrete simulations* is performed by using state equations that contain all the variables of the system. Continuous simulations on the other hand also contain state variables. These, however, change continuously in time. For this purpose, continuous simulations are usually modelled using differential equations that track the states of the system according to the elapsed time.

Simulations frequently make use of random number generators to create input data which approximates real-world stochastic events. Generated artificial numbers are usually not random in the strictest sense, as they are created using a set of equations. Such numbers are known as *pseudo-random numbers*. When making use of pseudo-random numbers the user has to assure the true randomness of the numbers. Various algorithms for random number generators exist. Their quality is determined by their cycle size, which describes the quantity of numbers that are generated before the same series of numbers starts again.

One last remark should always be kept in mind. Simulation results are only as good as the model and they only estimate the real world. Therefore, received results should always be treated with respect to these restrictions.

2.3. Reconfiguration of Interconnection Networks

Static network structures are not as flexible as any distribution of traffic would require. Tremendous work has been done to invent new technologies that are capable of a fast reconfiguration according to the changing requests to a network system. A reconfiguration can be done either by changing the behaviour of a circuit, which forms the network, or by changing the various connections linking network components, which can be a function of a circuit, too.

The development of reconfiguration methods has begun to soften the boundaries between hardware and software. The logical design of a computer's hardware is made by hardware engineers using HDLs (**H**ardware **D**escription **L**anguages) to configure PLDs (**P**rogrammable **L**ogic **D**evices) or the more complex FPGAs (**F**ield **P**rogrammable **G**ate **A**rrays), which they then assemble with other integrated circuits, such as microprocessors, to build a computer. Software engineers, are used to work with programming languages such as C and C++. The latter specify instructions to be executed on fixed hardware.

In a reconfigurable system, some parts of the whole system are defined using a system-level description language. This combines both, the features of the software and the hardware languages, allowing a software engineer to write code that could run on a microprocessor or could be converted into a logic description on a FPGA.

Simulations of a reconfigurable system use traffic predictions to design a certain network structure. This can be very efficient if hot-spots within the network can be identified. As a main feature, CINSim is capable of reconfiguring the whole network, changing the circuit and the connections between the components. It is able to deal with "old" packets from the network sent before reconfiguration and redirects it to the new targets. Potentially occurring deadlocks are analyzed and solved by an observation algorithm. Another feature could be the search for the optimal network design depending on the a traffic profile using some kind of evolutionary algorithm. This task could be a feature of an upcoming version of CINSim.

Part II. Architecture and Tools

Table of Contents

3. Introduction to the CINSim Tool Chain	20
3.1. Simulation Core	20
3.1.1. Distributed Simulation Runs	20
3.1.2. Simulation Sequences	20
3.2. Graphical User Interface	21
3.3. Gnuplot	22
4. Installation	23
4.1. Getting the Sources and Libraries	23
4.1.1. CINSim	23
4.1.2. MPI	23
4.1.3. GNU Scientific Library	23
4.1.4. Xerces-C++	24
4.1.5. Bison++	24
4.2. Upgrading from CINSim 0.4 to 0.6	25
4.3. How to Configure, Compile and Install	26
4.3.1. CINSim Tarball Archives	26
4.4. Converting your network models from 0.4 to 0.6	29
5. Simulation Setup with <i>cinsim-gui</i>	30
5.1. Editor Framework	30
5.1.1. General Information	30
5.1.2. Getting Started	30
5.1.3. Editing Net Class Files	32
5.2. Net Class Extension CINSimPlus	39
5.2.1. Hierarchical Structure	39
5.2.2. Local/Global Settings Menu	42
5.2.3. Network Components	47
5.2.4. Using parameters and formulas	50
5.2.5. Defining Measurements	52
5.2.6. Deriving Net Descriptions	54
5.2.7. Reconfiguration Steps	57
5.2.8. Specialities of Dynamic Reconfiguration	60
6. Simulation Runs with <i>cinsim</i>	61
6.1. The Basics	61
6.2. Using <i>cinsim</i>	62
6.2.1. Command Line Options	62
6.2.2. Simulation Sequences	64
6.2.3. Distributed Simulation	64
6.2.4. Intermediate Results	65
6.2.5. Gnuplot Support	65
6.3. Configuration	67

Chapter 3. Introduction to the CINSim Tool Chain

3.1. Simulation Core

The simulation of interconnection networks for performance analysis using the simulator CINSim usually invokes a single binary that comes with the installation package and is called *cinsim*. This simulation engine is started by typing *cinsim* followed by some options and the XML description of the simulation set. If sufficient confidence and accuracy levels are reached, long time simulation runs can be terminated with the shortcut CTRL-C and the current results can be received.

In addition, *cinsim* also supports two important features, to be discussed in the following. More information on how to use *cinsim* can be found in [Chapter 6](#).

3.1.1. Distributed Simulation Runs

To speed up simulations, CINSim is able to run simulations in parallel. Therefore, CINSim incorporates the *Message-Passing Interface (MPI)* [7][8] to handle distribution and communication of the parallel tasks. MPI was designed for large parallel computers, like cluster computers, but some MPI-Implementations (see [Section B.2](#)) support SMP and/or multicore computers.

With MPI, it is possible to start an arbitrary number of tasks. Every task simulates the model with its own random number sequence. All intermediate measurements are collected in one task and the end result is estimated. The simulation stops automatically if the precision and confidence requirements are fulfilled. This scheme is called *Multiple Replications in Parallel (MRIP)* [9].

3.1.2. Simulation Sequences

The simulator CINSim can run a simulation several times, describing a sequence of simulations. CINSim will generate a nearly infinite series of random numbers and therefore each simulation can be executed using different random numbers. The simulator obtains the results at the end of each simulation run and returns the estimates with the highest number of data points for each measurement.

In contrast to simply repeating a simulation run several times, the CINSim can also be used to vary many network settings defined by numerical values, for example, the size of the buffers or the input load of the sources. In this case, the estimates for every single simulation setup, determined by the values of the varied settings, will be returned. Each setup can also be simulated several times.

3.2. Graphical User Interface

The graphical user interface *cinsim-gui* of the CINSim project is not designed to start and observe simulation runs but to define what should be simulated. A simulation set is described in XML based on a XML schema and can either be a single interconnection network or a sequence of networks, defining reconfiguration steps. *cinsim-gui* is a schema-driven XML editor designed with focus on interconnection networks to be simulated using the CINSim core. Both, core and graphical user interface, use the same XML schema file for validation, so it is recommended to use *cinsim-gui* for creating simulation setups.

3.3. Gnuplot

Gnuplot is a command-driven interactive function plotting program using the X-library. It plots a large number of functions, built up of C operators, C library functions, and some things C doesn't have like **, sgn(), etc. and also supports plotting data files, to compare actual data to theoretical curves. The *cinsim* simulator can produce a plot directly after finishing a simulation or can create a data file to be processed with Gnuplot afterward. [Section 6.2.5](#) gives detailed information on how to visualize simulation results obtained by the csequencer.

Chapter 4. Installation

4.1. Getting the Sources and Libraries

4.1.1. CINSim

You can download CINSim from the CINSim homepage ([Section B.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 4.3.1](#).



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++`.

4.1.2. 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 implementation 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 B.2](#) for more information and download. Independent from the implementation that you actually use, you will need the library, the development (header) and runtime files (compiler, execution program). To use MPICH on multicore computers use the MPICH Device: `ch_shmem`.

4.1.3. GNU Scientific Library

The statistical evaluation of simulation requires sophisticated mathematical procedures that are provided by the GNU Scientific Library. A packaged version should be available for any Linux distribution, check for `gsl`. For more information see [Section B.3](#).



Release Information

CINSim was tested with several versions ≥ 1.7 of the library. You will need the library and the development files.

4.1.4. 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 B.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.

4.1.5. 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 B.5](#)) or search for Debian or RPM packages.

4.2. Upgrading from CINSim 0.4 to 0.6

Before you install the new version of CINSim, please make sure, that have you removed the old installation files. Usually, CINSim was installed at */usr/local* on your machine. In this case, you can invoke:

```
rm -r /usr/local/cinsim-0.4 /usr/local/bin/cinsim-gui-0.4 \  
      /usr/local/bin/cinsim /usr/local/bin/csequencer \  
      /usr/local/bin/cinsim-gui
```

Maybe, you need root-privileges for doing this. Please remove old configuration files, too. Candidates are:

- `~/.cinsim/config.xml`
- `INSTALLPREFIX/share/cinsim/config.xml`
- `~/.cinsim-gui.conf`

CINSim 0.6 doesn't depend on Akaroa anymore. You can also remove your Akaroa installation, if you like. Akaroa does not influence the new version of CINSim.

After installation of the complete CINSim 0.6 tool-chain refer [Section 4.4](#) to convert your old network models.

4.3. 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 4.1](#) to a specific directory and all the libraries needed are properly installed.

4.3.1. 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 4.1](#) 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
<i>--prefix=PATH</i>	This parameter sets the installation path. Within the specified directory the sub directories <i>bin</i> , <i>doc</i> , <i>include</i> , <i>lib</i> and <i>share</i> 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 <i>PATH</i> is <i>/usr/local</i> .
<i>--with-gui=GUIPREFIX</i>	Use this parameter with the <i>cinsim-gui</i> installation path to create an appropriate configuration file for <i>cinsim</i> .
<i>--with-xerces-prefix=PATH</i>	This option sets the <i>Xerces-C++</i> path. Configure will create, compile and run a small sample program to check if <i>Xerces-C++</i> is running fine. You need to use this option only if the header-files and libraries of <i>Xerces-C++</i> are unknown to your compiler. In case of this the macro assumes that the missing headers and libraries are to be found in the <i>includes</i> and <i>lib</i> directories within the specified <i>Xerces-C++</i> folder.

option	description
<code>--with-gsl-prefix=PATH</code>	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.
<code>--enable-mrip</code>	This enables 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.
<code>--enable-meshes</code>	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.
<code>--enable-asserts</code>	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 4.1. 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 script 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 4.1](#).

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.

4.4. Converting your network models from 0.4 to 0.6

CINSim 0.6 has many new features implemented and old features were enhanced. Hence, the XML schema of the network description files have changed. You have to convert your old cinsim files to the new format. CINSim-gui provides a xsl-conversion script to do this for you. You need an xslt-processor for this step (see [Section B.6](#)). We recommend *xsltproc* which is often included in current Linux distributions. To convert your 0.4-file invoke:

```
xsltproc GUIPREFIX/04to06.xsl your-0.4-file.xml > new-0.6-file.xml
```

Now, open the new file with *cinsim-gui* and check your network model and save it. Most parameters were converted automatically. *cinsimvars* can not be converted automatically because they are typeless.



Note

One major change between version 0.4 to 0.6 is the new dimension of probabilities: in version 0.4 they were given as values between 0-100. In cinsim 0.6 all probabilities are declared by a floating point number between 0.0-1.0.

Chapter 5. Simulation Setup with *cinsim-gui*

5.1. Editor Framework

The graphical user interface of the CINSim project is used to set up the network models to be simulated. It extends a generic editor for net graphs, developed as part of the [TimeNET](http://pdv.cs.tu-berlin.de/~timenet) (<http://pdv.cs.tu-berlin.de/~timenet>) project, with an own net class *CINSimPlus* for interconnection networks. Both, framework and extension, are written in *Java* and therefore are platform independent.

5.1.1. General Information

The simulation sets processed by the CINSim simulation core are to be specified in XML based on an XML schema. The editor framework provides basic functionalities for schema-driven editing of XML documents and the extension CINSimPlus is build around the same XML schema used by the core for parsing. To create and edit XML documents related to net graphs, the framework defines basic net components and their behaviour:

- Net - Holds net components like nodes and connectors.
- Node - Can be moved and connected to other nodes.
- Connector - Directed connection between nodes. May have decorations for head and tail.
- Text - Textual information. No connections are allowed.
- Handles - Used to redirect connections.

These components are specified by a basic XML schema, defining *types*, and their behaviour by Java classes, defining *views*. An extension, called *net class*, has to extend the schema and the related Java class for every component to be available. A buffer component, for example, needed for interconnection networks, extends the schema element *NodeType* by adding attributes and subelements. It also extends the Java class *NodeView* to specify what connections to other nodes are allowed and where the subelements are to be placed. At last, a small graphic description, again in XML, defines the visualization of a buffer component.

With the information provided by the net class, the editor can create a *view* on a given XML document within an editor frame. A view starts from an XML element extending the *NetType* element and shows all its subelements by using their specified graphical representation. According to the XML schema of the net class, subelements, defining nodes, connections or text fields, can be added or removed and their attributes can be set. These operations directly affect the underlying XML structure but keep the document well-formed and valid.

5.1.2. Getting Started

[Chapter 4](#) provides all information needed to install the editor and to locate the files that come with the installation package. To bring up the editor just type *cinsim-gui* within a shell or create a link on your desktop.

Starting the editor brings up an initial window. By selecting the only entry *File* from the menu bar at the top of the window the small menu shown in [Figure 5.1](#) appears. You can create a new file by choosing the first entry *New* or open an existing file by choosing the entry *Open*. By selecting *Open Recent File* a popup menu will appear on the left, where you can choose recently opened files. The entry *Settings* opens a dialog to change several editor related settings. The last entry *Exit* of the menu will close the editor. If there are any unsaved files opened, a message dialog will be displayed.

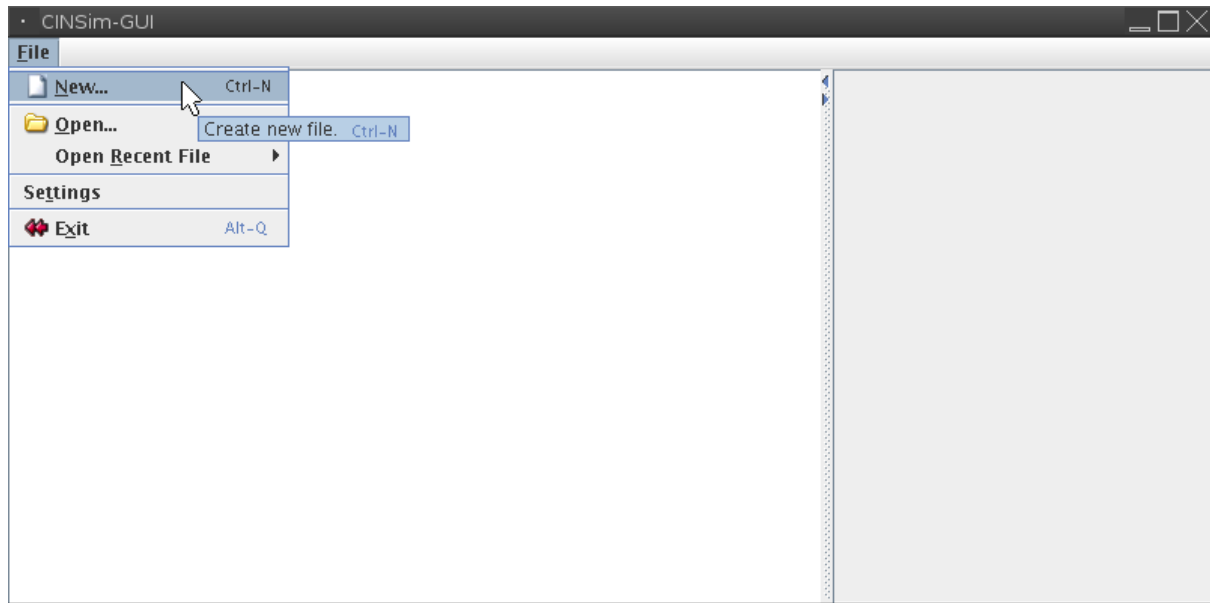


Figure 5.1. Initial editor window

The dialog for the editor setting is shown in Figure 5.2. The entry *file-open-dialog-path* specifies the path where the file open dialog, displayed when an existing file is to be opened, will be located. If you move to an other directory to open a file, this entry will be set to the new directory. So you usually won't need to set this entry by hand.

The second entry *schema-location-base-path* defines where the schema files to the available net class extensions of the editor framework can be found. The installation process will set this entry to locate the schema file *CIN-SimPlus.xsd* of the net class CINSimPlus. The specified path must also contain a subdirectory *builtin* holding the schema file *net-objects.xsd* of the framework, defining the base types of all net components, and a schema file *graph.xsd*, defining the graphic elements used for the graphic descriptions of the net components. Again, there is usually no need to set this entry by hand.

In the lower right corner of the dialog there are three buttons to accept or dismiss changes or to load the default settings that were set during the installation process. In the HOME directory the file *.cinsim-gui.conf* will be created that holds the current editor configuration including some more settings not shown in the table, for example, the window size.

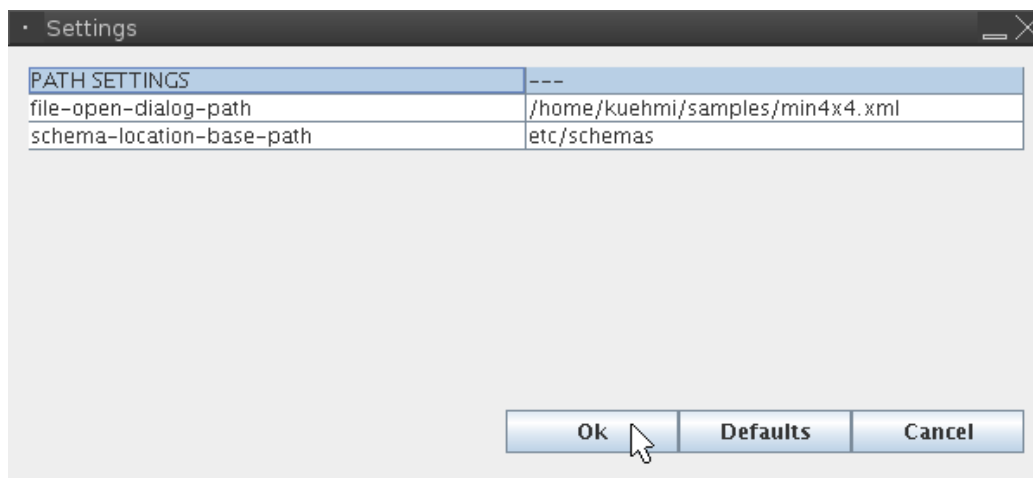


Figure 5.2. Editor settings



Important

If you have formerly used an older version of *cinsim-gui*, you should remove the configuration file before using the new version the first time. Alternatively, you can load the default settings for the new version via the settings menu.

Selecting the entry *New* from the *File* menu will open the dialog shown in [Figure 5.3](#). It asks you to choose the net class for the new net description. The combo box shows all XML schema files that are located in the specified *schema-location-base-path* without their file suffixes *.xsd*, assuming that a net class with the same name exists. As mentioned before, the net class that comes with the *cinsim-gui* package is called *CINSimPlus* and will be shown. If not, the button *Ok* will be disabled and you will first have to set a correct path. The button *Cancel* will close the dialog and abort creating a new file.

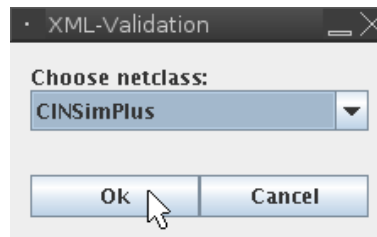


Figure 5.3. Choosing a net class

Opening an existing net description can be done in two ways: By choosing a file with a file chooser dialog as shown [Figure 5.4](#) or by selecting a recently opened file. In both cases the dialog for choosing a net class will be shown. Clicking *Cancel* will not abort opening the selected file but will leave the editor to choose the net class to be used. By setting the schema location path you can force the editor to use a different schema file than the one that comes with the package. Usually the net class *CINSimPlus* will be preselected and you can proceed by hitting the *Ok* button.

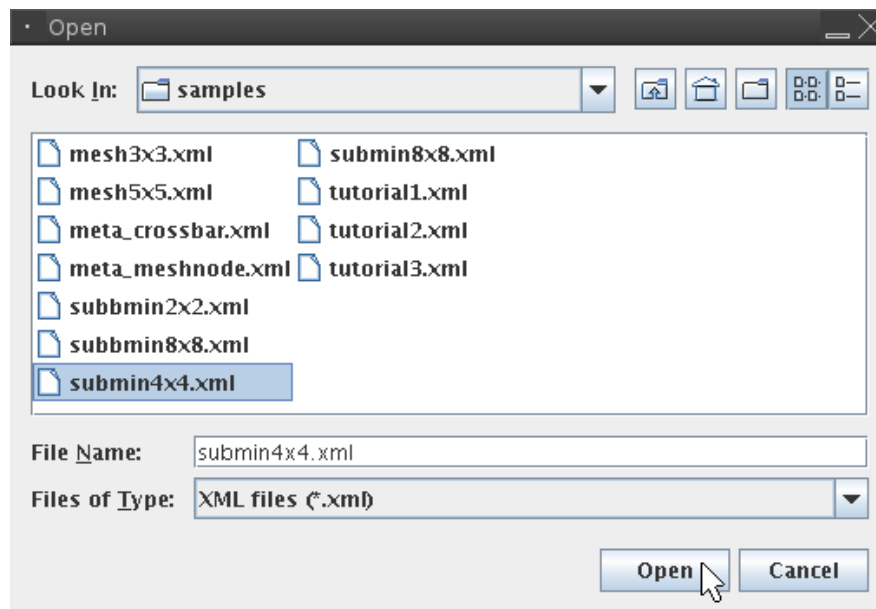


Figure 5.4. Choosing a file to open

5.1.3. Editing Net Class Files

Loading a new or an existing net class file will change the editor window as shown in [Figure 5.5](#). The main part of the editor window now holds a frame displaying the opened file, in the following called editor frame. The

former minimal menu bar is extended by some new available entries and two new tool bars are added to the editor window. The general features provided by the editor framework for editing net class files will be discussed in the following, special features and concepts added by the net class CINSimPlus in [Section 5.2](#).

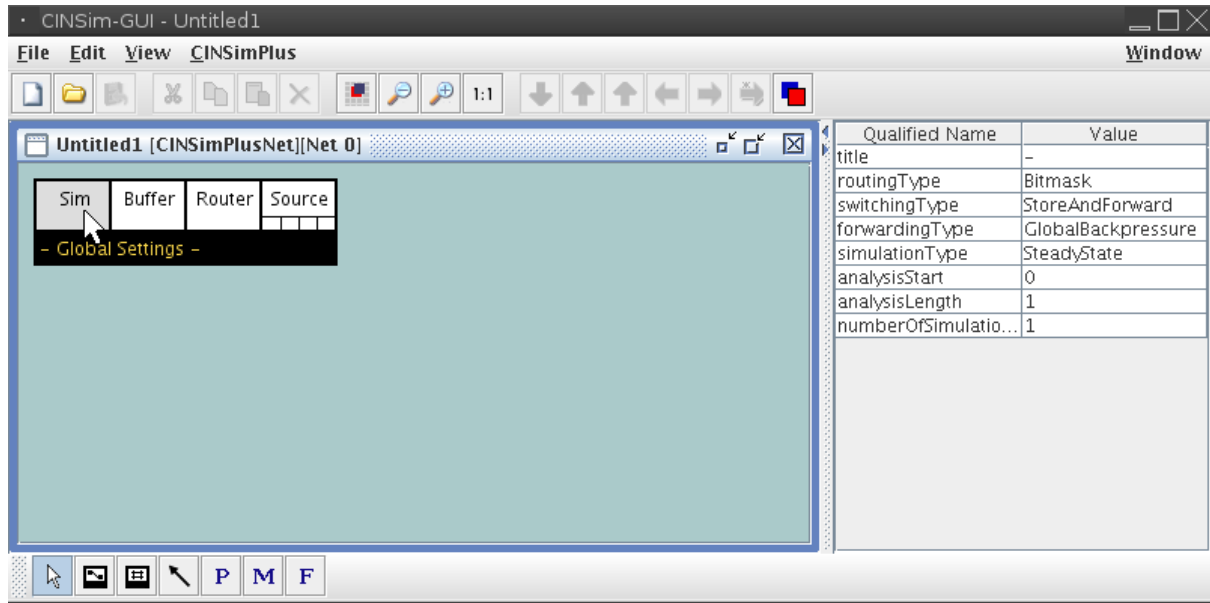


Figure 5.5. Editor window with new file

The *File* menu introduced in [Section 5.1.2](#) now contains three new entries ([Figure 5.6](#)). The entry *Save* can be used to save changes to an existing file, *Save as* opens a file chooser dialog to specify a name and a directory for an opened or unsaved file. Using *Save* on a previously unsaved file will open the file chooser dialog as well. The entry *Export to SVG* can be used to export a generated net graph as an SVG image.

The *Edit* menu ([Figure 5.7](#)) provides several standard actions useful for editing: *Cut*, *Copy*, *Paste* and *Remove*. These actions work with respect to the XML schema of the used net class. Therefore, not all objects that can be selected within the editor frame can be cut or removed. Cut, copy and remove action will be activated, if all elements of a selection within the editor frame can be removed. Cutting or copying one or more objects activates the paste action. Pasting works with respect to the used XML schema, too. Therefore, not all objects that can be cut or copied can be pasted within any editor frame. In these cases objects will be automatically removed from a selection.

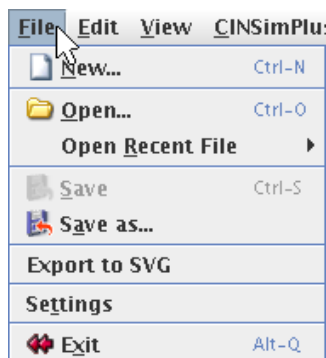


Figure 5.6. Editor framework - File menu

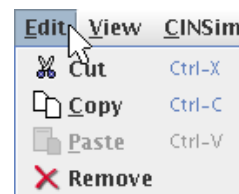


Figure 5.7. Editor framework - Edit menu

A net class extension to the editor framework may specify a hierarchical net structure, like the net class exten-

sion CINSimPlus (Section 5.2.1), allowing net elements to be nested within nodes of other net elements. The editor frame can only provide a view on one net, therefore ascending and descending actions must be defined to navigate through the various levels of a hierarchical net structure. The first entry of the *View* menu shown in Figure 5.8 can be used to create a new view on an already opened file in another editor frame. Within this editor frame you can move to a different level of the network structure to display more than one level within the editor window at the same time. The menu entries *Close* and *Close all views* can be used to close the currently focused view on a file or all of them. Closing the last view on a modified file will bring up a message dialog, asking you, whether the changes are to be saved or dismissed.

The entry *Align to grid* of the *View* menu aligns all elements of a selection to the grid. You can also activate the check box next to *Grid activation* to make newly inserted or moved elements already aligned. The entries *Scale down*, *Scale up* and *Scale to normal size* can be used to zoom in to a net graph and out of it again. If you favour angular handles over the round ones as show in Figure 5.14, you can activate the check box next to *Angular handles*.

The *Window* menu (Figure 5.9) shows the currently opened editor frames. They can be arranged using the entries *Cascade* or *Tile*.

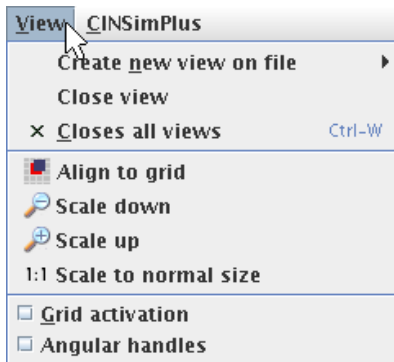


Figure 5.8. Editor framework - View menu

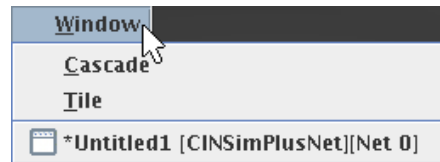
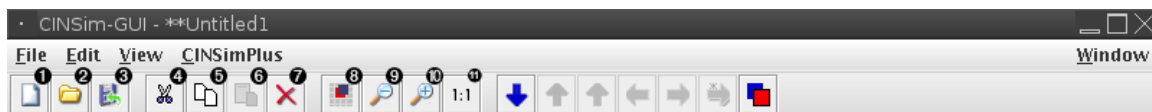


Figure 5.9. Editor framework - Window menu

Under the menu bar there is a tool bar holding action buttons (Table 5.1). The first group provides quick access to some entries of the *File* menu, the second group is a shortcut to the *Edit* menu and the third group a shortcut to the *View* menu. The last group belongs to the net class CINSimPlus that will be discussed in Section 5.2.



- | | |
|--|---|
| <ol style="list-style-type: none"> ❶ ❷ ❸ ❹ ❺ ❻ ❼ ❽ ❾ ❿ | <ul style="list-style-type: none"> Create a new file Open an existing file Save changes to an opened file Cut selected objects to clipboard Copy selected objects to clipboard Paste selection from clipboard Remove selected objects Align selected objects to grid Scale down the view Scale up the view Scale view to normal size |
|--|---|

Table 5.1. Editor framework - Tool bar



Tip

Moving the mouse cursor over menu entries and tool bar symbols will bring up short descriptions. The use of the shortcuts listed next to most menu entries may speed up your work.

The XML schema of the chosen net class is used for schema-driven editing. The tool bar added at the bottom of the editor window, called *insertion tool bar*, shows the components that can be added to the displayed net element according to the schema file. [Section 5.2](#) provides a detailed description of the insertable components. The symbol at the left does not represent an insertable component. It is used to set the *selection mode* of the editor as shown in [Figure 5.5](#). In this mode the left mouse button can be used to select and move objects displayed in the editor frame. Selecting another entry of the tool bar sets the *insertion mode*, where the left mouse button is used to insert new objects within the editor frame as shown in [Figure 5.10](#).

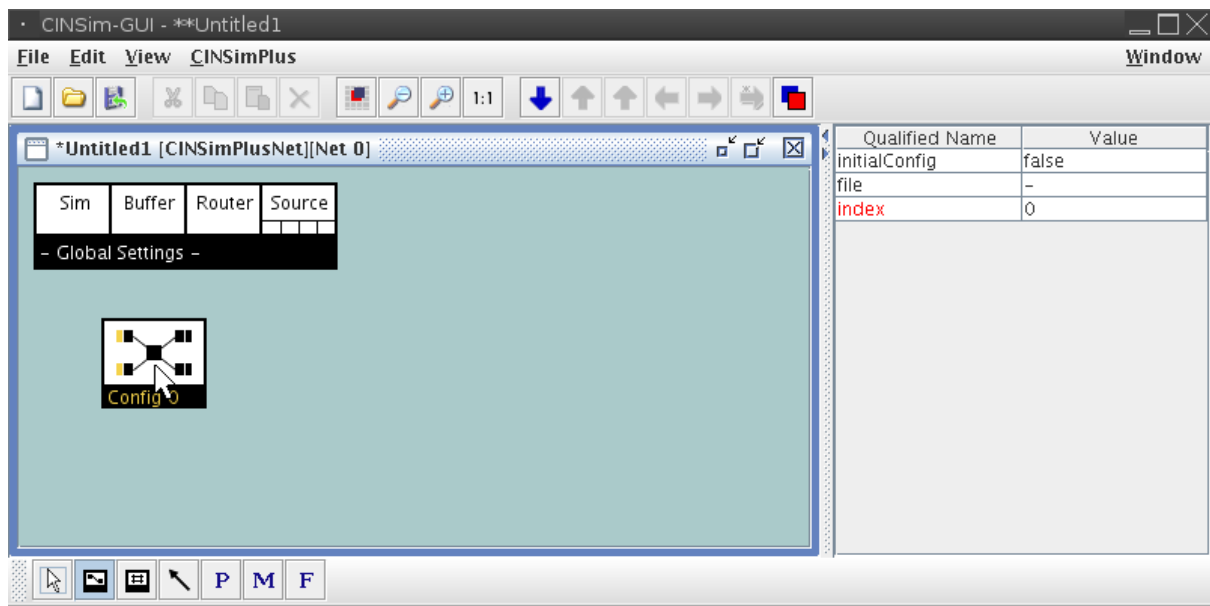


Figure 5.10. Inserting a new node

Only nodes and text fields can be inserted directly into the net element displayed within an editor frame by a left-click. The insertion of a connector starts by selecting the node to start at with the left mouse button and holding it down. The node will then be highlighted and a dashed line will follow the mouse cursor ([Figure 5.11](#)). If the mouse cursor is moved to a valid destination node, this node will also be highlighted ([Figure 5.12](#)) and releasing the left mouse button will insert the connector. The net class has to specify the rules for valid connections, otherwise a connector can connect any pair of nodes.

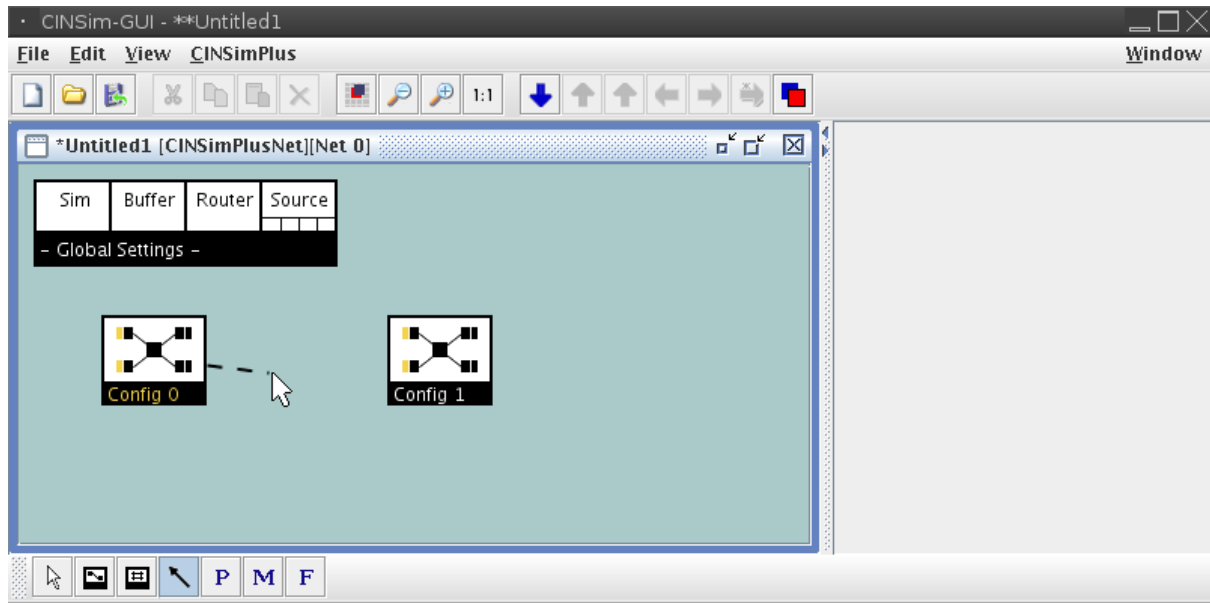


Figure 5.11. Inserting a connection (1)

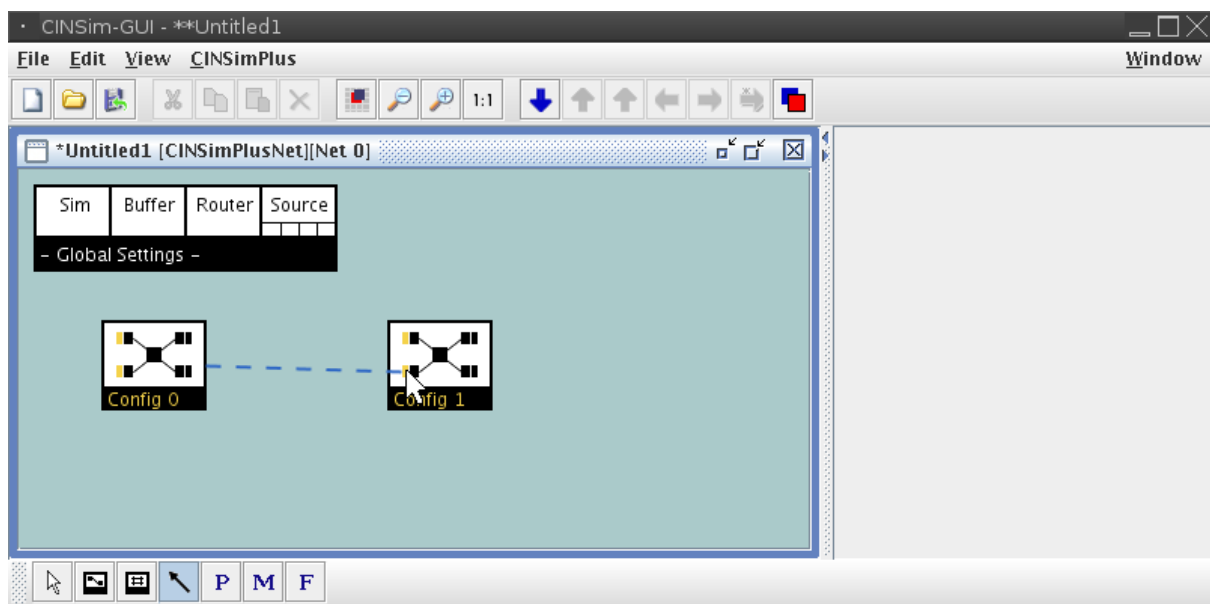


Figure 5.12. Inserting a connection (2)

After returning to the selection mode, you can move and select the inserted objects. Only nodes and text fields can be moved, connectors follow the nodes they connect. To move an object you have to move the mouse cursor to its location and press the left mouse button. While holding down the button you can move the object to the desired position. You can select more than one object by holding down 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 object from a selection by selecting it while holding down the ctrl-key. Moving one object of a multiple selection moves all the other objects, too. To remove one or more objects you have to select them within the editor frame and use the highlighted remove button from the tool bar at the top of the window.

If you select an object and then click it with the right mouse button, a small menu as shown in [Figure 5.13](#) will appear. The menu usually contains the entry *Remove* that can also be used to remove an object. The entry will be disabled if the object can not be removed.

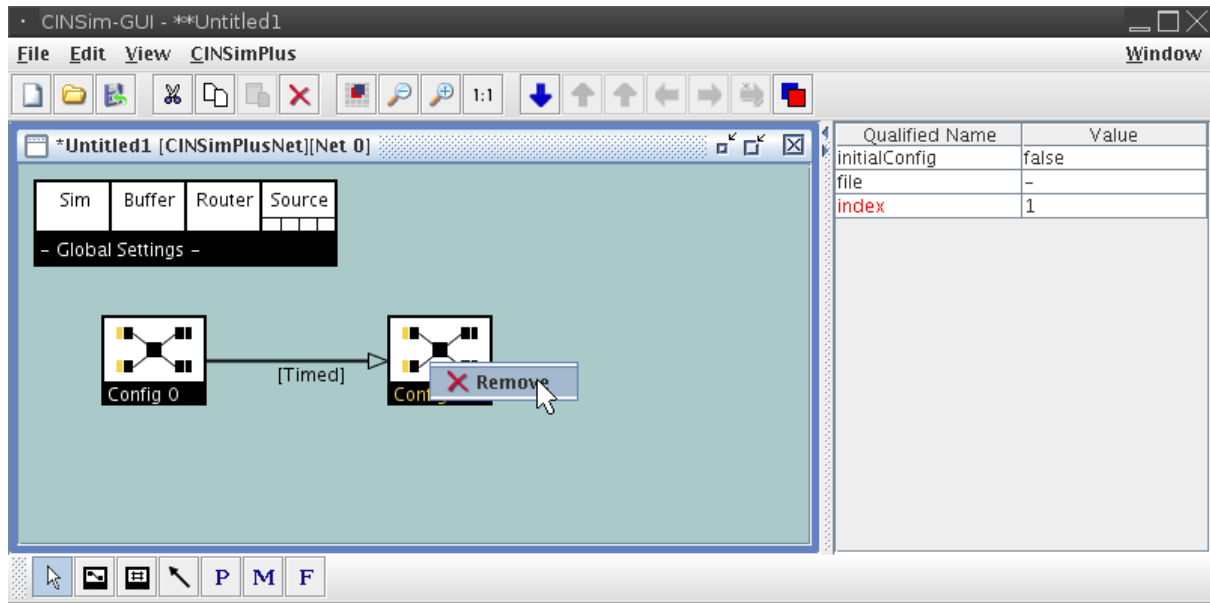


Figure 5.13. Default right-click menu

Double-clicking a connector inserts another net component - a handle. Handles along a connector can be used to redirect the connector to optimize the layout (Figure 5.14).

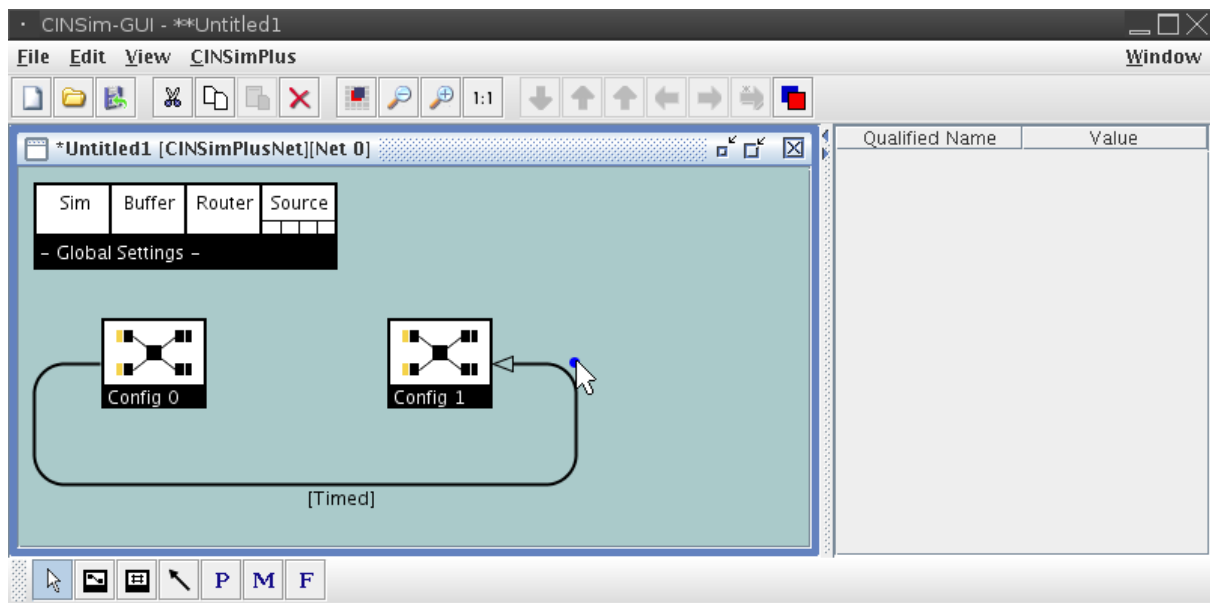


Figure 5.14. Using handles to redirect connectors

The area on the right side of the editor window holds the *attribute tables*. The XML schema of the net class specifies various attributes for most XML elements related to a net component. Selecting a single object within the editor frame brings up the attributes of the related element, showing the names on the left and the current values on the right. The schema specifies the value type and some other parameters related to an attribute. A red attribute name indicates that a value is required. Default values can be used if a value is needed but no user defined value is given. The attribute tables of newly inserted objects will show the attributes' default values if defined. The value type of an attribute can be a built-in type like an integer or a string, a pattern definition or an enumeration. Left-clicking the value field of an attribute with an enumeration type will turn this field into a combo box as shown in Figure 5.15, holding the specified enumeration values. Attribute values for builtin types or pattern

based types must be entered after double-clicking the related value using the left mouse button (Figure 5.16). Some attribute values can not be set, clicking the value field will neither bring up a combo box nor an editor prompt.

Qualified Name	Value
title	-
routingType	Bitmask
switchingType	VirtualCutThr... ▼
forwardingType	StoreAndForward
simulationType	Wormhole
analysisStart	VirtualCutThrough
analysisLength	PartialCutThrough
numberOfSimulatio...	1

Figure 5.15. Attribute with enumeration type

Qualified Name	Value
title	-
routingType	Bitmask
switchingType	VirtualCutThrough
forwardingType	GlobalBackpressure
simulationType	SteadyState
analysisStart	0
analysisLength	1
numberOfSimulatio...	1

Figure 5.16. Attribute with built-in or pattern based type



Note

Some value fields show a combo box and an editing prompt when selected. These are no enumerations specified by the XML schema, but possible values, for example defined variables, depending on the context.



Warning

Attribute values entered within the attribute tables are validated against the XML schema and ignoring warnings about invalid values may produce invalid documents. [Section 5.2](#) specifies what values are allowed for attributes that can be set.

5.2. Net Class Extension CINSimPlus

5.2.1. Hierarchical Structure

A single editor frame provided by the editor framework visualizes only a part of the XML document tree belonging to a net class file. This part always starts from an element extending the *NetType* element of the framework's XML schema and shows all subelements defining the net components given in [Section 5.1.1](#). For several reasons, the net class CINSimPlus goes beyond this simple approach and defines a hierarchical structure, allowing more than one net element. The tool bar at the top of the editor window holds buttons used for navigation within the hierarchical net structure ([Table 5.2](#)). They are part of the net class extension CINSimPlus and provide quick access to the menu entries of the *CINSimPlus* menu. If an action is applicable to the currently focused editor frame, the related symbol will be highlighted as shown in [Table 5.2](#), otherwise it will be grey.



- | | |
|---|---------------------------------|
| ❶ | Descend into subnet |
| ❷ | Ascend from subnet |
| ❸ | Ascend to top level |
| ❹ | Previous configuration |
| ❺ | Next configuration |
| ❻ | Copy/Clone subnet |
| ❼ | Open subnet in new editor frame |

Table 5.2. Net class CINSimPlus - Tool bar



Tip

Moving the mouse cursor over the tool bar symbols brings up short descriptions. The CINSimPlus menu also specifies short cuts to the available actions.

The top level net element of type *CINSimPlusNet* defines the simulation set. On this level global settings and variables are defined, but also network configurations. A network configuration holds a subelement that again specifies a net component used for setting up an interconnection network. The top level has to specify at least one configuration to define the simulation of a single network, but can also hold several more connected configurations defining reconfiguration steps. There are two different types of network configurations: One type, called *network configuration*, can be used to define any network structure based on the network components known to CINSim. The other type, called *mesh configuration*, is used to define regular mesh structures. [Table 5.3](#) shows the insertion tool bar related to a CINSimPlusNet and gives the names and component types of all insertable objects.

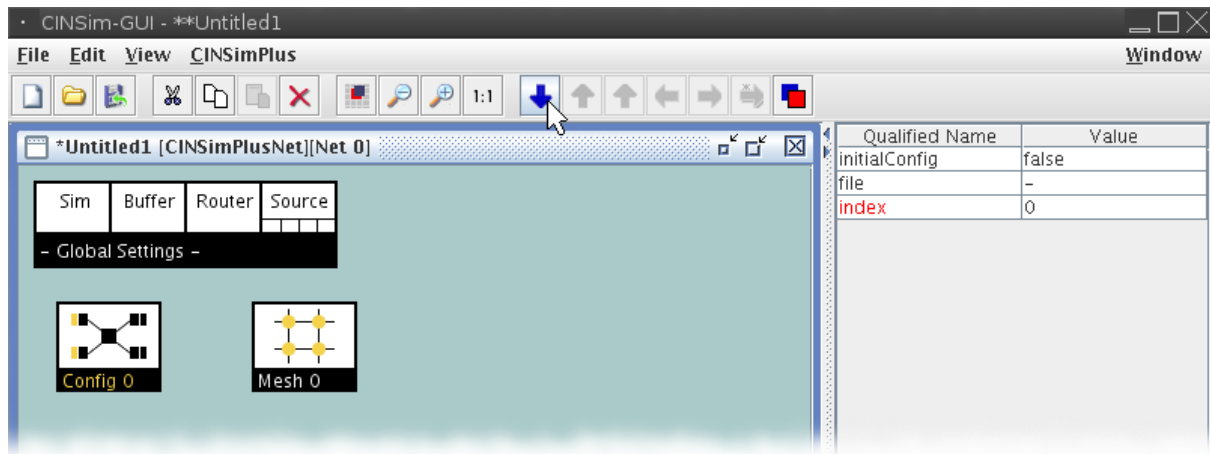


- | | |
|---|------------------------------|
| ❶ | Selection mode |
| ❷ | Network configuration (node) |

- ③ Mesh configuration (node)
- ④ Reconfiguration step (connector)
- ⑤ CINSim parameter (text field)
- ⑥ Measurement variable (text field)
- ⑦ Formula variable (text field)

Table 5.3. CINSimPlusNet - Insertion tool bar

The descend button (Table 5.2) of the tool bar will be activated, if an object with a nested net element is selected. At the top level, the objects related to network (*Config 0*) and mesh configurations (*Mesh 0*) hold nested net descriptions and selecting one of them activates the button (Figure 5.17). A left-click on the highlighted button moves the view of the editor frame from the currently displayed net to the nested net and activates several other buttons, for example, both buttons used for ascending (Table 5.2).

**Figure 5.17. Descending into network configuration**

If a network or mesh configuration object is connected to another configuration, the buttons for switching to the previous or the next configuration (Table 5.2) are also activated. So you can move along the specified reconfiguration steps without ascending to the top level. Directed connections for reconfiguration steps can only be inserted between network or between mesh configurations, not between network and mesh configurations. A configuration can be end point for several connections but start point for only one.

Within a network configuration, the insertion tool bar shows the available components for the displayed net element (Table 5.4). These components are specified by the XML schema, defining a net of type *CINSimNetType*, used for describing interconnection networks. CINSim only knows basic network components like *buffers* (*sources*, *targets*), *routers* and directed connections, called *routes*. More complex components like crossbars must be described using the given components. For this purpose there is a special component, called *meta element*, that again holds a nested net of type *CINSimMetaNet*, defining the complex structure with the available basic components.

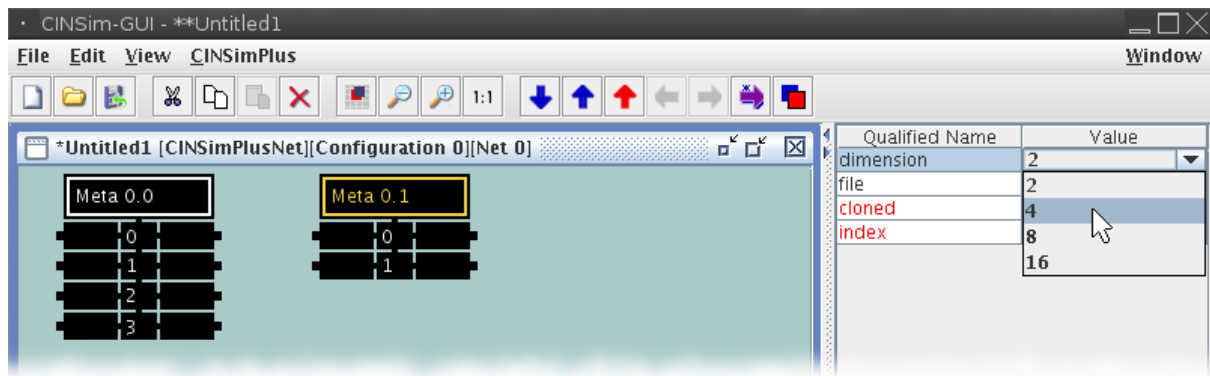


- ① Selection mode
- ② Local settings menu (node)
- ③ Meta element (node)
- ④ Source buffer (node)
- ⑤ Buffer (node)

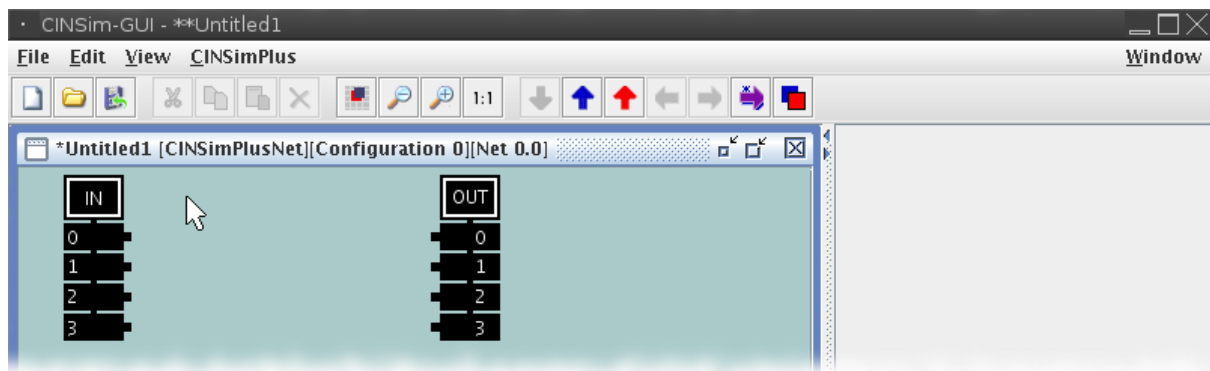
- ⑥ Target buffer (node)
- ⑦ Router (node)
- ⑧ Analyser (node)
- ⑨ Route (connector)
- ⑩ Observation (connector)

Table 5.4. CINSimNet/CINSimMetaNet - Insertion tool bar

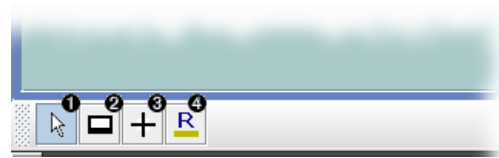
A meta element has a default dimension of two, allowing two ingoing and two outgoing routes connected to the numbered connection points. This is suitable to define a 2x2 crossbar, for example. If more connections are needed, the attribute *dimension* can be set up to 16 (Figure 5.18). Using meta elements for parts of a network that appear more than once, optimizes the layout of a network structure. One could, however, specify networks without using meta elements, as the CINSim parser flattens the XML structure anyway, but keeping a network as clearly arranged as possible leads to a better comprehension.

**Figure 5.18. Changing the dimension of a meta element**

The nested net of a meta element allows the same objects to be added as a nested net of a network configuration, but holds additional I/O panels for ingoing and outgoing connections as shown in Figure 5.19. These panels provide as much connection points, as the related meta element does, the numbers refer to the numbers of the related meta element. They can be moved to any desired position for an optimal layout, but can not be removed. Within a meta element there can again be meta elements, so the depth of the hierarchy is quite unbounded, depending on ones hardheadedness.

**Figure 5.19. CINSimMetaNet - I/O panels**

Meshes are special net topologies with a stronger focus on the position of net components. According to given positions, usually specified by coordinates, specialized routing strategies, like *xy* and *west-first*, can be applied. CINSim can handle regular two-dimensional meshes without obstacles. A nested net of type *MeshNet* within a mesh configuration object (Figure 5.17) can hold mesh nodes and directed routes to connect them (Table 5.5).



- ❶ Selection mode
- ❷ Local settings menu (node)
- ❸ Mesh node
- ❹ Mesh route

Table 5.5. MeshNet - Insertion tool bar

Mesh nodes are special kinds of meta elements with a fixed dimension of four and an additional attribute *coords* that specifies the coordinate. The coordinates shown in the attribute table and next to the node (Figure 5.20) are valid for all nested elements. A mesh node also holds a nested net of type CINSimMetaNet as shown in Figure 5.19 but the numbers now represent the directions of the outer node. The ordering is clockwise and connections from and to the north are related to the connection points on the I/O panels labeled with 0.

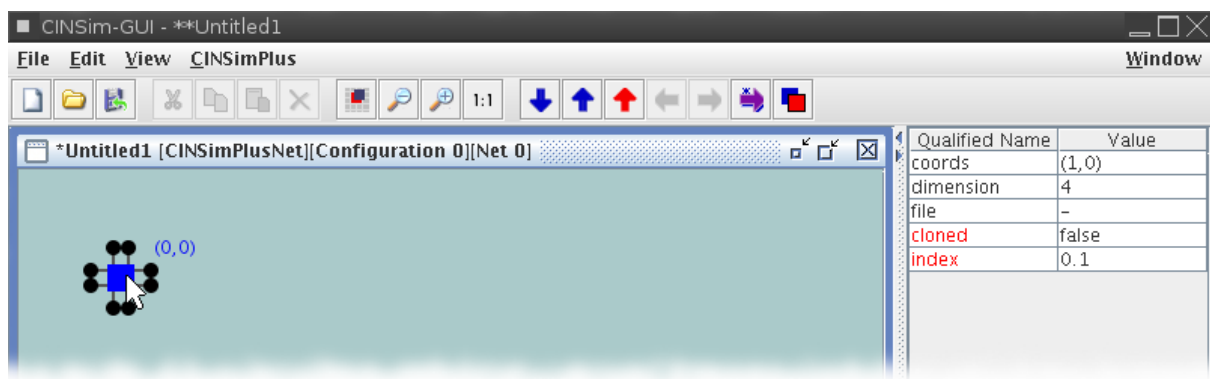


Figure 5.20. Changing the dimension of a meta element

5.2.2. Local/Global Settings Menu

A top level net element of type CINSimPlusNet is the starting point for a simulation set. The XML schema of the net class specifies a required node component to appear once as its first subelement, holding additional subelements and attributes used for simulation setup. This node will be visualized, labeled with *Global Settings*, in the upper left corner of an editor frame and can neither be moved nor deleted. The zoned white areas of this menu-like object represent independent sections, each with specific settings. Selecting an area, brings up the available attributes as shown in Figure 5.21. Every subnet in the net class hierarchy can specify its own settings menu, labeled with *Local Settings*. These menus can overwrite some of the settings specified in upper level subnets. This is useful if you want to set up the components of a specific configuration or subnet quickly.

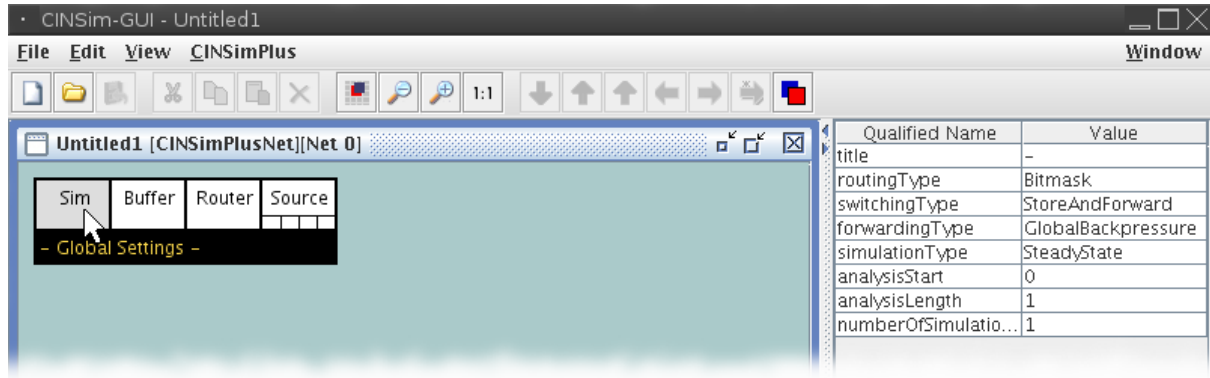


Figure 5.21. Simulation setup

The first section, labeled with *Sim*, is used to set the simulation preferences. *Routing* and *switching* strategy, the use of *local* or *global backpressure* and the type of the simulation can be specified by setting the related attributes. Table 5.6 shows their valid values according to the XML schema file and gives some general conditions to the accepted usage. Information to the meaning of each attribute value can be found in Part III. You can set a title string for a simulation to be used as a label in generated plots via the first attribute *title*. The last attribute *numberOfSimulations* is used to set up a simulation sequence. Specifying a value greater than one will force the simulation engine *cinsim* to start a sequence of simulations.

Attribute	Value description
title ^a	xsd:string
routingType	Bitmask ^b ShortestPath ^c StandardXY ^d WestFirst ^d
switchingType	StoreAndForward ^e Wormhole ^f VirtualCutThrough ^e PartialCutThrough ^e
forwardingType	LocalBackpressure GlobalBackpressure
simulationType ^a	SteadyState Terminating
analysisStart ^{ag}	[0-9] ⁺
analysisLength ^{ag}	[1-9][0-9] [*]
numberOfSimulations ^a	[1-9][0-9] [*]

Table 5.6. Valid attribute values for simulation setup

^a Only available in *Global Settings* menu.

^b Works only for non-cyclic net topologies.

^c Does not work for meshes.

^d Works only for meshes.

^e The size of a packet must not exceed the physical space of any buffer component.

^f Works only for unicast.

^g Only needed if attribute *simulationType* is set to *Terminating*.

Network components can be set up with common properties by using the remaining areas of the menu object.

Most of the available attributes can be redefined by using the attributes of the components itself, but if they are left unset, the global or local settings will be used. Furthermore, the global menu specifies default values, so you will usually not have to set all parameters of the network components, but only the parameters of interest. The field labeled with *Buffer* is used to set up buffer components, the field *Router* to set up router components and the field *Sources* to set up source buffers. Target buffers have no settings to be set as they only receive, analyse and delete flits. The following tables give a general survey over the available attributes and their values. The value descriptions for menu entries and the related components are the same. Information to the meaning of each attribute value can again be found in [Part III](#). The pattern *[a-z]* stands for CINSim parameters and the pattern *[f][0-9]+[/-]* for formulas. Parameters and formulas need to be defined globally and can then be selected from a combo box as introduced in [Section 5.2.4](#).

Attribute	Value description
sharedQueueTotal ^a	[0-9]+
sharedQueueMin ^a	[0-9]+
queueType	QueueOneSize
minSpace	[a-z] [1-9][0-9]*
maxSpace ^b	[a-z] [1-9][0-9]*

Table 5.7. Valid attribute values for buffer setup

^aFeature currently disabled, fixed value.

^bIgnored if less than *minSpace*.

Attribute	Value description
controllerType	FirstFit
schedulingType	Random FixedOrder RoundRobinLocal RoundRobinGlobal LeastRecentlyUsed MostRecentlyUsed LeastFrequentlyUsed MostFrequentlyUsed OldestPacket-First LongestWaitingFirst Deadline Priority

Table 5.8. Valid attribute values for router setup

The settings for source buffers are the most complex ones as these components are able to produce a wide range of distributions for newly generated packets, including distribution in time, multicast distribution and spatial distribution. More information on the proper setup of these distributions can be found in [Section 8.1](#).

Attribute	Value description
traffic	Geometric Periodic RandomBurst Pareto
multicast	Unicast Broadcast Uniform Formula Probabilities
mc_formula ^a	[f][0-9]+[/-] ^b
mc_leftBorder ^a	[0-9]+

Attribute	Value description
mc_rightBorder ^a	[0-9]+
mc_probabilities ^c	all=([0-9]+([.] [0-9]+)?) ([:] ([0-9]+[.]) * [0-9]+ [=] [0-9]+ ([.] [0-9]+) ?) *
spatialDistribution	Uniform Formula Probabilities
sd_formula ^d	[f][0-9]+[-] ^b
sd_leftBorder ^d	[0-9]+
sd_rightBorder ^d	[0-9]+
sd_probabilities ^e	all=([0-9]+([.] [0-9]+)?) ([:] ([0-9]+[.]) * [0-9]+ [=] [0-9]+ ([.] [0-9]+) ?) *
priorityDistribution	Uniform Formula Probabilities
pd_formula ^f	[f][0-9]+[-] ^b
pd_leftBorder ^f	[0-9]+
pd_rightBorder ^f	[0-9]+
pd_probabilities ^g	all=([0-9]+([.] [0-9]+)?) ([:] ([0-9]+[.]) * [0-9]+ [=] [0-9]+ ([.] [0-9]+) ?) *
maxPriority	[0-9]+
deadline	[0-9]+
packetSize	[a-z] [1-9][0-9]*

Table 5.9. Valid attribute values for source buffer setup

^aOnly needed if attribute *multicast* is set to *Formula*.

^bDefined formulas can be selected.

^cOnly needed if attribute *multicast* is set to *Probabilities*.

^dOnly needed if attribute *spatialDistribution* is set to *Formula*.

^eOnly needed if attribute *spatialDistribution* is set to *Probabilities*.

^fOnly needed if attribute *priorityDistribution* is set to *Formula*.

^gOnly needed if attribute *priorityDistribution* is set to *Probabilities*.

Every source buffer component in CINSim is set up with a component defining the distribution in time to be used, according to the attribute *traffic* (Table 5.9). Under the area labeled with *Sources* there are four unlabeled areas. These areas can be used to set up common properties for the available traffic distribution functions. The first attribute *trafficType* has a fixed value and specifies the related distribution. All attributes and their valid values are given in the following.

Attribute	Value description
trafficType	Geometric
load	[a-z] [0](.[0-9]+)?[1](.[0])?

Table 5.10. Valid attribute values for geometric distribution setup

Attribute	Value description
trafficType	Periodic
offset	[a-z] [0-9]+
modulo	[a-z] [1-9][0-9]*

Table 5.11. Valid attribute values for periodic distribution setup

Attribute	Value description
trafficType	RandomBurst
burstLoad	[a-z] [0](.[0-9]+)?[1](.[0])?
burstIntervalMin	[a-z] [0-9]+
burstIntervalMax	[a-z] [0-9]+
gapLoad	[a-z] [0](.[0-9]+)?[1](.[0])?
gapIntervalMin	[a-z] [0-9]+
gapIntervalMax	[a-z] [0-9]+

Table 5.12. Valid attribute values for random-burst distribution setup

Attribute	Value description
trafficType	Pareto
load	[a-z] [0](.[0-9]+)?[1](.[0])?
onShape	[a-z] [0-9]+(.[0-9]+)*
offShape	[a-z] [0-9]+(.[0-9]+)*

Attribute	Value description
-----------	-------------------

Table 5.13. Valid attribute values for pareto distribution setup

5.2.3. Network Components

A simulation set must always contain at least one interconnection network description, nested within a network or mesh configuration object (Section 5.2.1). More than one configuration object can be used to define reconfiguration steps, described in detail in Section 5.2.7. A network description consists of basic components defining sources to create the network traffic formed by constant size packets, buffers to store packet fragments (flits), targets to send packets to and routers to route packet heads according to their targets. These components are connected by directed wires, specifying the desired network topology. The simulation core does not know anything about topologies, any description will be transformed into an abstract graph representation. Therefore, the user has to take care of the right routing strategy, following Table 5.6.

The network components are available from the insertion tool bar of any displayed net element of type CINSimNet and CINSimMetaNet (Table 5.4). To insert one of the node components, the related symbol has to be selected and a left-click will insert the desired object to the editor frame. A source buffer component, as given in Figure 5.22, defines a source. Selecting the white area brings up the related attribute table on the right side of the editor window. Attributes and valid values follow the specification given in Table 5.9, but are initially unset, falling back to the global settings related to source buffers. The attribute *traffic* sets the distribution in time, visualized by the yellow area of a source buffer object. This area is a nested element and selecting it will bring up the attribute table for the chosen traffic distribution. Changing the distribution to be used, will change this element and therefore the available attribute table. The labels follow the chosen distribution: G - Geometric, P - Periodic, RB - RandomBurst, P - Pareto. Initially, there is no distribution set and therefore the distribution from the global or local settings be used.

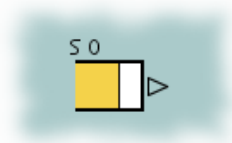


Figure 5.22. Source buffer element

All network components discussed in this section own two required attributes that can not be set manually: *cloned* and *index*. The attribute *index* holds a unique index string, starting with the index string of the net element a component was first introduced. The title bar of an editor frame shows the index string of a displayed net element in squared brackets prefaced by the string *Net*. All network components added to a net element as subelements are grouped according to their type specified in the XML schema file. Within a group the position of an element represents the insertion order. The last number of a component's index string reflects this order, whereas all elements with their attribute *cloned* set to *true* are ignored. This is needed to define reconfiguration steps, described in detail in Section 5.2.7. The label of any network component, except of meta elements, will show only the last number of the index string if the attribute *cloned* is set to *false*, otherwise the complete index string will be displayed.

Buffer components are quite similar in their visualization and their behaviour. The circular area on the left and the lace on the right represent so called connection points (Figure 5.23). A directed wire, called *route*, always starts at a start connection point and ends up at an end connection point. The circular area represents an end point, the lace a start point, allowing exactly one route to be connected. The available attributes of a buffer element follow the specification given in Table 5.8, for a target buffer element there are no attributes to set.

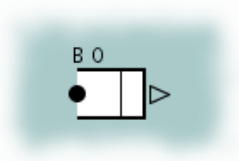


Figure 5.23. Buffer element

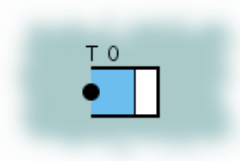


Figure 5.24. Target element

In contrast to the buffer-like elements, a router element, shown in Figure 5.25, combines both, a start and an end connection point. Therefore, routes can start and end at a router element and the number of routes connected to a router is unbound in both directions. The available attributes of a router element follow again the specifications of the related settings menu entry given in Table 5.9.



Figure 5.25. Router element

The insertion of a route has to be carried out as described in Section 5.1.3. Valid nodes for this kind of connection are the previously discussed connection points. In addition to the restrictions in number and direction of a route to be inserted, a route can not connect two buffer-like elements and two router elements. The only exception is a route between a source buffer and a buffer element. Figure 5.26 shows a small example following the given restrictions. The routes have to be inserted from left to right, connecting laces with circular areas, laces with router elements and router elements with circular areas.

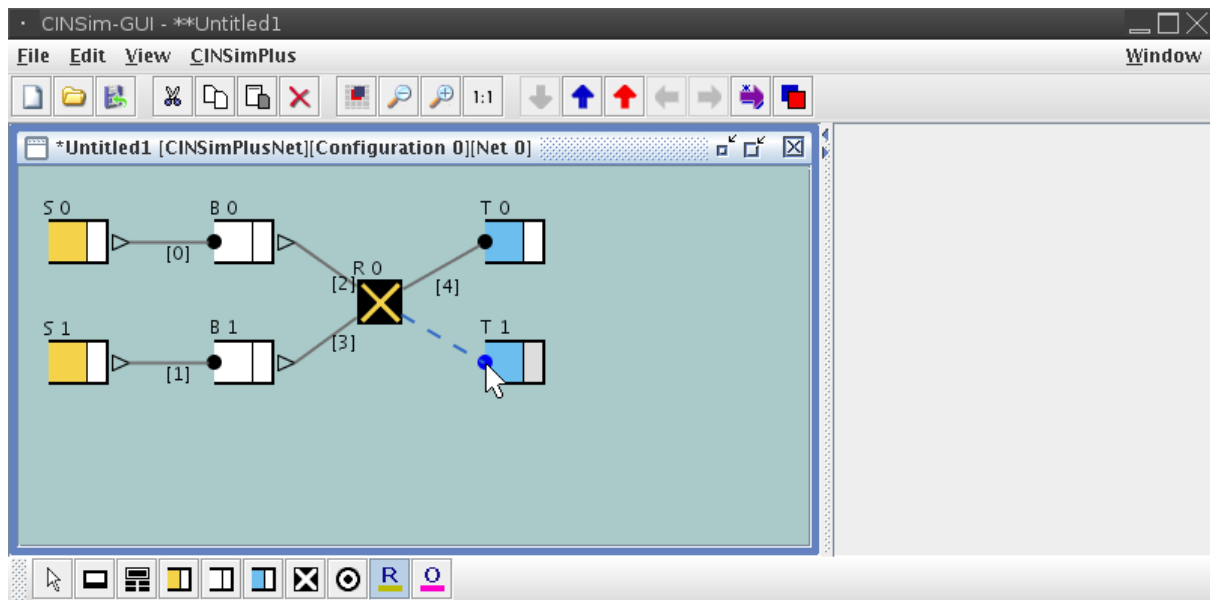


Figure 5.26. A small network example

Meta elements and I/O panels of their nested net elements, introduced in Section 5.2.1, provide several connection points, allowing routes to connect network components on different hierarchy levels. A meta element (Fig-

ure 5.27) has end connection points on the left and start connection points on the right. These connection points are represented by black rectangles on the left and on the right of the nested index numbers. The indexes and connection points appear again at the I/O panels of the nested net element (Figure 5.28) and set up a relationship between a meta element and the nested net element. A route ending up at connection point 0 (left) of a meta element, for example, can be continued at connection point 0 of the IN panel and a route ending up at connection point 0 of the OUT panel can be continued at the connection point 0 (right) of the meta element. Therefore, the IN panel holds the start connection points, the OUT panel the end connection points. It is not possible to connect connection points of both panels directly. Again, all connection points allow exactly on route to be connected.



Figure 5.27. Meta element



Figure 5.28. I/O Panels

Routes connected to one of the I/O panels will be labeled with - if no route is connected to the related connection point of the meta element, otherwise the (full) index of the connected route will be shown. The restriction of not connecting two buffer-like elements or two router elements will be assured. The attribute *twisted* of connection points at meta elements can be set to *true*, to make routes start or end at the opposite side of the meta element. For drawing, however, the connection point at the correct side must be selected.

Section 5.2.1 introduced another kind of meta element, the mesh node. To insert and arrange a mesh node, it is usually a good idea to activate the grid layout (Figure 5.8). The label of each mesh node shows its current coordinates, specified by the attribute *coords*. The first value defines the x-, the second one the y-coordinate. Moving mesh nodes changes their coordinates according to the current area of the editor frame. So they can and should be arranged with increasing values for both coordinates as shown in Figure 5.29, specifying rows and columns of a two-dimensional mesh. If the coordinates are not disjoint, the available routing strategies *XY* and *West-First* will not work. The circular areas around a mesh node represent connections points, the start and end points for the routes. For each direction there is a connection point for one ingoing and one outgoing route which fixes the dimension of this meta element to four. Selecting one of the connection points changes its colour. A directed route has to start at a yellow point and to end up at a blue one as shown in Figure 5.29. Connecting nodes in both directions sets up a bidirectional connection between mesh nodes.

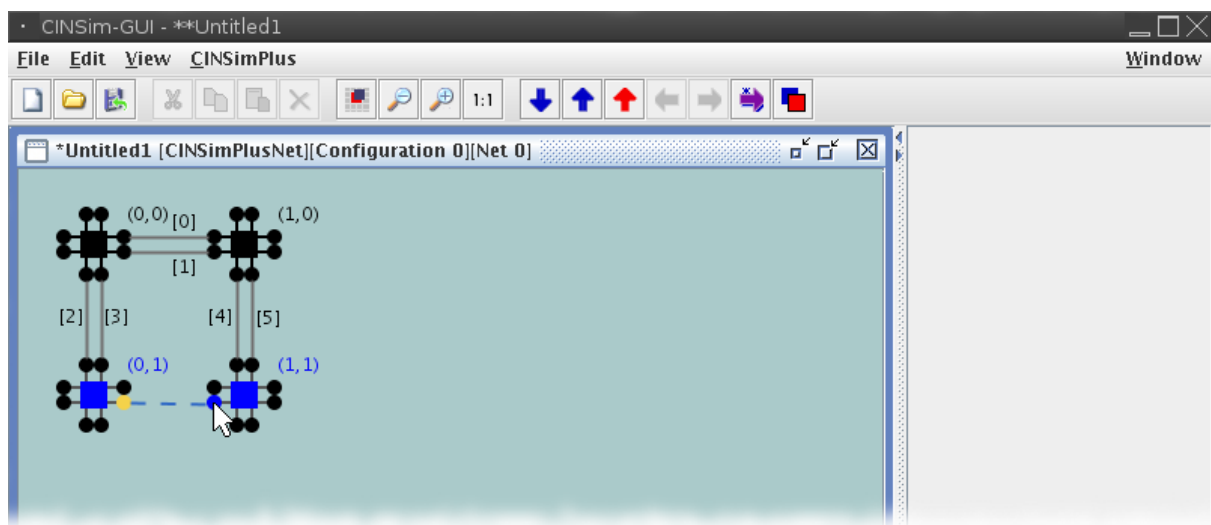


Figure 5.29. Connecting mesh nodes

The support for meshes is still an experimental feature of CINSim. The layout of the nested network is very restricted. There should only be a single router, that has outgoing connections to buffers in all four directions (in other mesh nodes) and to local target(s) only. Otherwise, the available routing algorithms based on coordinates will fail and leave the simulation in a state not suitable for analysing. A layout that can be used is given in [Figure 5.30](#) and can also be found among the sample files that come with the GUI.

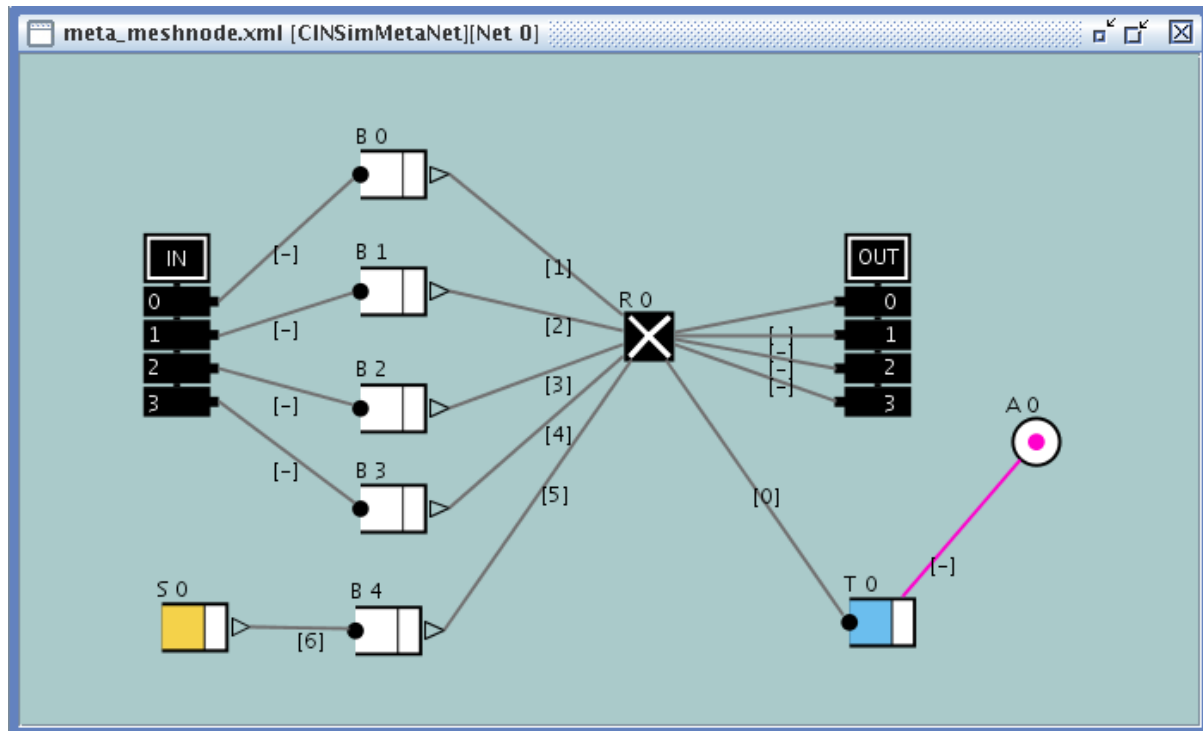


Figure 5.30. Valid layout of a mesh node



Note

[Section 5.2.6](#) introduces the insertion of network descriptions from external files. This might ease up the setting up of a mesh configuration.

5.2.4. Using parameters and formulas

Within a sequence several parameters can be varied. For this purpose, *CINSim parameters* must be added to a simulation set, defining a start value and an increment. [Figure 5.31](#) shows the CINSim parameter *a* starting at 10 and raised by 2 after every simulation run of a sequence. CINSim parameters will be automatically named with lower case letters according to their order within the XML document, start value and increment can be integral or floating point numbers depending on the parameter to be varied.



Figure 5.31. Setting up a CINSim parameter

The value field of any attribute that can hold a parameter will turn into a combo box (Figure 5.32), showing all existing CINSim parameters. It is recommended to use only variables from this selection, as removing and re-naming of parameters will be passed to all referencing attributes.

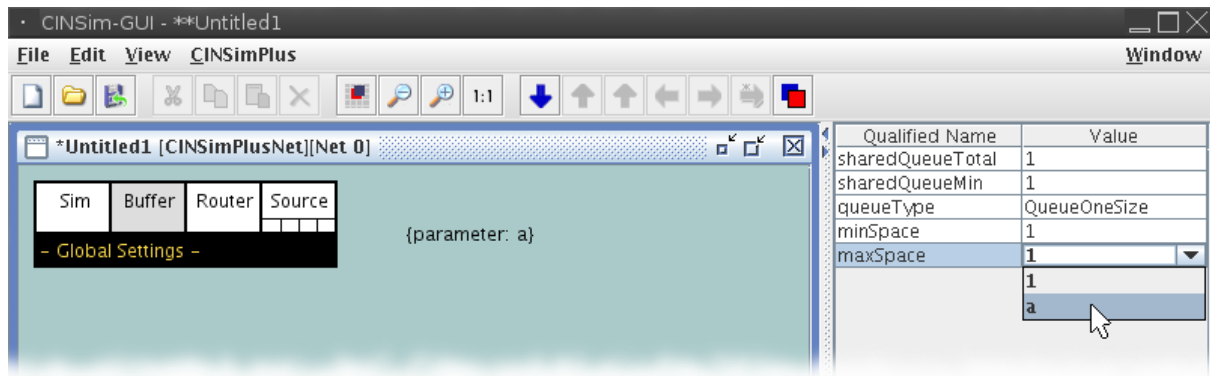


Figure 5.32. Using a CINSim parameter

Similar to parameters, *cinsim-gui* allows the definition and usage of formulas, that can be used to specify multicast, spatial and priority distributions of source buffers. Formulas are automatically numbered according to their order in the XML document and preceded by a small *f*. The attribute *definition* (Figure 5.33) holds the definition of the formula but can not be edited directly. To edit the definition you have to right-click the formula object in the editor frame and select the entry *Edit formula* as shown in Figure 5.33. This will open a small editor where the definition can be edited. Every line holds an expression and the last expression will be evaluated. The example in Figure 5.34 defines the function $f(t)=1/t$ which will be evaluated. The symbol *t* is a special expression and is used to specify the abscissa in function definitions. The complete syntax description can be found in Section A.1. If you close the editor by hitting *Accept* the new definition will be written to the attribute *definition*, whereas lines are separated by semicolons.

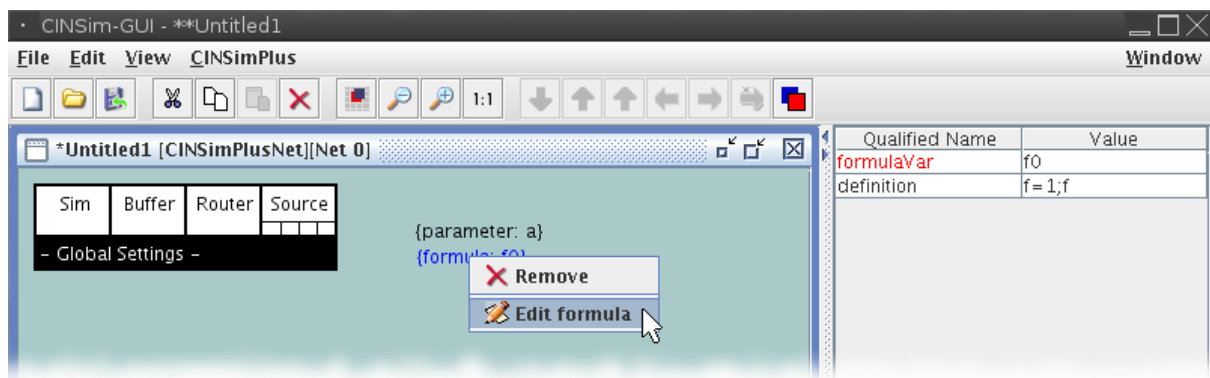


Figure 5.33. Setting up a formula (1)

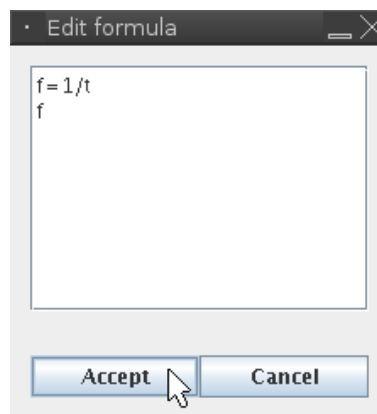


Figure 5.34. Setting up a formula (2)



Important

The use of formulas is still an experimental feature. The definitions will not be checked in the GUI, but when the simulation is prepared. At the moment it is recommended to use only simple definitions.

Once a formula is set up, it can again be chosen from any attribute field that can hold a formula as shown in [Figure 5.35](#). Currently, only source buffers use formulas to specify distributions and in all cases a related attribute must be set to *Formula* as documented in [Table 5.9](#). Then the formula will be used to determine the cumulative probabilities for the distribution of interest within the borders specified by the related attributes ending up with *leftBorder* and *rightBorder*. If, for example, a multicast with four targets shall be set up, the unicast is placed on the left border and the broadcast on the right border. The remaining cases are uniformly distributed between the borders under the graph of the formula.

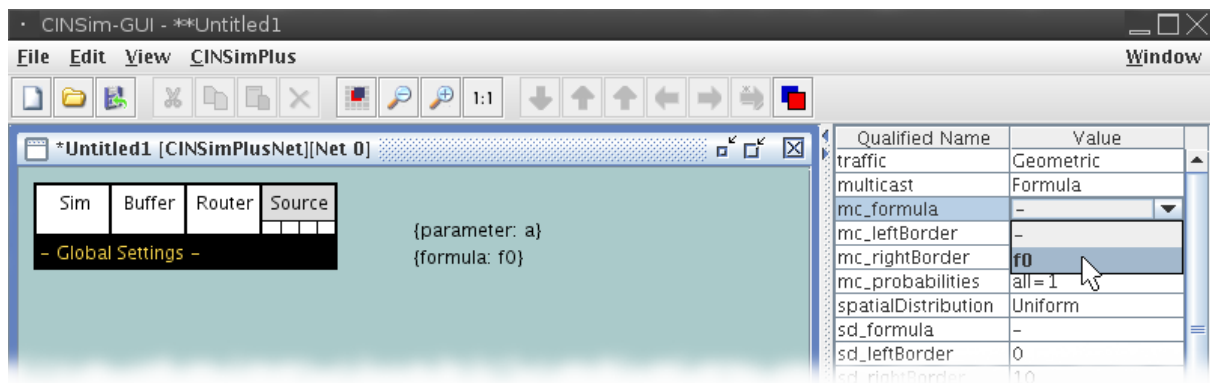


Figure 5.35. Using a formula

5.2.5. Defining Measurements

In addition to a network description, a simulation set needs at least one measurement, otherwise a simulation run will neither terminate, nor yield any result. Measurements are automatically handled by CINSim, determining whether a steady state for every single measurement is reached and the simulation run can be terminated with respect to a given precision and confidence. For this purpose, measurement variables have to be specified at the top level of a simulation set ([Table 5.3](#)) and are automatically numbered sequentially, starting with variable 1. The ordering follows the order of the elements within the underlying XML document. Every measurement in CINSim consists of two parts: On the one hand a series of data needs to be gathered according to a specific type of observation. The type can be set using the attribute *observationType* of an inserted measurement variable as shown in [Figure 5.36](#). Currently, the throughput of the sources and the targets, the queue length in a buffer component and the packet delay/latency can be observed. The available attribute values from the combo box should be self-explanatory. On the other hand the data series is analysed online to estimate the mean value or a p-quantile. The type can be set using the attribute *measurementType* of a measurement variable ([Figure 5.36](#)). The entries *Mean* and *Quantile[p]* are available from a combo box. In case of a p-quantile estimation a value for *p* from the interval (0,1) can be defined by setting the attribute *p*.

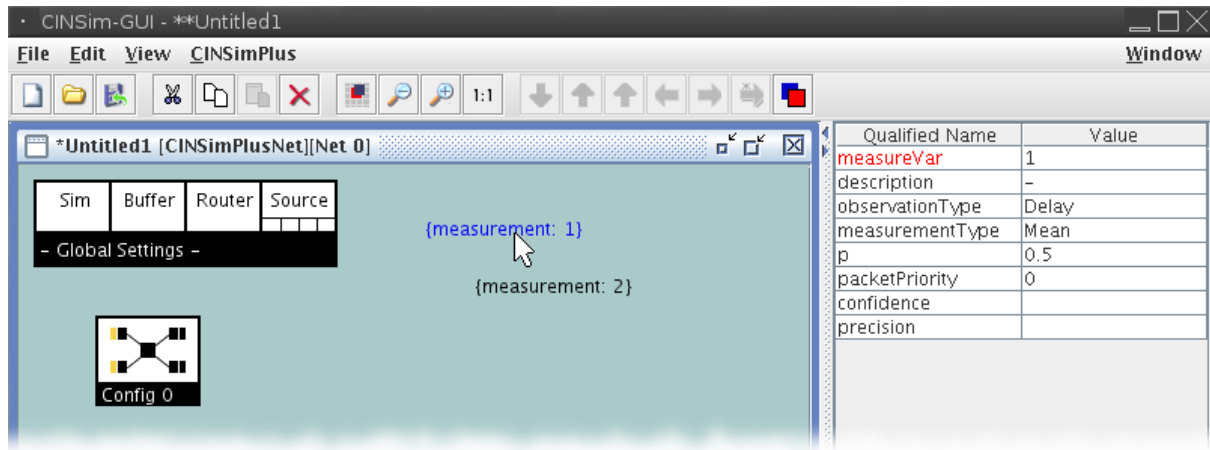


Figure 5.36. Measurement variable setup (1)

Choosing an observation type with prefix *Priority* enables a filtering of the packets that should be taken into account for observation data. The attribute *packetPriority* sets the priority of the packets to be observed. A source buffer can create packets with different priorities according to a user defined distribution, described in detail in [Section 8.1](#). The attribute *maxPriority* of the global settings entry *Source* sets the upper bound for packet priorities, a value greater than zero activates the priority distribution for newly created packets. Assuming that the value is set to four, the combo box related to the attribute *packetPriority* will allow priorities from zero to four to be chosen. If the maximum priority is decreased, the priority values for all measurement variables will be set back to zero. The example in [Figure 5.37](#) shows the setup of a 0.95-quantile estimation of the packet delay for packets of priority 3.

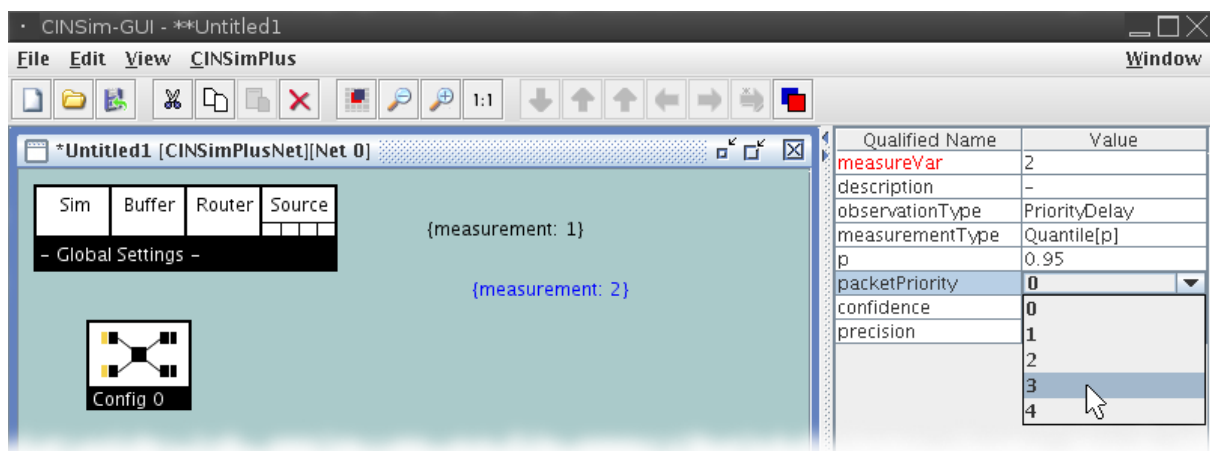


Figure 5.37. Measurement variable setup (2)

The last two attributes *precision* and *confidence* can be used to specify the relative precision and the confidence level for the measurement. These parameters can be set for all measurements at the command line when starting a simulation run. However, when a specific setting for a single measurement is needed it can be set here. Usually these fields can be left empty.

Setting up an measurement variable specifies only the type of measurement, but not what network components are to be involved in the measurement. The previous [Section 5.2.3](#) described how to set up a small network description ([Figure 5.26](#)) with the available network components. After setting up a measurement variable at the top level every nested network description can refer to this variable. For this purpose, a placeholder can be added to a network description, called *analyser* ([Table 5.4](#)). The related value field of its attribute *measureVar* shows the measurement variables separated by semicolons. An analyser can refer to more than one variable at a time. To add or remove a variable you have to right-click the analyser object in the editor frame and select *Add variable* or *Remove variable* which will show a selection of available measurement variables ([Figure 5.38](#)). For

every entry the variable number, the type of observation and the priority (in brackets) is displayed. If a measurement variable is removed or (automatically) renamed, the referencing analyser elements will be set back or updated.

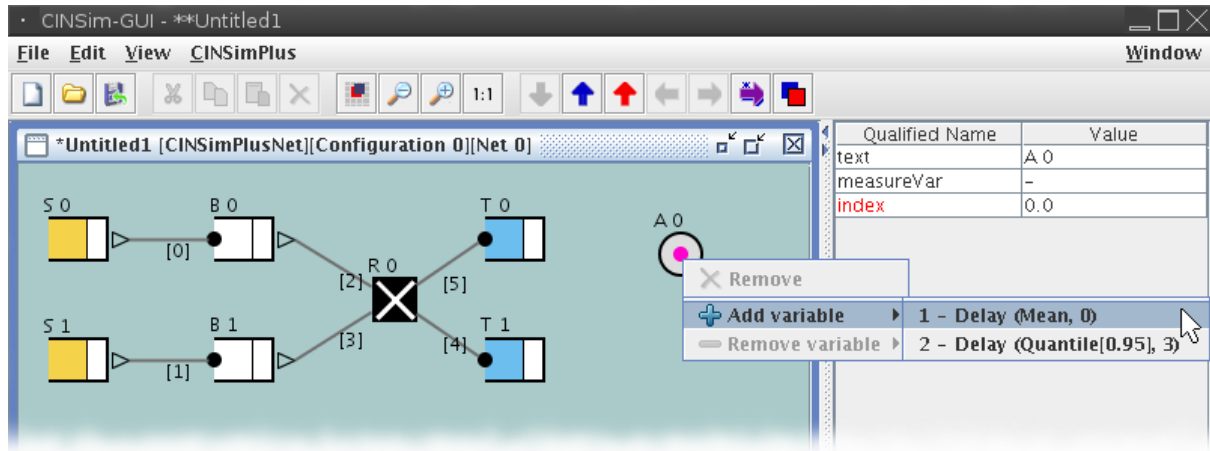


Figure 5.38. Analyser setup

The placeholder can now be connected to the components to be observed. The insertion tool bar holds a connector designated for this purpose, called *observation (line)* (Table 5.4). This connector always starts at an analyser element and ends up at the component to be observed, labeled with the related measurement variables. A single analyser element, or strictly speaking the related measurement variables, can be used to observe a loose number of components. To observe the average packet delay of all packets passed through the network, for example, all targets must be connected to an analyser element set up to observe the packet delay. Observing the priority delay as shown in Figure 5.39 restricts the measurement to packets with the given priority. Currently, only observations for buffer-like components are available and the user has to assure, that the involved components are suitable for a measurement. Observing the source throughput of a target, for example, does not really make sense. More information to the available observation types can be found in Part III.

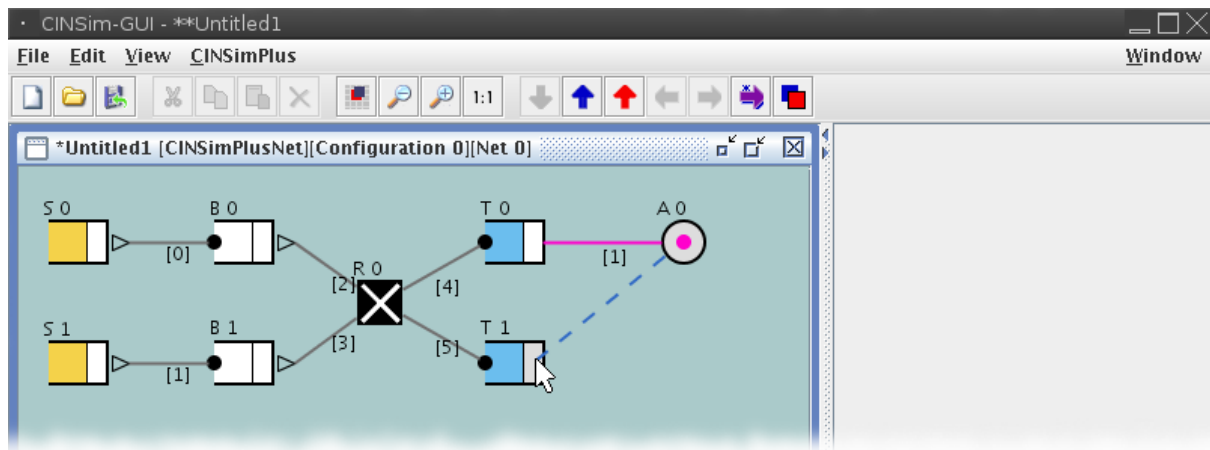


Figure 5.39. Observation

5.2.6. Deriving Net Descriptions

The tool bar extensions of the net class CINSimPlus (Table 5.2) not only provide quick access to actions for navigating within the hierarchical structure of a simulation set. There are also actions for creating nested net descriptions from existing net elements. The easiest way, however, is the use of the copy-paste feature of the editor framework. You can cut or copy a node with a nested net element and paste it several times to quickly enlarge the simulation set. This will always copy the nested net element and all its net components. The actions provided by the CINSimPlus extension allow you to copy only a selection of net components or to access net elements of external documents.

If an editor frame shows a nested net element, a button for creating a new net element (Figure 5.40) from the displayed one is activated. The related action creates a new node for the upper level net element and adds a copy of the visible net structure as its subelement. The type of the new node depends on the type of the parent node of the net element to be copied. The action forces the editor frame to ascend from the nested net to display the upper level net element. The newly inserted node, holding the created net copy, will be located in the upper left corner of the editor frame (Figure 5.41).

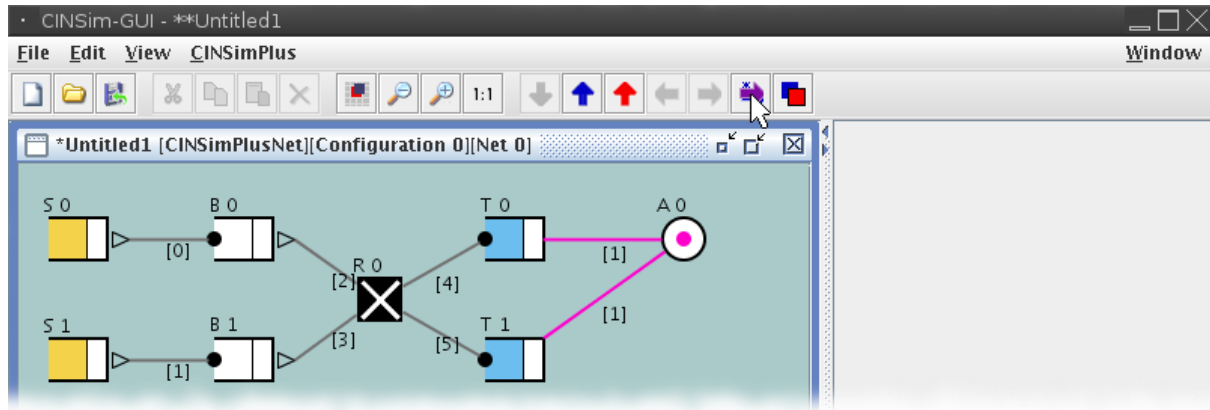


Figure 5.40. New nested net element from draft (1)

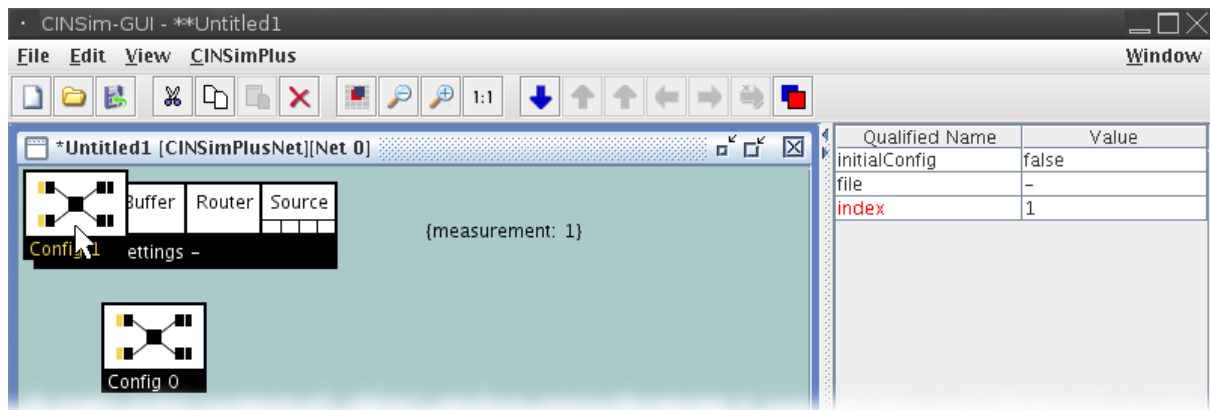


Figure 5.41. New nested net element from draft (2)

The example above shows how to create a new network configuration from a given one, but it works for all the other nested net descriptions and their parent nodes as well. Creating a new net from existing net elements of network and mesh configurations, however, differs in one important aspect, described in detail in Section 5.2.7: The indexes of the copied net components will remain the same, used for the specification of reconfiguration steps. For net elements within meta elements and mesh nodes, regular copies will be created, comparable to the copy-paste feature. At first glance this may seem redundant, but instead of copying all net components, one can only select some fragments to be copied, too. If at least one net component is selected, the editor will check, whether the selected objects can be added to a new net element, according to the rules of the net class CINSimPlus. A connector, for example, can only be added, if the associated nodes are added. Connection points, handles and labels are subelements of nodes and connectors, not of net elements and will be removed from a selection. If the validity check leads to an empty selection, all net components will be copied by default.

A different approach from re-using net elements within the same document is to put them to external documents and insert them from there. The last button of the tool bar (Table 5.2) is always activated and can be used to create a new document, starting from the visible net element of the currently focused editor frame. The new document will be opened in an own editor frame and can be saved to an independent XML file. At the top level, this will lead to a situation comparable to saving a simulation set to a new XML file and open it. But if you have a description of a 2x2 crossbar within a meta element as shown in Figure 5.42, you can, for example, create a new document only defining the given 2x2 crossbar (Figure 5.43).

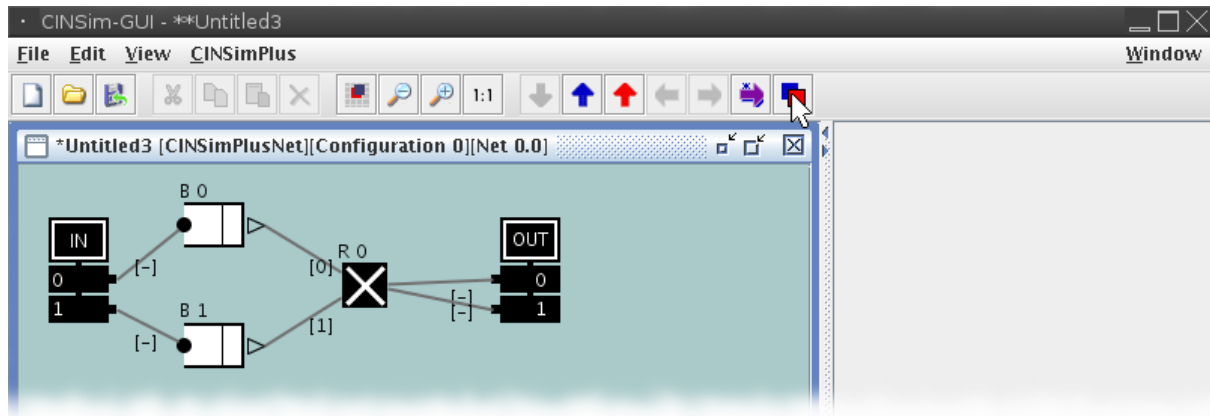


Figure 5.42. New document from draft (1)

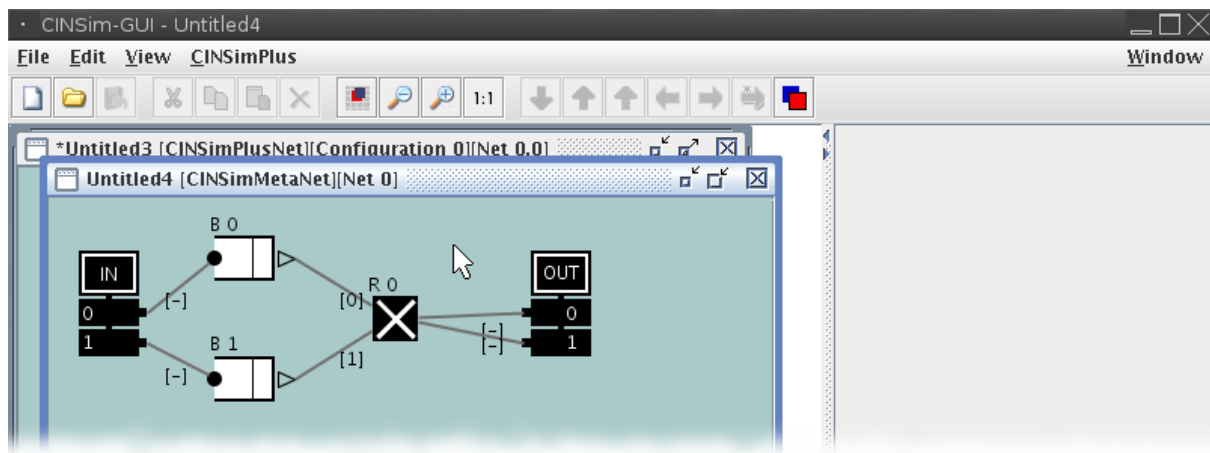


Figure 5.43. New document from draft (2)

In addition to the name of the document, the title of an editor frame shows some meta information in squared brackets. The first field displays the type of the top level net element, usually CINSimPlusNet for a complete simulation set. Creating new documents from nested net elements, will put them at the top level, therefore the title of the new editor frame in Figure 5.43 shows CINSimMetaNet. The simulation core will only accept regular simulation sets, titled with CINSimPlusNet, other documents can only be used for editing. Every node component holding a nested net element has an attribute *file* that can be used to insert a net element from an external document. Assuming that the 2x2 crossbar description was saved to a file named *meta_crossbar.xml* and is opened for editing, any meta element of dimension two can use this net element as shown Figure 5.44. This will replace the existing net element by the one of the external document.

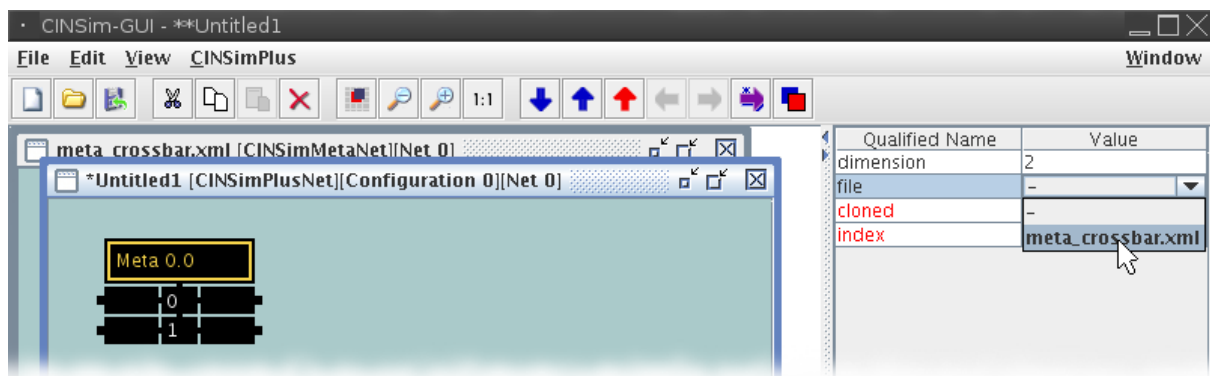


Figure 5.44. Inserting net element from external document

Whenever a node component with a nested net element is selected within the editor frame, all opened documents will be searched for suitable top level net elements. The [Section 5.2.1](#) gives an introduction to the various net element types and their parent nodes. To a meta element, for example, only documents with net elements of type CINSimMetaNet at the top level can be inserted and will therefore be available from the combo box related to the file attribute. Furthermore, the dimension of a CINSimMetaNet must fit the dimension of the new parent node, otherwise it will not appear in the combo box. It is not necessary to save a newly created document to a file before it can be used for insertion, but it may help to identify the desired net element to be inserted.

5.2.7. Reconfiguration Steps

CINSim is able to handle *dynamic* reconfiguration, thus allows to change the network layout or behaviour during operation. The CINSim core includes a dynamic-proof routing for packets that remain in the network. The net class CINSimPlus features a concept to define dynamic reconfiguration steps. The so called *static* reconfiguration flushes a network before reconfiguration. In case of uncompleted drain phase it removes all remaining packets from a network before a reconfigured network will be activated. Therefore no re-addressing of packets according to the new network layout is needed. The type of reconfiguration is selectable via a menu that appears when the reconfiguration arrow in the GUI is selected. Dynamic and static reconfiguration can be selected for every single reconfiguration.

Assuming that the network configuration as shown in [Figure 5.39](#) is given, a reconfiguration step could probably be defined to increase the buffer components' physical buffer space. Following the previous [Section 5.2.6](#) a new network configuration can be created based on an existing one by using the designated button from the menu bar. Descending into the newly created network configuration will bring up the network description shown in [Figure 5.45](#). The configuration holds the same elements, but now the network components are labeled with their complete index string, indicating that they were taken from an exiting configuration. Furthermore, the attribute *cloned* is set to *true*. Changes to the index string of network component will be reflected by all its clones. With this model given, a reconfigured network description can be defined by changing the network layout or the properties of some network components.

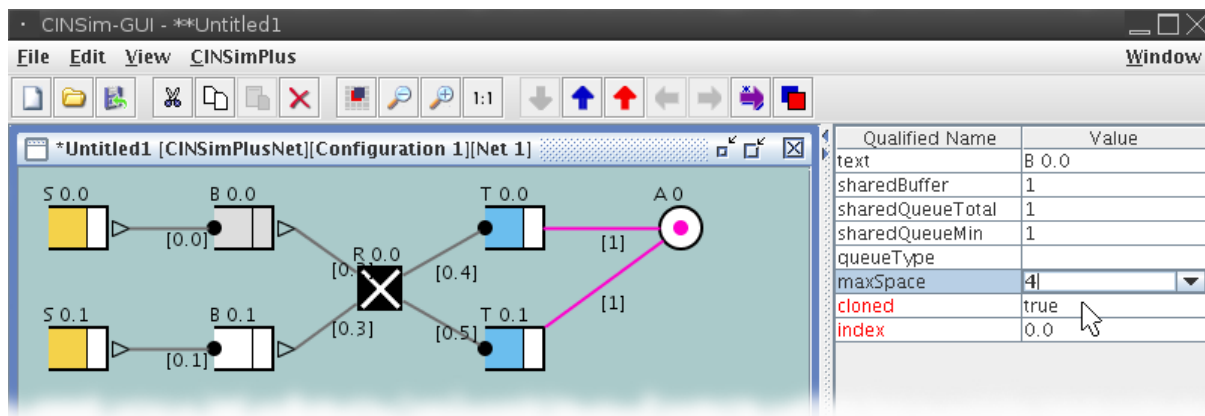


Figure 5.45. Reconfigured network

After ascending to the top level, the old and the new network configuration can be connected by a reconfiguration step available from the insertion tool bar ([Table 5.3](#)). The arrow head in [Figure 5.46](#) reflects the intended direction of the reconfiguration step. A network configuration can only be starting point for one reconfiguration step, but destination of several ones. Therefore, more complex reconfiguration steps as shown [Figure 5.49](#) can also be defined. Selecting a connector brings up its attributes used for setting up the reconfiguration step. The currently supported mechanism is divided into three phases: In the *operation phase* the old network configuration will be used, the *packet drain phase* specifies the time needed for all the packets to leave the network before the *reconfiguration phase* starts. After the reconfiguration phase, the newer network will be used. For these phases the desired number of ticks has to be specified, following the attribute specifications in [Table 5.14](#).

The chosen value for the drain phase only holds for static reconfiguration. In dynamic reconfiguration a user defined drain phase is not needed but a packet collection phase that is calculated by taking the maximum packet size into account. In contrast to static reconfiguration where the drain phase specifies the time needed to flush the network (all packets are forwarded to their targets), the collection phase of a dynamic reconfiguration defines the time needed to finish all active packet transmissions between buffers (no new transmissions are star-

ted). Assuming that the maximum packet size for a configuration is given and only one slice (flit) of a packet can be transmitted from one buffer to another per clock cycle, the time needed for all currently transmitted packets to be completely stored in a receiving buffer is at most *maximum packet size - 1*. The maximum packet size is in CINSim indirectly given by the minimum available buffer space (see [Section 5.2.2](#)), because all switching strategies, except wormhole that is not allowed in conjunction with dynamic reconfiguration, require a packet to be completely storeable at the receiving buffer before a transmission starts (see [Section 7.3](#)). In case of dynamic reconfiguration the length of the collection phase is therefore automatically obtained to assure that all packets are completely stored in a single buffer at the end of that phase. The value for the drain phase is ignored. More specialities of dynamic reconfiguration are summarized in [Section 5.2.8](#).

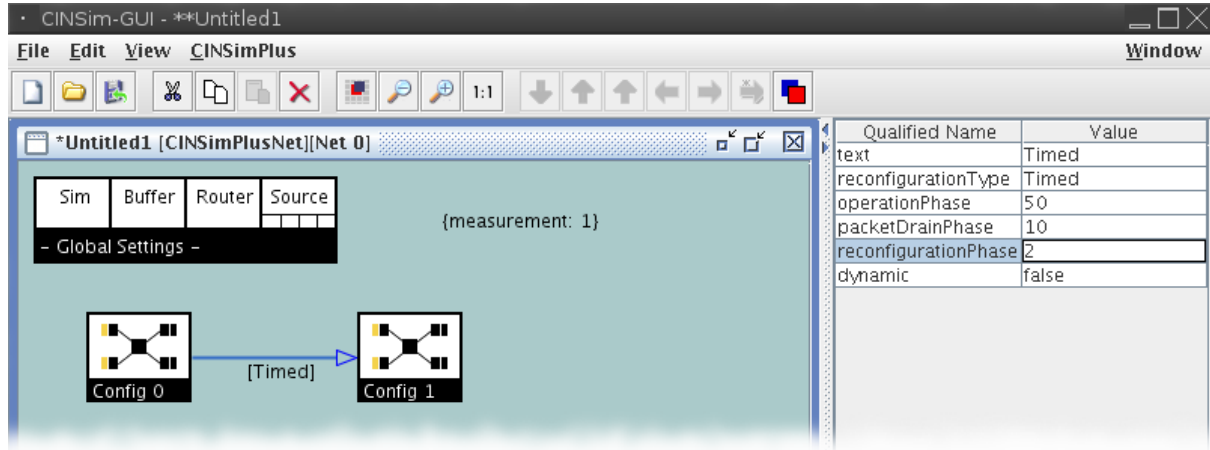


Figure 5.46. Single reconfiguration step

Qualified Name	Value
text	Timed
reconfigurationType	Timed
operationPhase	50
packetDrainPhase	10
reconfigurationPhase	2
dynamic	false

Figure 5.47. Setting for static reconfiguration

Qualified Name	Value
text	Timed
reconfigurationType	Timed
operationPhase	50
packetDrainPhase	10
reconfigurationPhase	2
dynamic	true

Figure 5.48. Setting for dynamic reconfiguration

Attribute	Value description
reconfigType	Timed
operationPhase	[1-9][0-9]*
packetDrainPhase	[0-9]+
reconfigurationPhase	[0-9]+
dynamic	false (default), true

Table 5.14. Valid attribute values for reconfiguration setup

The first configuration in a sequence of reconfiguration steps has to be specified manually by setting its attribute *initialConfig* to *true* as shown in Figure 5.49. If there is already an initial configuration specified then its related attribute will be set to *false* automatically.

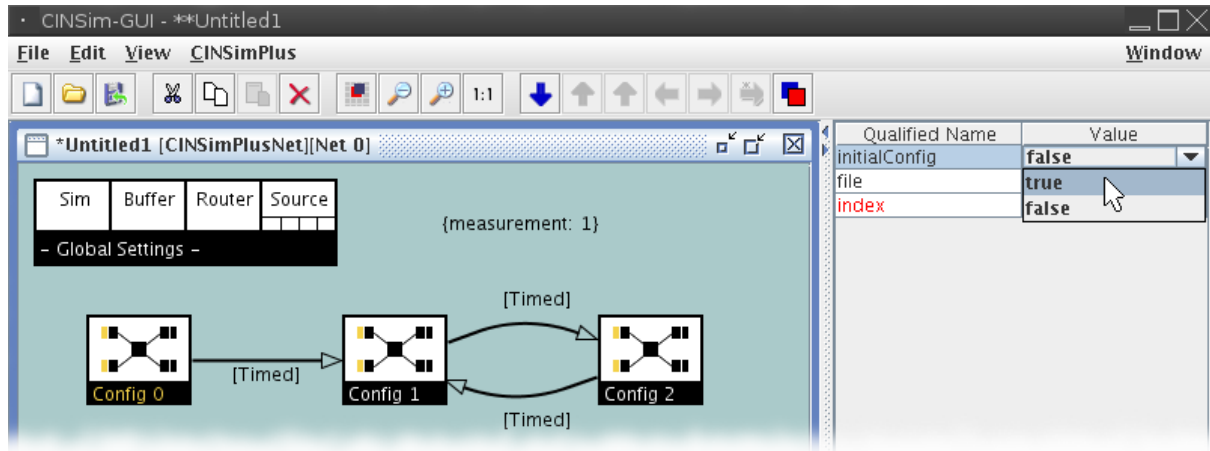


Figure 5.49. Reconfiguration steps with loop

In some cases, one might want to subsequently add network components to an already set up reconfiguration sequence. The net class extension CINSimPlus also supports this feature. In the following a reconfiguration sequence is assumed, starting with the reconfiguration step given in Figure 5.46. The initial configuration is extended by the new target element *TB 2* and the route 6, connecting it to the existing router element as shown in Figure 5.50. Furthermore, the new target element is connected to the analyser object. If the new network components are to be reflected in following network configurations, you can use the navigation button from the tool bar extension to bring them up one by one.

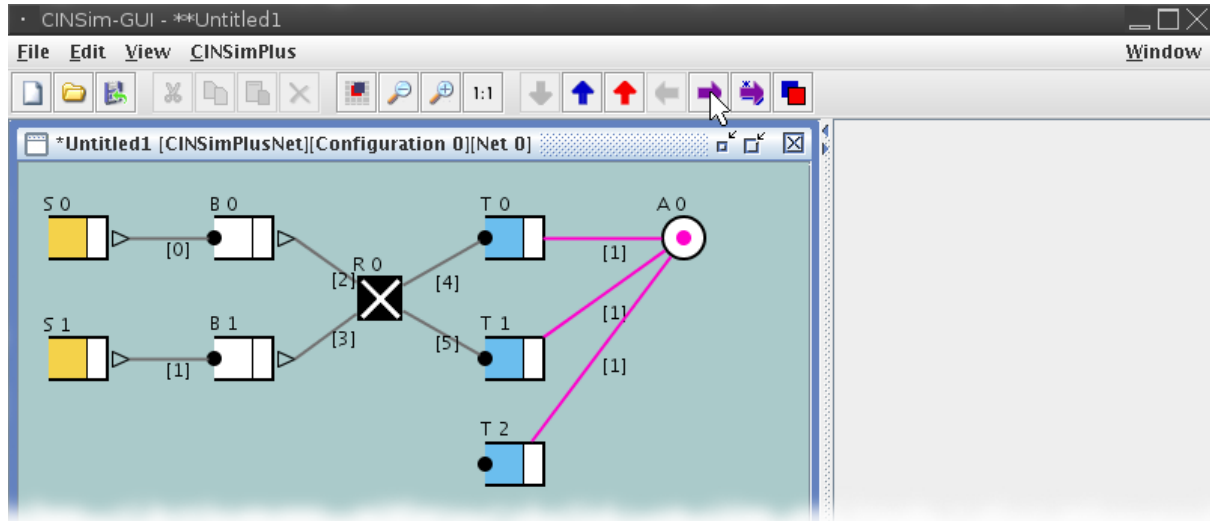


Figure 5.50. Switching to the next network configuration

After the editor frame was moved to one of the next configurations, a new target element has to be added there, too. The value field related to the attribute *index* of any network component turns into a combo box when selected by left-clicking. The combo box holds at least the current index string, but within a reconfiguration sequence, it also holds the indexes of all network components of the same type from previous network configurations, whereas indexes of components that are already part of the current configuration are omitted. For the newly inserted target element in Figure 5.51, the combo box holds its current index string *1.0* and the index string *0.2* of the target element subsequently added to the initial configuration, the possible entries *0.0* and *0.1* are omitted. Selecting the entry *0.2* maps the target element to the one from the initial configuration by setting the attribute *cloned* to *true* and retaining the selected index string. This mapping mechanism works for all unmapped source buffer, buffer, target, router and meta elements, that have no in- or outgoing route. Applied to a

meta element, the nested net element will also be replaced by the one from the chosen meta element. A newly inserted route can be mapped to a route from a previous configuration if this route connects the same network components. Therefore, the route 6 from the initial network configuration can also be added to the configuration in Figure 5.51. For analysers and observations there is no mapping needed, because they are only visual aids for defining measurements and no network components of importance for reconfiguration.

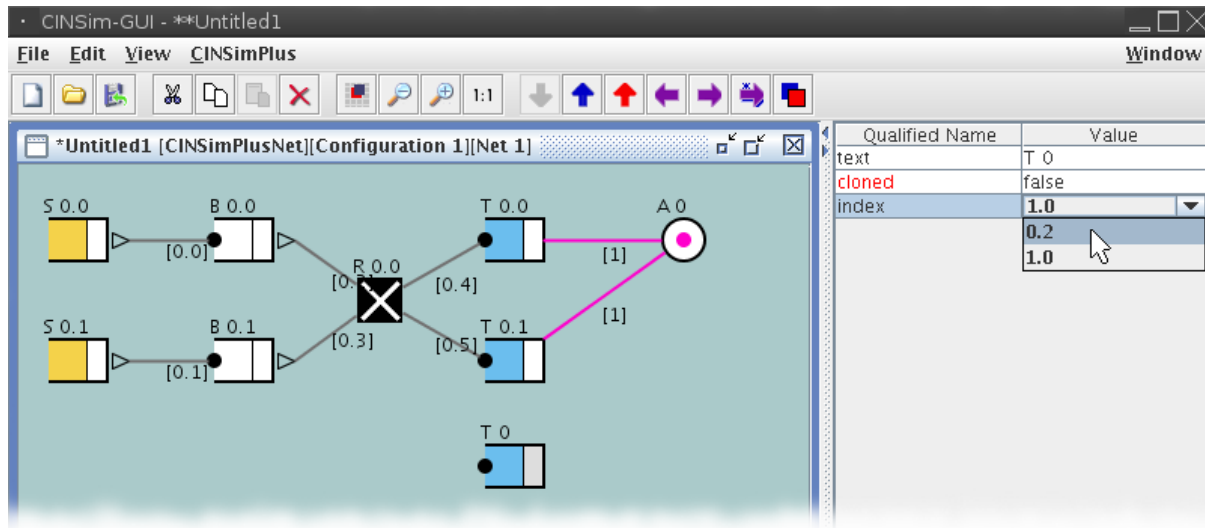


Figure 5.51. Mapping to a component of previous network configuration

5.2.8. Specialities of Dynamic Reconfiguration

It is very important for the correct simulation of dynamic reconfigurations that some rules are observed at building the network in the GUI: *First*, dynamic reconfiguration does only make sense if every target of a network is found as a clone in it's upcoming reconfiguration. Putting new targets that are not clones of the previous configuration is of no effect for packets that have been generated in the previous network run. Packets that do not find their targets any more will be deleted during reconfiguration. *Second*, if a target, although existing as a clone in the new configuration, is not reachable from any buffer that was able to reach the original target in the former configuration, a problem could occur. A packet could be stored in the buffer but not be routed to the aimed target. This packet will be deleted. As a consequence, a builder of dynamic reconfigurations must observe that this case is prevented. *Third*, what holds for targets does hold for buffers as well. Every buffer of the former configuration must have a clone in the new reconfiguration. Otherwise packets are lost because it is unknown in which buffer of the new configuration they belong. Linking two networks by a reconfiguration arrow and simulate it dynamically can not work, because every buffer of the first configuration must have a clone in the second. And this mapping must be meaningful, additionally.

Chapter 6. Simulation Runs with *cinsim*

6.1. The Basics

The CINSim installation packages provide two core applications: The simulation engine *cinsim* and the XML parser *cinparser*. To run simulations you will need the simulation engine. The XML parser is mainly intended for troubleshooting purposes as it allows to parse XML files without actually running a simulation.

The simulation engine provides all functionalities needed to parse a simulation setup created with *cinsim-gui* and to run simulations. A single or more simulation instances are run on an arbitrary number of hosts. The standard scenario, of course, is to run one simulation engine on a single host. The simulation will stop when the desired confidence level and precision is reached. A detailed description on the direct use of the simulation engine is given in [Section 6.2](#).

The engine makes use of a configuration file which is created during the installation process. This file specifies the basic settings and can be modified to force a different behaviour in some aspects. A short introduction to the syntax and the usage of the configuration file can be found in [Section 6.3](#).

6.2. Using *cinsim*

The simulation core *cinsim* handles the entire simulation. The user can benefit from various features that the simulator offers:

- Specify precision and confidence via command line options.
- Vary specific parameters in simulation sequences and check the outcome.
- Run simulations distributed over an arbitrary number of hosts using MPI.
- Use different random number streams for every simulation run.
- Benefit from more comprehensible simulation results.
- Generate plots of simulation results directly or put them in Gnuplot-ready data files.
- Access intermediate results and abort simulation runs at will.

The usage of *cinsim* is as follows: You (can) create a simulation setup with *cinsim-gui* as described in [Chapter 5](#) and just run it using the CINSim core:

```
kuehmi@presence:~/workspace$ cinsim -C 0.99 -P 0.01 gui-trunk/samples/submin4x4.xml
XML-File parsed, starting simulation...

*** Generating textual output for 1 result(s) *****
# Thu Jan 25 11:15:25 2007
# file:///home/kuehmi/gui-trunk/samples/submin4x4.xml
# Steady-state simulation

VAR  RUN  ESTIMATE  DELTA  ERROR  VALUES  CONF  PREC  TYPE  DESCRIPTION
1    1    2.73051    0.0264171  0.00967477  44544    0.99  0.01  Mean  Delay

*** Finished *****
```

The simulation stops automatically when the given confidence level and estimated precision is reached. While simulation runs, you can check the progress of a simulation run and current intermediate results by hitting *ctrl-c* once. In case of long running simulations, this helps to scan intermediate results for reaching weaker termination criteria that you are now willing to accept. If the results are to your liking, you can stop the simulation by twice hitting *ctrl-c*, whereas the current results are treated as the final results, allowing to generate plots and data files as well.

6.2.1. Command Line Options

If you run *cinsim* without any arguments or with the *-h* (*--help*) option, a brief description of the available command line options will be displayed:

```
CINSim v0.6 -- Simulation engine
Usage: cinsim [option]... [xml-file]

Valid options are:
-v,          --verbose          Print verbose messages
                                (only visible when not running in background)
-h,          --help            Print this message and terminate
-t,          --timer           Show execution time
-f FILE,     --config-file=FILE Specify the configuration file
-s FILE,     --schema=FILE     Specify the XML schema file
-n URL,      --namespace=URL   Specify the XML namespace url
-m,          --minimal-validation Only minimal validation (syntax checking)
-c,          --schema-validation Validation against schema (default)
-e,          --extended-validation Validation against schema and constraints
-i LEVEL,    --nice-level=LEVEL Specify the nice level; 0-19 (user), <0 (superuser)
-d FILE,     --data-file=FILE  Write results to FILE
-p FILE,     --plot-file=FILE  Create a plot (postscript) with gnuplot
-l,          --plugin-list     Show all plugins and terminate
-R INTEGER,  --repetitions=INTEGER The number of repetitions for every simulation run
-P DOUBLE,   --precision=DOUBLE Set the precision for simulation results
-C DOUBLE,   --confidence=DOUBLE Set the confidence level for simulation results
-G ID,       --random-generator=ID Select the generator for random numbers
                                (Available are CMRG (default) and MTwister)
```

```
-S INTEGER, --random-stream=INTEGER Specify the random number stream to be used
                                   (Otherwise, a stream will be randomly selected)
```

These information should again be self-explanatory but some more information will be given in the following. Depending on your environment, long options are available or not. If your system follows the POSIX conventions you will see the output above, otherwise the long options (starting with `--`) are omitted.

-v, --verbose	Verbose messages will be printed. For example, the entries of the process dictionary give information on how the engine was set up by parsing command line options and the configuration file.
-h, --help	Enabling this option prints a list of available command line options on your standard output and stops the engine.
-t, --timer	Enabling this option prints the execution time and the used processor time on your standard output.
-f, --config-file	Use this option if you want to use an alternative configuration file instead of the installed one. Section 6.3 gives information on how to write a configuration file.
-s, --schema	Sets the XML schema file to be used for XML validation. This option overwrites the directive of the configuration file.
-n, --namespace	Sets the XML schema namespace. The namespace is required, as it describes the context of an XML schema. The namespaces of the XML schema and an XML file to be validated must be equal, otherwise the schema will not be used for validation. This option overwrites the directive of the configuration file.
-m, --minimal-validation	If this option is enabled, only syntax and well-formed checking will be applied when parsing an XML simulation setup. This option overwrites previous <code>-c</code> and <code>-e</code> options.



Note

It is usually a good idea to do the full validation, as it checks for valid document structure and attribute values as well. Setting up a simulation relies on a valid XML description and skipping the tests may lead to unexpected behaviour. This option is used by the simulation sequencer as it usually checks the document and there is no need for the simulation engine(s) to repeat the validation.

-c, --schema-validation	Validate the XML document against the XML schema specified by the configuration file or command line options (default). This option overwrites previous <code>-m</code> and <code>-e</code> options.
-e, --extended-validation	This option enables progressive parsing, allowing advanced constraints to be checked while parsing the XML file. Currently, there are no constraints defined and therefore this option has the same effect as using the <code>-c</code> option. This option overwrites previous <code>-m</code> and <code>-c</code> options.
-i, --nice-level	With this option you can set the nice level of the simulation engine. The user nice level is a value between 0 and 20, whereas the lower the value the higher the priority. Super users may use nice levels down to -19.
-d, --data-file	If you want the simulation results to be written to file for post-processing, you can use this option.
-p, --plot-file	This option instructs the simulation sequencer to generate plots for every

single measurement (Section 5.2.5) using Gnuplot. You should have installed Gnuplot to make use of this feature. Plots can be generated for simulation sequences and terminating simulations with observations for more than one clock cycle. Otherwise, the output will be skipped.

-l, --plugin-list	Prints the list of available plugins to the simulation engine on screen.
-R, --repetition	Sometimes a simulation will stop pretending the required confidence and precision has been reached, although the number of observation seems to be insufficient. If you think this happens, you should use this option to repeat the simulation R times with different random number streams. The results for each variable will be checked and the one out of R values is used for which the most observations have been used will be accepted.
-P, --precision	The precision to be used for output data analysis can be set with this option. The default is specified in the configuration file.
-C, --confidence	This option sets the confidence level for output data analysis. The default is specified in the configuration file.
-G, --random-generator	This option chooses the random number generator. Currently, two different generators are available. The first one is a combined multiple recursive generator (CMRG) with a cycle of 2^{127} . The second one is based on twisted generalized feedback shift registers with a cycle of 2^{521} . It is called <i>Mersenne Twister</i> .
-S, --random-stream	This option sets the initial random stream identifier. A repetition of the simulation with the same sequence of random numbers becomes possible by using the same initial random stream identifier. If this option is omitted, the identifier is set by random.

6.2.2. Simulation Sequences

The idea of running simulation sequences is quite simple: Instead of repeating simulation runs several times with some modified settings, you can use CINSim parameters (Section 5.2.4) to do these modifications for you. CINSim will treat a simulation setup as a sequence, if the attribute *numberOfSimulations* is set to a value greater than one (Section 5.2.2). The advantage of this approach is that the delivered results of a simulation sequence already reflect the variations of the observed performance characteristics due to the modified settings without putting them together from miscellaneous simulation runs. Each run will be started with a new stream of random numbers.

```
kuehmi@presence:~/workspace$ cinsim -C 0.99 -P 0.01 gui-trunk/samples/subbmin8x8.xml
XML-File parsed, starting simulation...

*** Generating textual output for 4 result(s) *****
# Thu Jan 25 15:04:25 2007
# file:///home/kuehmi/gui-trunk/samples/subbmin8x8.xml
# Steady-state simulation

# Run: 1-4


| VAR | RUN | ESTIMATE | DELTA     | ERROR      | VALUES | CONF | PREC | TYPE | DESCRIPTION |
|-----|-----|----------|-----------|------------|--------|------|------|------|-------------|
| 1   | 1   | 5.92734  | 0.0584292 | 0.00985757 | 86528  | 0.99 | 0.01 | Mean | Delay       |
| 1   | 2   | 10.2877  | 0.102595  | 0.00997251 | 118784 | 0.99 | 0.01 | Mean | Delay       |
| 1   | 3   | 15.002   | 0.142513  | 0.00949962 | 204800 | 0.99 | 0.01 | Mean | Delay       |
| 1   | 4   | 19.6511  | 0.192204  | 0.00978083 | 114688 | 0.99 | 0.01 | Mean | Delay       |


*** Finished *****
```

The output is ordered by the measurement variables, the run and, in case of a terminating simulation, by the clock cycle (tick). In the latter case, a summary table for every single run of the sequence will be created. The output above shows the results for a sequence of four runs, investigating the variation of the packet delay as a result of using larger packets (and buffers).

6.2.3. Distributed Simulation

To run simulations distributed or in parallel, MPI support has to be enabled during configuration and compilation:

```
./configure [...] --enable-mrip
```

If there isn't an appropriate MPI library present, configuration will fail.

In the following, parallel simulation on a multicore computer is presented (e.g. MPICH Device: `ch_shmem`). To set up MPI on a cluster parallel computer, please consult the documentation of your MPI implementation (see [Section B.2](#)).

Now, to start a simulation on your two cores, invoke:

```
dluedtke@hornisse:~/cinsim$ mpirun -np 2 /usr/local/bin/cinsim submin4x4.xml
MPI-Mode: Use 'kill -USR2 29029' for intermediate results, ctrl-c for aborting XML-File parsed,
starting simulation...
*** Generating textual output for 1 result(s) *****
# Thu Feb 1 18:40:56 2007
# file:///home/dluedtke/cinsim/gui/samples/submin4x4.xml
# Steady-state simulation
VAR RUN ESTIMATE DELTA ERROR VALUES CONF PREC TYPE DESCRIPTION
1 1 2.72852 0.13422 0.0491914 512 0.95 0.05 Mean Delay
*** Finished *****
```

mpirun starts MPI jobs for the configured device (in this case the shared memory device). The parameter *-np* specifies the number of processors to run on. If you don't want a parallel simulation, you can just invoke *cinsim* without the *mpirun*-part.

6.2.4. Intermediate Results

As already mentioned, the simulation stops automatically when the given confidence level and estimated precision is reached. While simulation runs, you can check the progress of a simulation run and current intermediate results by hitting *ctrl-c* once. In case of long running simulations, this helps to scan intermediate results for reaching weaker termination criteria that you are now willing to accept. If the results are to your liking, you can stop the simulation by twice hitting *ctrl-c*, whereas the current results are treated as the final results, allowing to generate plots and data files as well.

In case of simulation sequences, *ctrl-c* lists the final results of already finished simulation runs (where the termination criteria have already been met) and the intermediate results of the current simulation run.

```
kuehmi@presence:~/workspace$ cinsim -C 0.99 -P 0.01 gui-trunk/samples/subbmin8x8.xml
XML-File parsed, starting simulation...
*** Generating textual output for 2 result(s) *****
# Thu Jan 25 15:13:09 2007
# file:///home/kuehmi/gui-trunk/samples/subbmin8x8.xml
# Steady-state simulation
# Run: 1-2
VAR RUN ESTIMATE DELTA ERROR VALUES CONF PREC TYPE DESCRIPTION
1 1 5.90007 0.0508018 0.00861037 102400 0.99 0.01 Mean Delay
1 2 10.3139 0.591582 0.057358 6786 0.99 0.01 Mean Delay
*** Finished *****
```

Twice hitting *ctrl-c* stops the entire simulation sequence with these values.

If you are running a simulation with MPI, don't use *ctrl-c* for intermediate results. Some MPI implementation will abort the computation immediately if they receive *ctrl-c*. Instead, invoke the command, that *cinsim* gives you during startup:

```
dluedtke@hornisse:~/cinsim$ mpirun -np 2 cinsim submin4x4.xml
MPI-Mode: Use 'kill -USR2 29029' for intermediate results, ctrl-c for aborting XML-File parsed,
starting simulation...
```

6.2.5. Gnuplot Support

The best way to understand or present simulation results is to create plots. A popular tool for generating high-

quality plots is Gnuplot, which is supported by CINSim in two ways: You can directly create plots from simulation results by using the command line option *-p* (*--plot-file*) or you can put the results in a Gnuplot-ready data file by using the *-d* (*--data-file*) option. In both cases, you need to specify the file name for files to be created.

The direct generation of plots works for simulation sequences and terminating simulations, observing performance characteristics for more than one clock cycle. In the first case the changes over the performed simulation runs for each measurement will be plotted and in the latter case the changes according to the considered clock cycles. If a sequence of terminating simulations is defined, a plot for every run will be generated.

If you want to create your own plots from the results you can put the results in a file and use the various Gnuplot options to generate the plots. See the Gnuplot manual for details.

6.3. Configuration

Most of the command line options presented in [Section 6.2](#) overwrite defaults specified in the configuration file *config.xml* located in *PREFIX/share/cinsim*, whereas *PREFIX* specifies the installation directory of the core applications ([Section 4.3](#)). You can use a copy of this file to set frequently used command line options for your simulation runs and pass it to the simulation engine or the simulation sequencer via the *-f(--config-file)* option.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmlconfig xmlns="http://www.pdv.cs.tu-berlin.de/CINSim/0.6/schemas/XMLConfig"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.pdv.cs.tu-berlin.de/CINSim/0.6/schemas/XMLConfig
    /home/kuehmi/workspace/core-trunk/share/cinsim/xmlconfig.xsd">
  <parser exitOnFirstError="1" processing="schema" ❶
    namespace="http://www.pdv.cs.tu-berlin.de/CINSim/0.6/schemas/CINSimPlus"
    schema="/home/kuehmi/workspace/gui-trunk/etc/schemas/CINSimPlus.xsd"/> ❷
  <engine binary="cinsim" niceLevel="5" number="1"> ❸
    <prng factoryId="CMRG"/> ❹
  </engine>
  <simulation confidence="0.99" precision="0.01" repetitions="1"/> ❺
  <output progress="1" verbose="0"> ❻
    <data file="result.dat"/>
    <plot file="result.ps"/> ❼
  </output>
</xmlconfig>
```

Example 6.1. Customized configuration file

- ❶ These two attributes set the way the XML documents are validated. *processing="schema"* enables XML schema validation and *exitOnFirstError="1"* stops the parser after the first error occurred, omitting further errors. Alternatively, all errors can be displayed by setting the second attribute value to *0* and the processing can be set to *minimal*, doing only some syntax and well-formed checking, or to *extended*, which is currently equal to *schema*.
- ❷ The attributes *schema* and *namespace* of the element *parser* are used to determine the XML schema for validation. Usually, these attributes are set during the installation process and should be left unchanged.
- ❸ The element *engine* specifies some properties for the simulation engines. *binary* specifies the name of the binary that is used to start simulation engines, and *niceLevel* sets the nice level to be used. *number* specified in former versions of CINSim the number of engines to be started in case of a distributed simulation. Now, MPI is used for this purpose and this parameter is obsolete.
- ❹ The subelement *prng* specifies the random number generator to be used.
- ❺ This element sets the simulation parameters when using the simulation sequencer. You can set the precision, the confidence and the number of repetitions. When using more than one repetition, each single run will be repeated several times, according to the number specified, and the results with the most observations will be returned.
- ❻ The attributes *progress* and *verbose* of the element *output* specify whether progress (simulation sequencer) and verbose information should be printed (1) or not (0).
- ❼ The optional subelements *data* and *plot* of the element *output* can be used to enable the output to a data file or the generation of plots using Gnuplot.

Part III. Specifications

Table of Contents

7. Simulation Components	70
7.1. The Coroutine	70
7.1.1. Local Backpressure	70
7.1.2. Global Backpressure	71
7.1.3. Alternatives	71
7.2. Routing Strategies	72
7.2.1. Bitmask Routing	72
7.2.2. Shortest-Path Routing	72
7.2.3. XY Routing	73
7.2.4. West-First Routing	73
7.3. Switching Techniques	74
7.3.1. Store-And-Forward Switching	74
7.3.2. Virtual Cut-Through Switching	75
7.3.3. Partial Cut-Through Switching	75
7.3.4. Wormhole Switching	75
7.4. Steady-State vs. Terminating Simulations	76
8. Network Components	77
8.1. Source Buffer Component	77
8.1.1. Constant-Size Packets	77
8.1.2. Distribution in Time	77
8.1.3. Cumulative Distribution Functions	79
8.2. Buffer Component	82
8.2.1. Queue Types	82
8.3. Target Buffer Component	83
8.4. Router Component	84
8.4.1. Arbiter	84
8.4.2. Arbiter Controller	88
8.5. Analyser Component	89
8.5.1. Observation Types	89
8.5.2. Measurement Statistics Types	90
9. Reconfiguration	91

Chapter 7. Simulation Components

7.1. The Coroutine

The simulation engine of CINSim parses a given simulation set and creates an abstract graph representation of the interconnection network to be simulated. Furthermore, the simulation components are set up according to the entry *Sim* of the local/global settings menu introduced in [Section 5.2.2](#). Currently, there are three major components that can be combined in many ways: The *coroutine*, the *routing* and the *switching strategy*. In cooperation these components determine how, when and where packets within the interconnection network are moved. To achieve synchronously working network components, the network is internally clocked. In every clock cycle the coroutine is activated and initiates the packet forwarding by triggering the network components. Routing and switching strategy, described in detail in [Section 7.2](#) and [Section 7.3](#), are nested within the coroutine as independent components and applied on demand.

Packets to be passed through an interconnection network are divided into packet fragments of fixed size, called flits (flow control units). The number of flits used for a packet defines its size. The first flit is called *head*, the last flit is called *tail*. A connecting route between network components can be passed by one flit in every clock cycle. Router components connect their ingoing routes to outgoing routes according to the destination of packets, determined by their head. A connection is maintained until the tail has passed. Within every single clock cycle the coroutine has to make sure, that all movable packet fragments are moved, according to the buffer components' queues, the connections built up by the router components and the restrictions given by the switching strategy. For this purpose, the components are activated to send and request packets. According to the resulting states, the packet fragments are received, remain at their current buffer queue or may even be removed. Varying the order of the components' activation and different handling of the resulting states leads to coroutines with a specific behaviour. Currently, two implementations of the coroutine interface are available, one for *global* and one for *local backpressure*. They can be used by setting the attribute *forwardingType* ([Table 7.1](#)) of the section *Sim*, to be found in the local/global settings menu. It is worth mentioning, that each network configuration in a reconfiguration sequence can have a different coroutine.

Value	Short description
LocalBackpressure	Simulation run with coroutine for local backpressure.
GlobalBackpressure	Simulation run with coroutine for global backpressure (default).

Table 7.1. Value descriptions for attribute *forwardingType*

7.1.1. Local Backpressure

The local backpressure mechanism applied to interconnections networks observes only the state of the following buffer component for a packet fragment to be sent. If sufficient buffer space is available, the flit will be sent, otherwise it will remain at its current position. That means that no packet fragments, and therefore no packets, are lost. However, the packet flow of the current clock cycle will be ignored. So flits that actually could have been sent, because the next buffer sent its first flit to its follower within the same clock cycle, will stay at their current positions. This may also lead to an asynchronous behaviour in simulation, because some following buffer components might have already sent their leading flit in the current clock cycle, while others have not. To prevent from this kind of situation, the buffer components' states of the last clock cycle will be used for analysis by making all decisions first and then moving the packet fragments to be sent to their next buffer components within the network.

Using local backpressure in conjunction with wormhole switching leads to a mentionable behaviour: A worm packet is usually distributed over several buffer components storing a single flit and one might assume that moving the head flit automatically leads to the movement of the remaining flits of the worm. This would, however, require a specific implementation of the coroutine for wormhole switching which conflicts with the goals of CINSim to make use of independent components. Therefore, worms are treated like any other packets

leading to appearing and disappearing bubbles (reserved but empty buffer components). The bubbles can be avoided using global backpressure.

7.1.2. Global Backpressure

The global backpressure mechanism invokes, in contrast to local backpressure, additional information concerning the packet flow of the current clock cycle. With these information, flits can be sent to the next buffer component, although the available buffer space was insufficient at the beginning of the clock cycle. Newly available buffer space will be used on the spot, if a flit can be received. This behaviour is achieved by activating the network components beginning with the target buffers and successively moving the flits to buffer components with free or freed resources. Figure 7.1 gives an example of this mechanism: Targets T1 and T2 are activated. Green arrows depict transmissions. The labels describe orientation and order of the transmission. So buffers a and c are both oriented to T2, c transmitting first and a second. Buffer b is oriented towards T1.

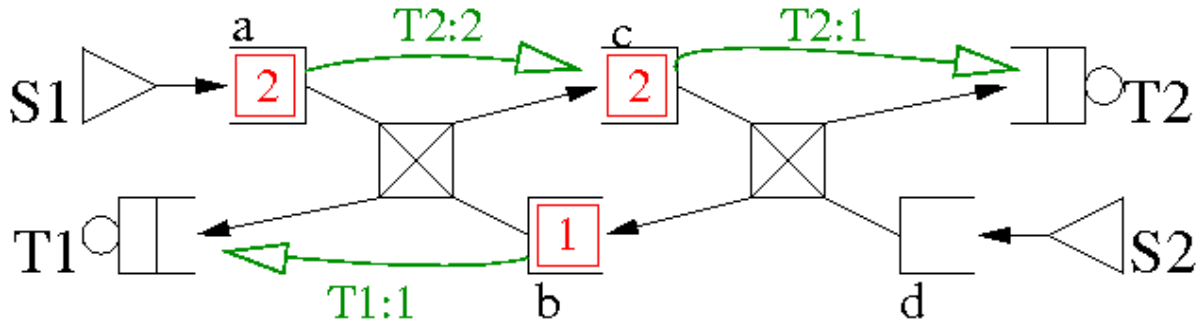


Figure 7.1. Global backpressure routing oriented on targets

After a dynamic reconfiguration, it is not always possible to move every flit activating targets only. In the analyzed and solved deadlock-case, *buffers* must be chosen according to a distinct dependency analyzing algorithm that are activated additionally to all targets. Figure 7.2 shows the same example as Figure 7.1 after reconfiguration. Targets were swapped. Now a is oriented towards T2 and c to buffer b. After c scheduled its transmission, b is scheduled as well, being oriented the same way as c. b is then oriented to itself, perplexingly. But there is no other way to do global backpressure, because b and c are both dependent on each other.

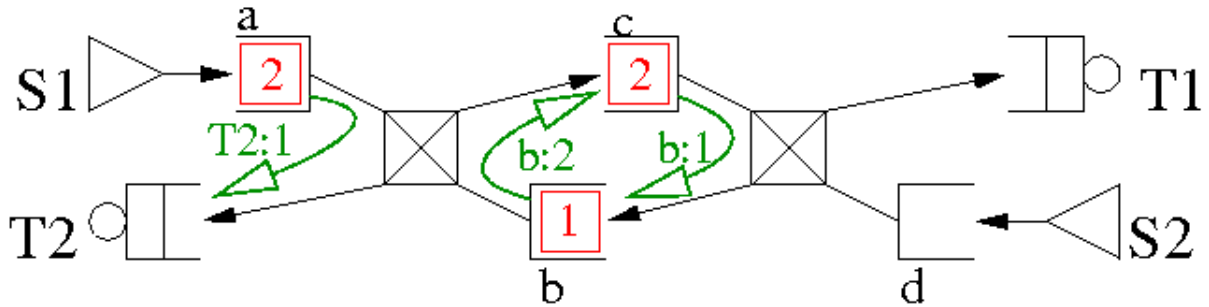


Figure 7.2. Global backpressure routing, deadlock case, oriented on targets and buffers

7.1.3. Alternatives

Backpressure mechanisms keep packets at their current positions until the next buffer component becomes available again. Therefore, no packets are lost. Other ways to handle insufficient buffer resources could be to drop or re-route packets. As packet dropping is under investigation in combination with network reconfiguration and packet deadlines, an implementation of the coroutine that drops packets in case of unavailable buffer components might be a future feature of CINSim.

7.2. Routing Strategies

The layout of an interconnection network sets up one or, in case of redundancy, more paths between a given source and a set of destinations. A routing strategy determines for a given message the paths to its destination. In case of multicast, the paths to each single destination are determined. CINSim only knows unidirectional connections, so a complete path starts at a source buffer and ends up at a target buffer. Messages are currently modelled by constant size packets. A source buffer sets up the targets and passes the created packet to the network. The currently available routing strategies are *distributed*, all branches are modelled by router components which determine the output port to be taken according to the information provided by a packet's head. The routing itself may allow more than one output if the network holds redundant paths or a multicast is performed. Furthermore, all strategies are *deterministic*, only the current position and the destinations of a packet are considered, the current state of the network has no influence on the routing decision.

The abstract graph representation of the network is traversed at startup to set up *bitmasks* for every network component, determining the reachable destinations. The size of every generated bitmask depends on the number of reachable target buffers. If n target buffers are reachable, bitmasks of size $\log n$ will be created. These bitmasks reflect the physical structure of the network. Based on this structure, the routing strategies provided by CINSim set up a layer restricting the set of physically available paths to a subset of valid paths according to the routing logic to be applied. The head of every packet holds a bitmask defining its destinations. A router component compares this bitmask with the restricted bitmasks related to its output ports to determine what ports might be taken. The routing strategy itself does not prefer any of the valid paths, the final selection is done by the router component. More information to the selection process can be found in [Section 8.4](#).

The routing strategy to be used can be set for each network configuration in the first section of the local/global settings menu ([Section 5.2.2](#)) by choosing the related value for the attribute *routingType*. [Table 7.2](#) gives an overview with short descriptions. Upcoming sections discuss appliance for static and dynamic reconfiguration.

Value	Short description
Bitmask	Bitmask routing according to bitmasks determined by the layout of the network (default).
ShortestPath	Shortest-path routing restricting the valid paths to shortest paths only.
StandardXY	XY routing for two-dimensional meshes.
WestFirst	West-first routing for two-dimensional meshes.

Table 7.2. Value descriptions for attribute *routingType*

7.2.1. Bitmask Routing

The standard routing strategy of CINSim is *bitmask routing*. The bitmasks of the network components are used without any restrictions. Bitmask routing can be used for many regular network topologies like Clos networks or MINs. If the topology contains cycles, the use of this strategy leads to deadlock situations. CINSim supports bitmask routing for both static and dynamic reconfiguration.

7.2.2. Shortest-Path Routing

The *shortest-path routing* strategy restricts for any source-destination pair the possible paths given by the network's layout to a subset of paths with minimal length. The length of a path is determined by the number of buffer components that must be passed by a packet, including the final buffer component. This strategy can be applied to topologies with redundancy and even cycles, for example to BMINs. CINSim supports it for both static and dynamic reconfiguration. However, when applied to meshes, shortest-path routing leads to deadlock situations that can not even be solved by the reconfiguration-proof routing of dynamic reconfiguration.

7.2.3. XY Routing

The *xy routing* strategy can be applied to regular two-dimensional mesh topologies without obstacles. A mesh topology must be specified using a mesh configuration introduced in [Section 5.2.1](#). The position of the mesh nodes and their nested network components will be described by coordinates, the x-coordinate for the horizontal and the y-coordinate for the vertical position. A packet will be routed to the correct horizontal position first and then in vertical direction. XY routing produces minimal paths without redundancy, assuming that the network description of a mesh node does not define redundancy. CINSim does *not* support xy routing for dynamic reconfiguration.

7.2.4. West-First Routing

The *west-first routing* is another strategy for regular two-dimensional mesh topologies. In contrast to the xy routing, this strategy does not guarantee minimal paths and produces redundancy. The movement of a packet from one buffer component to the next one is characterized by cardinal points. Assuming that the upper left mesh node has the smallest coordinates, an increasing x-coordinate defines a movement to the east, while an increasing of the y-coordinate defines a movement to the south. West-first routing restricts the valid direction changes: Turnarounds and changes to the west are forbidden. If a target lies westwards, a packet will therefore first be routed to the west to reach the desired horizontal position, similar to the xy routing, but it can be also be routed more westwards than needed. If a target lies eastwards, a packet can be routed to the east, north or south until the final horizontal position is reached. Then a direct path to the target must be taken. This behaviour is adaptive, because unavailable resources can be compassed, but the implementation of the routing strategy itself is not. The behaviour results from the redundant paths determined by the routing logic and the nondeterministic handling of redundancy by CINSim. Even if a minimal path exists and all resources are available, a packet might use a non-minimal path. CINSim does *not* support west-first routing for dynamic reconfiguration.

7.3. Switching Techniques

Apart from determining valid paths between sources and destinations within a interconnection network, a switching technique is needed that specifies how messages are to be fragmented before passing them to the network and how the resources along the path are to be allocated. Furthermore, a switching technique gives preconditions to be fulfilled before a fragment can be moved on to the next network component. Currently, CINSim features several *packet switching* techniques. Messages are modelled by a single packet consisting of constant size flow control units (flits). Within every single clock cycle an allocated connection between two components can only be used to transmit one flit. Before the head flit of a packet can be transmitted, the connection must be allocated. A router component has to create a connection between an input port, ready to send a packet, and a valid output port. After setting up this connection, the output port is unavailable for other input ports and therefore allocated for the packet to be transmitted. The allocated connection will be freed after the tail flit of the packet has passed. Input ports are implicitly allocated, because packets can not pass one another along a single path or, more precisely, within the related buffer queues.

The described behaviour above is the same for all implemented packet switching techniques, only the prerequisites differ. *Wormhole switching* demands only space for one flit to be available at the next buffer component, whereas the remaining techniques require space for the whole packet, checked before the head of a packet is forwarded. Therefore, wormhole switching can cope with buffers of size one independent from the size of the worm packet while other techniques assume that the physical size of all buffer components of the network has a lower bound greater or equal to the size of the packets to be transmitted. To determine whether the whole packet can be completely stored in the next buffer component, the estimated space to be available for the next clock-cycles is used. If a head flit leaves a buffer queue, the assumption can be made, due to the allocation of the connection to the next buffer and the fulfilled prerequisites, that in each of the following clock-cycles another flit of the packet leaves the buffer queue, too. Thus, the estimated space includes the space currently allocated by the remaining flits of a fragmentarily sent packet. This leads to an optimized utilisation of the buffer queues and shorter delay times.

The attribute *switchingType*, to be found in the local/global settings menu ([Section 5.2.2](#)), sets the packet switching to be used for each or for all network configurations. An overview is given by [Table 7.3](#). Future versions of CINSim may also provide *circuit switching* techniques and a more complex message model.

Keep in mind that every switching technique is capable to statically reconfigure a network. Dynamic reconfiguration, being more complex, causes problems in two of the listed switching schemes and are therefore not supported in CINSim. Ideas to support VirtualCutThrough exist and may be provided in a future release of CINSim.

Value	Short description
StoreAndForward	Store-and-forward switching (default).
Wormhole	Wormhole switching.
VirtualCutThrough	Virtual cut-through switching.
PartialCutThrough	Partial cut-through switching.

Table 7.3. Value descriptions for attribute *switchingType*

7.3.1. Store-And-Forward Switching

The default switching technique of CINSim is *store-and-forward switching*. The name already implies its behaviour: Every buffer component along the determined path of a packet (message) from its source to its destination has to store the complete packet first, before it can be forwarded to the next buffer. Therefore, never more than one single connection between two buffer components along the path of a packet needs to be allocated. Remember, even for store-and-forward switching packets are always divided into flits and only one flit will be

transmitted per clock cycle. Store-and-forward switching is allowed for dynamic reconfiguration, where buffer-overflows may occur rarely, but will not harm the simulation results.

7.3.2. Virtual Cut-Through Switching

The *virtual cut-through switching* technique allows packets to be forwarded although they are not completely stored in the current buffer. A packet can therefore be stored in more than two buffer components at the same time and more than one connection might be allocated. However, if the head of a packet can not be forwarded, due to insufficient or unavailable resources, the whole packet has to be stored in the buffer currently holding the head before it can be forwarded again. This results in a degeneration to store-and-forward behaviour in case of heavy traffic. Virtual cut-through switching is prohibited for dynamic reconfiguration, because due to network situations that cause complicated switching behaviour for this switching scheme, CINSim is not yet disposed to support it.

7.3.3. Partial Cut-Through Switching

In contrast to virtual cut-through switching, the use of *partial cut-through switching* lifts the restriction to store the complete packet in case of insufficient or unavailable resources. If the resource becomes available again, the packet can be forwarded even if the tail is not yet received. If the head of a packet is blocked for several clock cycles, the number of allocated connections will usually be decreased, because the remaining flits can always be forwarded until the complete packet is stored in one buffer. This behaviour is the same for virtual cut-through and store-and-forward. Partial Cut-Through is supported for dynamic reconfiguration. Again, cases of buffer-overflow may occur, but will not harm the simulation results.

7.3.4. Wormhole Switching

The only prerequisite for the forwarding of a single flit specified by the *wormhole switching* technique is the actual availability of buffer space for one flit. This must be checked for every flit because the whole packet can probably not be stored in one buffer. Furthermore, allocated connections can usually not be freed unless the head of the worm can be forwarded. This may lead to more deadlock situations than for other implemented strategies. Dynamic reconfiguration is prohibited for wormhole switching out of profound conceptual reasons.

7.4. Steady-State vs. Terminating Simulations

Coroutine, routing and switching strategy of a simulation control the packet movements for every single clock cycle of a simulation run. What clock cycles are actually considered in a simulation run is determined by the simulation type. CINSim provides two different types of simulation runs. A *steady-state* simulation can be used to determine the performance characteristics for the steady state of an interconnection network. The simulation invokes consecutive clock cycles until the results of all measurements reach steady states. In contrast to this approach, the transient behaviour of an interconnection network within a given range of clock cycles can be investigated using a *terminating* simulation. A terminating simulation run will be stopped and restarted after the last clock cycle of the desired range has been simulated. For every single clock cycle within the specified range, a result for all set up measurements will be determined. Both, steady-state and terminating simulation, are observed by CINSim and stopped, if all performance characteristics to be investigated reach steady states according to a given confidence level and a desired precision. The specification of these parameters is described in detail in [Section 6.1](#).

The simulation type can be set up by using the attribute *simulationType* related to the section *Sim* of the global settings menu ([Section 5.2.2](#)). The possible values are given in [Table 7.4](#). If a terminating simulation is to be set up, the first clock cycle and the number of consecutive clock cycles of the analysis needs to be specified using the attributes *analysisStart* and *analysisLength* to be found in the same attribute table. This results in a multiplication of the defined measurement variables ([Section 5.2.5](#)), as results for each clock cycle of the analysis will be calculated.

It is important to be aware that steady state simulation does not make sense for reconfiguring networks. CINSim is not able to decide which network's steady state it should calculate.

Value	Short description
Steady	Steady-state simulation run (default).
Transient	Terminating simulation run.

Table 7.4. Value descriptions for attribute *simulationType*

Chapter 8. Network Components

8.1. Source Buffer Component

A source buffer does not receive packets but creates them on demand invoking various distribution functions. In some aspects a source buffer can be considered as the counterpart of a client connected to a network, like a single CPU or computer. A client usually transfers a message to a network addressed to a specific client or to a set of clients also connected to the network. However, a source buffer component in CINSim can not receive messages, defined by constant-size packets, so you will need an additional target buffer component to simulate a client sending and receiving messages.

The generation of packets is done in two steps: The first step is to check, whether a new packet is to be generated or not, the second step is to set up the packet and to place it in the buffer queue for transmission. Source buffers, and all other buffer-like components, are activated one time within each clock cycle, checking whether they have packets, or strictly speaking a flit of a packet, to be forwarded. The distribution in time used by source buffers then determines, whether a new packet is to be generated. If there is still a fragmentarily transmitted packet in the buffer queue, however, no new packet is generated. Packets that do not enter the network in the same clock cycle they are created will be dropped.

After creating a new packet the source buffer has to set its properties. The first thing to do is, of course, to set the destination or, in case of multicast, the destinations of the packet. This process starts by determining the number of destinations, which is accessed via a *multicast distribution*. With the number of destinations known, the actual destinations are determined using a *spatial distribution* and are set in the bitmask of the packet. Finally, the packet gets a priority, determined using a *priority distribution* and considered by the optional priority scheduling of a router component, a deadline, considered by the optional deadline scheduling, and the current clock cycle is set to calculate latency and delay times.

8.1.1. Constant-Size Packets

The packets that are created by source buffers are flit-based and of constant size, specified by the attribute *packetSize* of a source buffer component or the related entry in the local/global settings menu. The size of a packet specifies the number of flits associated with it. Each flit will require exactly one place within a buffer queue. The first flit of a packet, called *head*, will enter the network if possible and will be received by the next buffer component(s) according to its destination(s). Apart from reconfiguration, the simulation will not drop a packet if at least the head enters the network. Currently, only packets of fixed size can be created, packets of varying size are not supported yet.

8.1.2. Distribution in Time

Source buffers use various distributions to create and set up packets. Setting up the distribution in time, used to determine whether a new packets is to be created, differs from the other distributions to be described in the following [Section 8.1.3](#). Presently, CINSim features four different time depended distributions to be specified via the attribute *traffic*. All of these distributions can be set up either for all source buffers together using the local/global menu or for each source buffer independently as discussed in [Section 5.2](#).

Pareto and Random-Burst Distribution refer to the ON/OFF model. In contrast to an artificial constant or even periodic rate of packets to be passed to a simulated network, this model allows a more lifelike behaviour of the generated traffic. *ON periods* with a high rate of generated packets (burst) are considered as well as *OFF periods* with a lower rate (gap). See [\[3\]](#) and [\[15\]](#) for more details.

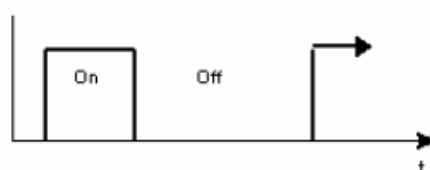


Figure 8.1. ON/OFF Model

Geometric Distribution

A packet is generated at random points in time with a given probability. A probability of 60 % means that at any point in time the probability that a packet can be generated equals to 60 %. In other words, a packet can be generated in 60 % of the simulated clock cycles, which is considered as the offered load of the source buffer. CIN-Sim also uses an adjustment term for packet sizes greater than one.

name	description
load	The offered load.

Table 8.1. Geometric traffic attributes

Periodic Distribution

Packets are generated with a constant rate, for instance every third clock cycle beginning at cycle five. Two attributes are used to configure this type of traffic.

name	description
modulo	The modulo factor to be used.
offset	The first network cycle a packet can be generated.

Table 8.2. Periodic traffic attributes

A packet will be generated if $clock\ cycle - offset \div modulo$ equals to zero. It should be noted, that packets with a size greater than one need several clock cycles to leave the buffer queue.

Pareto Distribution

Most of the traffic of the real world backbone networks seems to be self-similar. The Pareto algorithm can emulate some kind of self-similar traffic. This type of traffic distribution results in streams (trains) of packets which form the ON periods. These periods are interrupted by OFF periods without generating packets. The length of both periods are distributed using the Pareto distribution depending on several parameters: The desired offered load, the (constant) packet size and the shape parameters for ON and OFF period. How these parameters work together can be found in [3] and [15].

name	description
load	The offered load.
onShape	Shape parameter of the ON period (burst).
offShape	Shape parameter of the OFF period (gap).

Table 8.3. Pareto traffic attributes

The values for *onShape* and *offShape* must lie between one and two to create self-similar traffic. The combination *onShape* = 1.7 and *offShape* = 1.1 turned out to deliver acceptable results [15].



Note

Only packets of size one can currently be used in conjunction with the Pareto distribution.

Random-Burst Distribution

The random-burst distribution applies to various phenomena of discrete nature whenever events come suddenly of high amplitude. Applied to a source buffer we speak of bursts in the sense of a stream of packets generated

within a certain time interval with high rate and gaps of certain length with low packet rates.

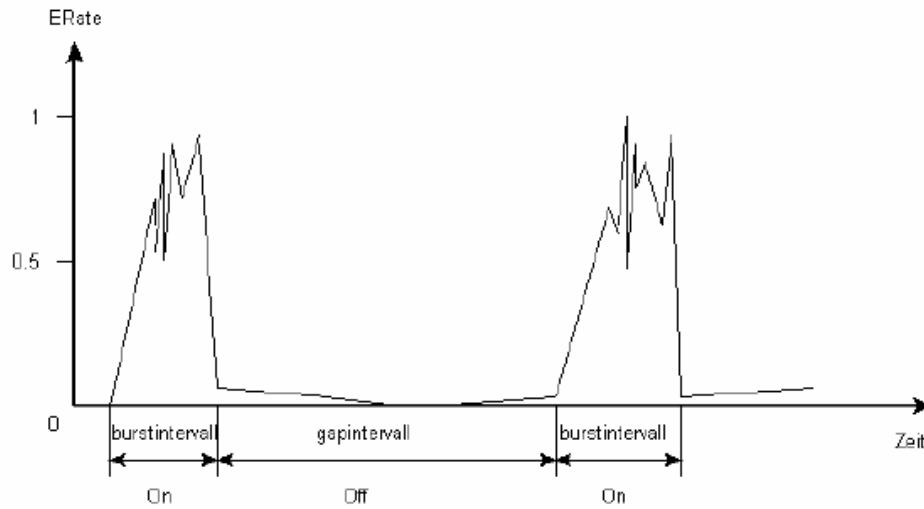


Figure 8.2. Bursty traffic

In contrast to the Pareto distribution, the offered load for ON and OFF period can be specified separately and packets can be generated during the OFF period. The lengths of both periods are randomly distributed (uniform) using lower and upper bounds. The offered load for the ON period (burst) should exceed 50 % while the offered load for the OFF period (gap) should not exceed 20 % to create bursty traffic. Refer to [15] for more information.

name	description
burstLoad	The probability of generating a packet during a burst (offered load).
gapLoad	The probability of generating a packet during a gap (offered load).
gapIntervalMin	The minimal gap size.
burstIntervalMin	The minimal burst size.
gapIntervalMax	The maximal gap size.
burstIntervalMax	The maximal burst size.

Table 8.4. Bursty traffic attributes



Note

Only packets of size one can currently be used in conjunction with the Random-Burst distribution.

8.1.3. Cumulative Distribution Functions

The multicast, spatial and priority distributions applied by source buffers to set up packets, work almost identical, except that different elements need to be distributed. Assuming, that a fixed number of elements should be distributed, one can use a distribution function to specify the probabilities for each element to be chosen. CIN-Sim supports this approach by giving the user the ability to define a formula using a simple gnuplot like language as described in Section 5.2.4. In conjunction with borders that need to be specified, one can define many different distributions to be used.

```

m=3.5
v=0.75
f=exp(-(((t-m)/1.6)^2)^(2*v))
f

```

The sample formula above results in the red function curve shown in Figure 8.3. The syntax of the mathematical language for distribution functions uses infix-notation, the complete list of available syntax elements can be found in Section A.1.

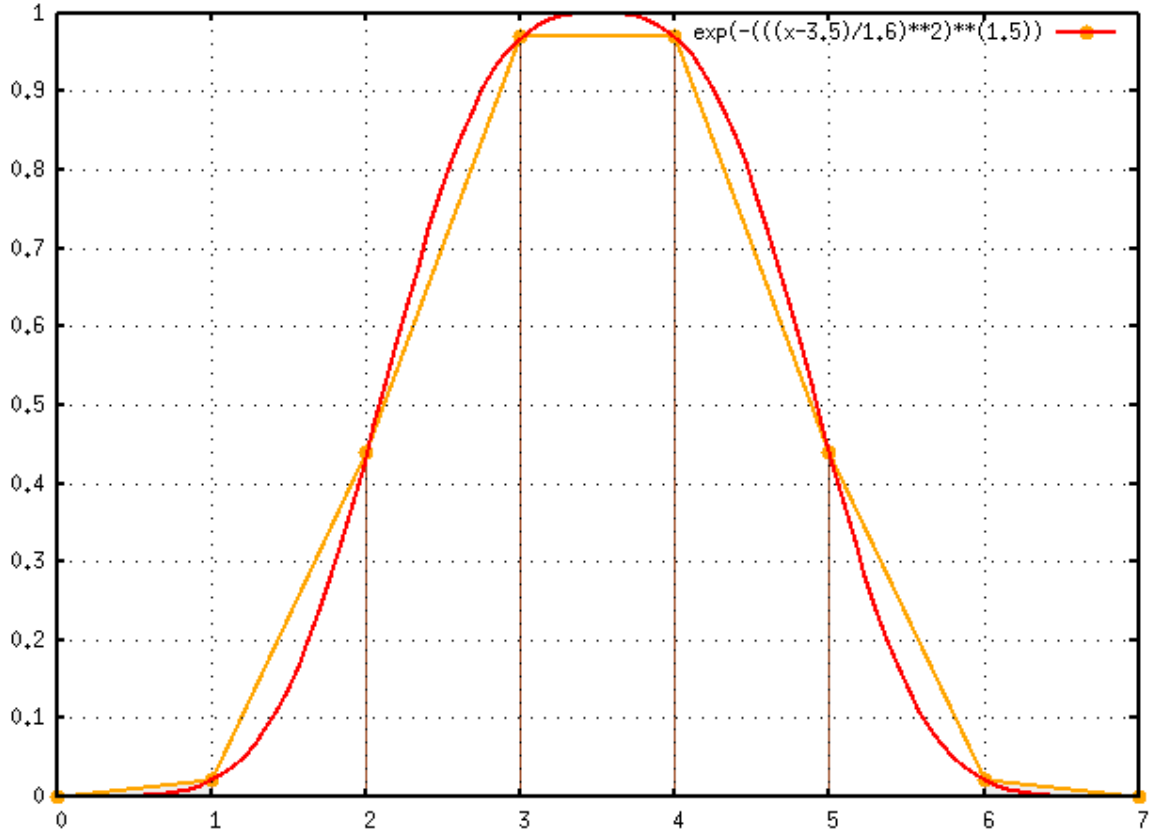


Figure 8.3. Non-normalized sample multicast distribution function $f(n)$

Usually the probability $P(a \leq x \leq b)$ is being calculated using the area under the curve between a and b . Unfortunately integration is an expensive, resource-consuming procedure. Apart from that we want to choose discrete points only, thus a density function is not needed and CINSim abstains from integration. The normalized cumulative distribution functions are used instead, whereas the elements to be distributed are uniformly distributed between the left and the right border (inclusive).

Assuming that four elements are to be distributed using the formula above and the borders $x=0$ and $x=3$, the first element, according to an order defined on the elements, will be placed on the left border and the last element on the right border. The remaining elements will be placed between them at $x=1$ and $x=3$. This results in a distribution with a probability of (almost) zero for the first element to be chosen, and a maximum likelihood for the last element to be chosen. Whenever an element is chosen the distribution will be normalized.

In contrast to calculating the initial (unnormalised) values of the elements to be distributed from a formula, CINSim allows to specify the initial values via a semicolon separated string as well. The string has to start with *all=VALUE*, which sets a common value for all elements. Then values for each single element can be specified with an equation $ID=VALUE$, whereas ID depends on the elements to be distributed and $VALUE$ can be any positive decimal number. If the given values for all elements sum to one, they represent the probabilities for each element to be chosen, otherwise it can be interpreted as a weighting. The string *all=1* defines a uniform distribution, independent from the number of elements to be distributed, whereas the string *all=1;1=2* defines a distribution with a two times larger probability for the element 1 to be chosen than for all other uniformly dis-

tributed elements. Assuming four elements to be distributed, the string *all=0.2;0=0.4* defines a distribution, with a probability of 40 % for the element 0 to be chosen and a probability of 20 % for the remaining elements.

Multicast Distribution

The multicast distribution of a source buffer considers the elements $1, \dots, n$ to be distributed, whereas n represents the number of target buffers reachable from the source buffer. The element 1 represents the *unicast* and the element n the *broadcast*. CINSim provides three built-in multicast distributions: *Unicast*, *Broadcast* and *Uniform*, which can be set using the attribute *multicast* of a source buffer or the related entry in the local/global setting menu. For a customized multicast, the same attribute can be set to *Formula* or *Probabilities*. In the first case, the attribute *mc_formula* must specify the formula and the attributes *leftBorder* and *rightBorder* the borders to be used to calculate the cumulative distribution function. The unicast will be placed on the left border and the broadcast on the right. If *Probabilities* is selected, the attribute *mc_probabilities* is used to specify the distribution, whereas the IDs $1, \dots, n$ can be used.

CINSim does only support unicast for dynamic reconfiguration while static reconfiguration is simulative for all variants of unicast/multicast. Multicast for dynamic reconfiguration is conceptually prohibited.

Spatial Distribution

In contrast to the multicast distribution, the spatial distribution of a source buffer considers the index strings $i(1), \dots, i(n)$ of the reachable target buffers, provided by their attribute *index*, to be distributed. CINSim provides the built-in distribution *Uniform* which can be set via the attribute *spatialDistribution* of a source buffer or the related entry of the local/global settings menu. This distribution uses the same probability for all reachable targets. For a customized spatial distribution the same attribute can be set to *Formula* or *Probabilities*. Using *Formula* involves the attributes *sd_formula*, *sd_leftBorder* and *sd_rightBorder*, which set the formula and the borders to be used. The problem with this approach is that the order of the reachable targets is currently undefined as it depends on the order the target buffers are parsed and added to a network configuration. This drawback can be compassed by using *Probabilities* and setting the attribute *sd_probabilities*. In this approach the unique index strings can be used to specify the probability for each target buffer to be chosen.



Note

It is currently recommended to use *Probabilities* to set up a customized spatial Distribution.

Priority Distribution

The priority distribution of a source buffer distributes the elements $0, \dots, p$ representing the possible priorities for newly created packets. The element p is set via the attribute *maxPriority* of the related entry of the global settings menu. Again, CINSim provides the built-in distribution *Uniform*, which can be set via the attribute *priorityDistribution* of a source buffer or the related entry of the local/global settings menu. This distribution does not prefer any of the available priorities. For a customized priority distribution the values *Formula* and *Probabilities* can be used. In the first case, the attributes *pd_formula*, *pd_leftBorder* and *pd_rightBorder* must be set appropriately to specify a formula and the borders. In the latter case, the attribute *pd_probabilities* holds the string defining the distribution, whereas the IDs $0, \dots, p$ can be used.

8.2. Buffer Component

Buffer components are used to store packets along their way through an interconnection network. Within every single clock cycle only one packet fragment (flit) can be received and stored while on the other hand one fragment can be sent and removed. Bypassing is not supported. The fragments are stored in a *queue* component. A *register* component is wrapped around a queue component and analyses the next buffer components a packet is to be stored. If a multicast is applied more than one buffer may be served, therefore a register creates flit-by-flit copies of each packet according to the number of receiving buffers. Each copy will be stored separately and tagged with a new target bitmask representing the reachable targets when stored to its designated buffer component. If a single flit, strictly speaking its copies, has been sent to all served successors it will be removed from the buffer's queue, increasing the free buffer space by one. If a successor receives a flit, the register will make the next flit of the same packet available in the next clock cycle for this component. The next packet, however, will only be available if all successors received the complete packet currently distributed by the register. This implementation of the register component leads to *partial* multicast. For regular network topologies, like MINs and BMINs, the multicast will take place as late as possible, while for irregular topologies a nondeterministic multicast behaviour may occur.

The physical buffer space is determined by two parameters to be set using the attribute table of the global settings menu ([Section 5.2.2](#)) or the attribute table of a single buffer component. The attribute *minSpace* sets the lower bound for storeable flits while the attribute *maxSpace* sets the actual buffer space. The lower bound can only be set globally for all buffer components, the actual buffer space can be set independently for each buffer component. If the lower bound is under-run, the buffer space defaults to the lower bound. Some switching-techniques described in detail in [Section 7.3](#) assume that each buffer component can store a whole packet. Setting the lower bound appropriate according to the size of the generated packets guarantees that this prerequisite will be satisfied without checking all buffer components one by one.

8.2.1. Queue Types

According to its component based approach, CINSim features an interface for different types of buffer queues. Presently, only a simple non-shared FIFO queue is implemented, following the guidelines given above. The queue to be used can be set globally or locally using the attribute *queueType* which defaults to the implemented *QueueOneSize*. Future versions of CINSim will provide more queue types.

8.3. Target Buffer Component

The target buffers are special buffers that are used as destinations for packets passed through an interconnection network simulated with CINSim. Each target buffer is represented by a single bit within the target bitmasks of packets and network components. Targets can receive one packet fragment (flit) per clock cycle and their queue will be cleared at the end of each clock cycle. Analyser components connected to target buffers will, of course, analyse the received flits first.

8.4. Router Component

The router component is an abstract switch using an I/O matrix to route packets. Routers are responsible for routing, scheduling and switching. Routers have input and output ports. Input ports serve incoming packets while an output port is the contentious resource. The operation of a router can be divided up into the routing, the scheduling and the the switching. Actually, it is rarely possible to distinguish between them, because there are usually cross connections to be found such as port reservations. Nevertheless, we conceived strict borders and made it possible to sequentialize the allover routing process.

Independent from the coroutine which triggers a network node, a router works as follows: The **routing** algorithm will fill the columns and rows of the routing matrix where the row number depends on the input number and the column number refers to the number of the output port. The algorithm will mark the matrix using the packets' bitmasks. Each point of the matrix represents a possible I/O switching. Indeed, a column can be selected by multiple input ports. In this case, a conflict occurs, that can be solved using a **scheduling** algorithm (see [Figure 8.4](#), [Figure 8.5](#), and [\[10\]](#)). For this purpose, an *arbiter* component is assigned to each output port of a router and an *arbiter controller* searches the matrix for conflicting requests for an output port (see [Section 8.4.1](#) for details). Once a decision is made, the **switching** mechanism sets up a connection between an input port and one or (in case of multicast) more output ports which will be removed if the packet has been completely transmitted.

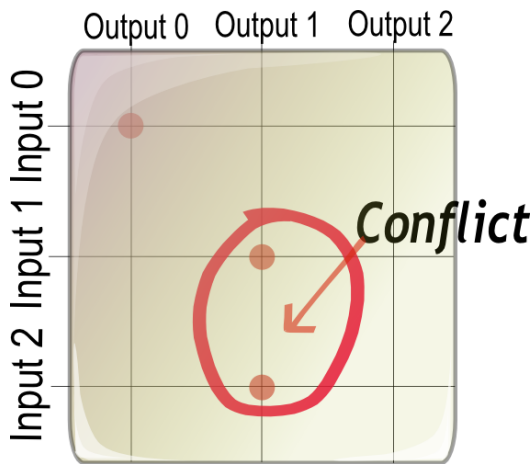


Figure 8.4. Routing matrix with conflict

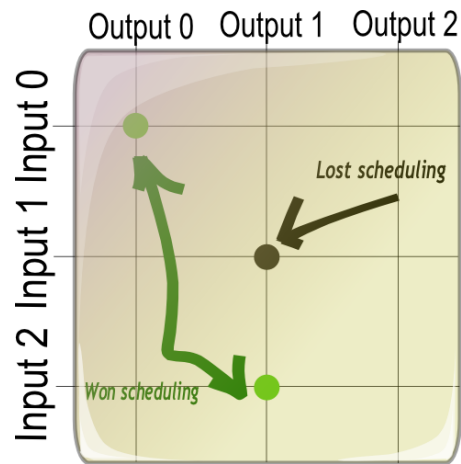


Figure 8.5. Routing matrix after scheduling

8.4.1. Arbiter

Arbiters are listening at the output ports of a router. Each output port is connected to a single arbiter. Arbiters take control of the column of the routing matrix which belongs to the output. Depending on the type of the scheduling, different arbiters are provided. Each arbiter is responsible for solving a conflict for its connected output, hence arbiters cannot include decisions of other arbiters. Unfortunately, some situations may require a communication and exchange of information between arbiters. The decision of an arbiter may depend on the decision of another arbiter, e.g. if one input could be switched to more then one output. A communication between arbiters is implicitly done by the arbiter controller, that activates all arbiters in case of conflicts and handles their results.



Note

All arbiters will call the random number generator, if the scheduling algorithm did not solve the problem.

The following scheduling algorithms are implemented, each represented by an own arbiter.

Random

A conflict is solved using the random number generator. The list of conflicting inputs will be resorted using random order which is accessed in a descending order.

Global Round Robin

The list of conflicting input ports is being accessed in round robin order. Global round robin (GRR) uses a global variable to calculate the current preferred input. Each arbiter uses the same variable. The list of inputs is accessed in ascending order beginning with the list element being calculated by the use of the following equation.

$$\text{clock cycle} \bmod \text{number of router input ports}$$

The figures below show a possible conflict situation at two different sized routers. Figure 8.6 and Figure 8.7 illustrate a column of a routing matrix of a router with three input ports, Figure 8.8 and Figure 8.9 show a column of a router with four input ports. Green bullets mark the winning entry while black bullets represent input ports that lost the scheduling process.

This is a network tick, which is a multiple of three, the number of input ports at this router. This means, that the round robin algorithm will prefer input 0. Because input 0 does not serve a packet, the next input will be selected.

The basis of the equation is the network clock cycle (or the tick), such that all arbiters of all routers in a network will continue the round robin cycle synchronously. Still they will not prefer the same input port, because the router sizes may differ.

Assuming all routers are of the same size, they will always prefer the same input port.

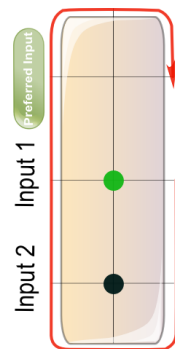
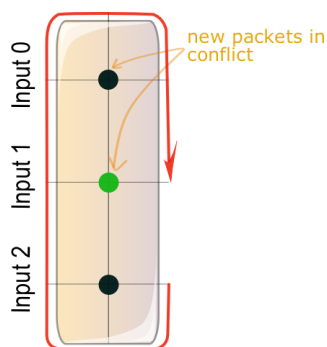
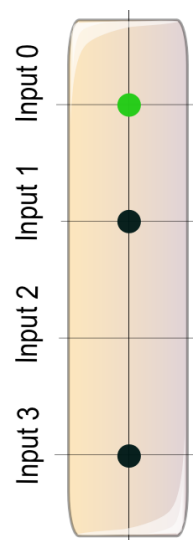


Figure 8.6. GRR (1)

**Figure 8.7. GRR (2)**

This is the following tick. Input port one obviously won the competition again. That's why the global round robin variable complies with the network tick, which is the next tick to the one used in the illustration before. Now the corresponding input port actually serves a packet, which might take the appropriate output.

The following [Figure 8.8](#) and [Figure 8.9](#) show another router with four instead of three input ports. This is a network tick, which is a multiple of four. The round robin algorithm will again prefer input port 0.

**Figure 8.8. GRR (3)**

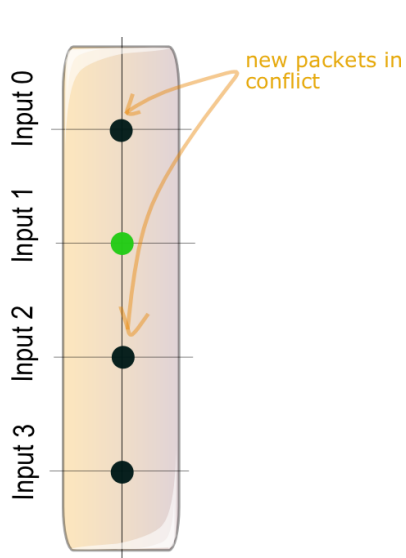


Figure 8.9. GRR (4)

In the following tick of the second example the router will switch input port one to the related output port. Input port two will be preferred next round and input port three the round after the next.

To sum up, one can say that the global round robin algorithm is basically the same as the conventional round robin definition but it is using a global state, such that they would prefer the same input synchronously assuming an equal number of input ports.

Local Round Robin

In contrast to the global round robin scheduling, each arbiter has its own variable reflecting the last switched input port. The next preferred input port is the one that is the next of the last switched input port. If the preferred input port does not serve a packet for the appropriate output port, the next input port will be chosen until there is a packet ready to use the output port. This related input port will be marked as the last switched input port.

Fixed Order

The preferred input port is chosen using a fixed order given by the order of the input ports of the related router component. The preferred input port is the first one according to the specified order. If it has no packet to serve, the next port will be preferred and so on.

Most Recently Used

This scheduling type is a very expensive one. Each arbiter will manage a list of pairs which represents the points in time (network ticks) each input won its last competition for the given output port. The arbiter will walk through the list and select the most recently used input port of the conflicting input ports. The list is sorted in reverse order, starting with the highest tick, hence the top element of the list holds the most recently used input port.

Least Recently Used

The least recently used input port will be preferred in case of a conflict. It is a very resource consuming scheduling, too, as it is using list and sort operations. In contrast to the most-recently-used arbiter, the list is sorted in ascending order, hence the top element of the list will hold the least recently used input port.

Most Frequently Used

This arbiter works similar to the most-recently-used arbiter, as it is also using a sorted list of pairs. But in this case a pair represents the number of wins of an input port. The list is sorted in descending order, thus the top

element of the list holds the most frequently used input port.

Least Frequently Used

This scheduling algorithm is also based on a sorted list. In contrast to the most-frequently-used arbiter, the list is sorted in ascending order, hence the top element of the list will hold the preferred input port.

Oldest Packet First scheduling

This arbiter will analyse the flits at each input port out of the list of conflicting input ports. The input port that serves the oldest flits will be preferred. The list of conflicting input ports is being sorted by the use of the age of the flits at an input port. The age of a flit corresponds to the birth tick of the packet it belongs to, hence the list is being sorted in ascending order and the top element of the list will be selected for the switching.

Longest Waiting Packet First

The longest waiting packet will be preferred. That means the list of conflicting input ports is being sorted in descending order using the queue delay of the according flit at an input. The top element of the sorted list will be selected for switching.

Priority Scheduling

Source buffers will stamp packets with a priority number according to the priority distribution used. An arbiter using priority scheduling will sort the list of conflicting input ports in descending order according to the priorities of the packets to be transmitted, thus the top element is the input port that serves a flit belonging to a packet with the highest priority.

Deadline Scheduling

The deadline scheduling will prefer a flit belonging to the packet with the living time closest to the deadline. The list of conflicting inputs is being sorted in ascending order using the deadlines of the packets at each input port.

8.4.2. Arbiter Controller

The arbiter controller is an abstract meta component. It manages the list of arbiters. Whenever a router tries to switch an output port the arbiter controller will activate the corresponding arbiter in case of a conflict. It can evaluate the resulting list of scheduled input ports and reorder the list due to its own algorithm.

This is meant to solve a complex problem, that cannot be solved from an arbiter's point of view. A controller can interpret the outcome of an arbiter, maybe in case of redundant paths, e.g. to select the best path. Assuming input 0 won output 0 and output 1. In case of redundant paths, the controller may select the best path and choose the second input port out of the list of scheduled ports for the other available path. Presently, CINSim supports only first fit selection only. That means, the first available path is being selected in case of redundancy. The arbiter controller will activate the arbiter of an activated output only if necessary.

8.5. Analyser Component

This component has been introduced to provide some kind of measurement device that can be connected to other components of a network but it is actually not a network component. An analyser refers to one or more measurement variables defining measurements ([Section 5.2.5](#)). A measurement variable belongs to one parameter observed by CINSim or, in case of terminating simulations, to a sequence of parameters ([Section 7.4](#)). Multiple analysers may share the same measurement variables and therefore the same parameters.

Analysers will trigger their connected components one time at the end of each clock cycle. Depending on the type of measurement (delay, target throughput e.g.) the triggered components will return values to be used for analysis. As already mentioned, multiple analysers can share a single measurement variable, allowing more than one measurement (of the same type) to be written to the same measurement variable. This makes it possible to observe an overall target throughput, for example.

It is worth mentioning, that some combinations of measurement types and components do not make sense. For instance, you cannot observe the target throughput of a source buffer. The connected component will not have such a proceeding implemented. The default behavior in this case is to ignore the request. The simulation will then not terminate because the associated parameter will not get values unless other analysers are invoked.

8.5.1. Observation Types

CINSim presently supports the following observation types, specified by the measurement variables used.

Delay (at targets)

This kind of observation calculates the delay of packets passed through the network according to the difference of the arrival time (clock cycle) of the head flit at a target buffer and the packet's birth time.

Connecting an analyser referring to a measurement variable of this type to more than one target will observe the network's delay times according to the packets received by those target buffers. Including all target buffers will therefore lead to the overall network delay times.

Latency (at targets)

An observation of type *Latency* at target buffers is equal to an observation of type *Delay*, except that it uses the arrival time of the *tail* flit.

Delay (at buffers)

Analysers including measurement variables of this type connected to buffers will check the delay times of packets within the buffer's queue. The difference between leaving and arrival time (clock cycle) of the head flit of a packet will be considered.

**Note**

Connecting an analyser of this type to buffers and targets leads to undetermined results.

Latency (at buffers)

The same as *Delay*, except that the difference between leaving time (clock cycle) of the tail flit and the arrival time of the head flit of a packet will be considered.

SourceThroughput

The *SourceThroughput* measurement depends on a source buffer's load and the availableness of network resources. The source throughput is flit-based and delivers *1* if a flit has been forwarded to the network and *0* otherwise. Depending on the occurrences of 1 and 0, the mean results to something between 0 and 1.

**Note**

Only source buffers deliver values for this kind of observation.

TargetThroughput

This measurement observes the flit-based target throughput at one or more target buffers. The probability of the arrival of a flit at a target meets the definition of target throughput measurements.



Note

Only target buffers deliver values for this kind of observation.

Queue

An analyser referring to a variable of this type observes the mean queue length of a buffer.



Note

Only buffers deliver values for this kind of observation.

Priority measurements

These measurements only consider packets or flits that have the given priority.

8.5.2. Measurement Statistics Types

CINSim presently supports the following measurement statistics types, specified by the measurement variables used [\[4\]](#).

Mean

This kind of measurement determines the mean (average value) of the corresponding observation type.

Quantile p

This kind of measurement determines the p -quantile of the corresponding observation type. The p -quantile gives the value of the measurement variable for which the fraction p (means: $p \times 100\%$) of all its calculated values is less than this p -quantile value.

Chapter 9. Reconfiguration

Reconfiguration of interconnection networks can be simulated both with the *static* [15] and the *dynamic* [16] approach. Both of them can be described by using the graphical user interface as introduced in [Section 5.2.7](#). The weaker *static* reconfiguration mechanism considers three time parameters that must be set using the attribute table related to the arrow connecting two network configurations. The arrow specifies a reconfiguration step to transform the configuration it starts at into the configuration it ends up.

The *operation phase* defines the number of clock cycles the first configuration will be used until the reconfiguration process begins. This can start with the initial clock cycle of the simulation or the first clock cycle after the last reconfiguration step. In this phase the simulation will run like any other non-reconfiguring simulation.

The *packet drain phase* describes the first phase of the reconfiguration process. In this period the source buffers of the currently simulated network configuration will not produce any new packets and the packets already somewhere within the network are to leave it. If the network is not empty after the end of this phase is reached, the remaining packets are removed. This is the easiest way of reconfiguration because it has not to be taken care of old packets with invalid routing tags according to the new network configuration.

The *reconfiguration phase* specifies the hardware dependent time for switching to the new network configuration. After this phase the reconfiguration process is finished and the new network configuration will be used.

The stronger approach is the *dynamic* reconfiguration. The same time parameters are considered that are to be set within the modelling of the reconfiguration in the graphical user interface. The arrow again depicts the reconfiguration process. Dynamic reconfiguration is actually very similar to the static case with the exception that there is a *packet collection phase* instead of the packet drain phase. The packet collection phase length is calculated by "packet length - 1" and does not have to be specified in the GUI. Packets remain in the network during reconfiguration, what makes the overall reconfiguration process faster. In the packet collection phase, the flits of a packet will be collected within a single buffer. Packet heads cannot be transmitted any more. This phase lasts at most one cycle less than the maximum packet length and is necessary to prevent packet breakup. For dynamic reconfiguration "Store And Forward" and "Partial Cut Through" are allowed as switching techniques. "Worm-hole Switching" is not permitted. Note that "Virtual Cut Through" has complex switching rules so that CINSim's current implementation is not able to simulate all cases correctly and thus prohibits this technique as well.

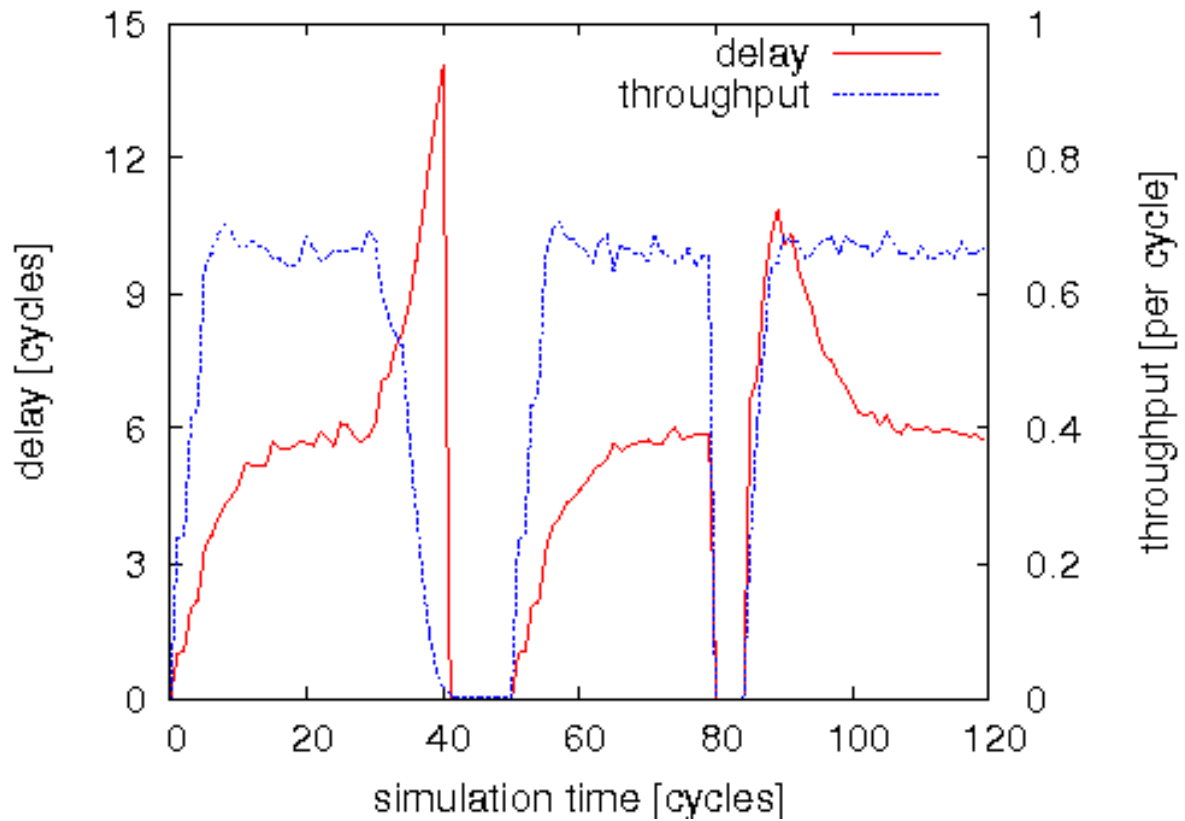


Figure 9.1. Simulation example

Figure 9.1 shows a simulation example containing both static and dynamic reconfigurations. A bidirectional multistage interconnection network (BMIN) is simulated and reconfigured twice. First it is reconfigured statically, then dynamically. The graph demonstrates how delay and throughput of the network behave before and after both of the reconfigurations. First, the delay rises in the packet drain phase and the throughput is falling to zero. After a phase of five cycles of reconfiguration, beginning at cycle 45, the network is filled again with packets. Delay and throughput show the same characteristics in cycles 50-79 as in cycles 0-29. The upcoming dynamic reconfiguration shows a different behaviour. Throughput and delay go to zero immediately (because we have packet length 1 and thus no collection phase) and rise again after five cycles of reconfiguration. Delay rises higher than before because remaining packets have higher ages now. Stationery throughput level is reached slightly slower than before, because the network was already filled and attenuates transmitted packets. However, dynamic reconfiguration reaches stationery levels faster than the static counterpart.

Appendix A. Distribution Functions

A.1. Syntax Description

Name	Description
sin	sine
asin	arc sine
cos	cosine
acos	arc cosine
tan	tangent
atan	arc tangent
ln	natural logarithm
log	common (base 10) logarithm
exp	exponential function
sqrt	square root
abs	the absolute value of
tanh	hyperbolic tangent
atanh	arc hyperbolic tangent
sinh	hyperbolic sine
asinh	arc hyperbolic sine
cosh	hyperbolic cosine
acosh	arc hyperbolic cosine
cot	cotangent
csc	cosecant
sec	secant
sgn	signum
sign	signum
lgamma	the logarithm of the gamma function
erf	error function
erfc	complementary error function

Table A.1. Functions

name	description
expr + expr	plus
expr - expr	minus
expr * expr	multiplication
expr / expr	division ^a
-expr	neg. operator
expr!	faculty

name	description
expr	the absolute value
exprs ^ expr	power function

Table A.2. Standard operators

^aall numbers will be treated as doubles

name	description
a	zero
0	zero
e	Euler number
pi	Π

Table A.3. Constants

name	description
t	the abscissa

Table A.4. Variables

Appendix B. Sources of Information

B.1. CINSim

1. [The CINSim Homepage](http://pdv.cs.tu-berlin.de/cinsim) (<http://pdv.cs.tu-berlin.de/cinsim>)
2. Online Tutorials (URL)
3. The CINSim Mailinglist < *cinsim AT pdv.cs.tu-berlin.de* >

B.2. MPI Implementations

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

B.3. GNU Scientific Libraries

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

B.4. Xerces-C++

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

B.5. Bison++

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

B.6. XSLT Processors

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

Glossary

A

arbiter See [scheduler](#).

B

backpressure We distinguish two backpressure mechanism - *global backpressure* and *local backpressure*.

Local backpressure only observes the destination buffer of the next stage: The packet of stage k is sent if space at stage $k + 1$ is available.

Global backpressure gets additional information about packet flows: The packet of stage k is sent even if no space at stage $k + 1$ is available but becomes available till the packet is received. Such a situation may arise if a packet leaves stage $k + 1$ at the same clock cycle.

best-fit algorithm The best-fit algorithm proceeds through a sorted list of items (output ports) until running out of items or meeting the best criteria (e.g. most places in the buffer of the next stage at the according output port).

bitmask A **bitmask** is usually an assembled list of bits. Each bit, a group of bits or even the bitmask itself represents data as like as a key for an object of a more complex data-type or crypted information such as targets of a packet. Bitmasks are often used to prevent a program from wasting memory. That's why toggling bit 7 of a bitmask instead of allocating an integer value 7 is obviously more effective.

Lets take a look on a short example. Assuming we store the targets of a packet using a bitmask a mask may look like this:

```
1111
```

Each bit represents a unique target-index. The packet will be send to target 0,1,2 and 3 in the example. The translation of a mask depends on the case using it, one may say bit 0 refers to target 0 or one may say it refers to target 42 if you like. As you might have already noticed it is a [broadcast](#).

bitmask routing **Bitmask routing** is a routing type. It can be used to discover a path through a network using a [bitmask](#) representing the targets of a packet. See Also [routing type](#).

broadcast A **broadcast** allows information to be sent to all targets on a given net rather than a specific target. The exact notation can vary by implementation (CINSim uses [bitmasks](#)).

C

circuit switching A **circuit switching** technique allocates the entire path to be used for a message from a given source to its destination first, before the message is actually passed to the network. The path will be deallocated after the message

	has been completely received by the destination component. See Also packet switching .
CINSimMetaNet	XML type of a net component nested within a meta element.
CINSimNet	XML type of a nested net component specifying a network configuration.
CINSimPlus	Name of the net class extension used by <i>cinsim-gui</i> to define simulation sets for the simulation core of the CINSim project.
CINSimPlusNet	XML type of the top level net component of a simulation set.
ConstRandom traffic type	ConstRandom is the CINSim-gui parameter for random traffic-load.
ConstUniform traffic type	ConstUniform is the CINSim-gui parameter for periodic traffic. Two attributes configure this traffic type - the modulo factor and the offset factor. If (time-offset)%modulo-factor equals 0 a source buffer will generate a packet.
controller type	This is a CINSim-gui parameter. The controller type implements the <i>knapsack</i> algorithms. See Also best-fit algorithm , first-fit algorithm .
crossbar	See crossbar switch .
crossbar switch (crossbar)	A Crossbar switch is one of the principal architectures used to construct switches of many types. Crossbar switches are sometimes referred to as "Cross-point switches" or "Crosspoint switches". The other principal switch architectures are that of a Memory switch or a crossover switch. A Banyan switch is an important type of crossover switch. If we talk of a crossbar we usually think of a router-buffer construct. Wires connect those. The router switches its inputs to its outputs using special rules called routing mechanism.

D

deadline scheduling (earliest deadline first)	At any scheduling decision time, the packet with the earliest deadline is scheduled. Deadline is calculated as the upper bound of the tolerable source-to-destination (end-to-end) delay, and thus measures the usefulness of data packets at the destination. See Also scheduler .
delay	The delay of a packet in a system is given by the difference of the arrival time (clock cycle) of the head flit at the system's output and the arrival time of the head flit at the system's input. If the delay is measured at the target buffer, the entire network represents the system in question. If the delay at any other buffer is measured, this buffer is the system in question. See Also observation type .
deadlock	A deadlock is a situation wherein two or more competing actions are waiting for another to finish, so neither ever does. A routing-protocol is deadlock-free if each packet can be forwarded. If a routing-protocol is not deadlock-free it is usually caused by a paradox, like the chicken or the egg. A situation may occur where the decision of a router sending packet X depends on its decision. That's often to be seen in cyclic networks. Deadlock-free routing algorithm can be constructed by restricting the set of routes within a network.

F

first-fit algorithm	The first-fit algorithm proceeds through an unsorted list and selects the first.
fixed-order scheduling	Conflicting ports are sorted by a fixed order, e.g. beginning with port 0. See Also scheduler .
forwarding type	Determines the state of switches. We distinguish between <i>local</i> and <i>global backpressure</i> mechanism. The forwarding type is implemented as a coroutine in the simulation program. See Also backpressure .

G

global backpressure	See backpressure .
---------------------	------------------------------------

L

latency	<p>The latency of a packet in a system is given by the difference of the arrival time (clock cycle) of the tail flit at the system's output and the arrival time of the head flit at the system's input.</p> <p>If the latency is measured at the target buffer, the entire network represents the system in question. If the latency at any other buffer is measured, this buffer is the system in question. See Also observation type.</p>
least-frequently-used scheduling (LFU)	<p>The least frequently used port will be preferred. All input ports targeting the same output port will be ordered in dependence from the frequency of the usage of that output by the appropriate input. The less an input port uses an output port the higher is the chance to win independent of the last time of the usage. See Also scheduler.</p>
least-recently-used scheduling (LRU)	<p>This attempts to predict future use by looking at past use. If a port has not been recently used then it is unlikely that it will be needed again soon. This is usually because of sequential packet generation or bursts or some bottle neck structures in a network. This requires that a list be kept of usage of each port, in order of last use. Whenever an input-output reference occurs, this list must be updated, moving entries around to keep the list up to date. This makes this system prohibitively expensive in general. See Also scheduler.</p>
local backpressure	See backpressure .
longest-waiting-packet-first scheduling	<p>Longest waiting packets at a switch will be scheduled. Conflicting packets will be sorted by the time they entered the according buffer. See Also scheduler.</p>

M

mean	<p>The mean determines the average value of the corresponding measurement variable. See Also measurement statistics type.</p>
------	---

measurement statistics type		The measurement type defines the kind of measurement statistics that is performed by the measurement variables. See Also mean , quantile .
MeshNet		XML type of a net component specifying a mesh configuration.
Message-Passing Interface (MPI)	Interface	A language-independent framework to realize parallel applications. Synchronization and communication between processes is realized by messages. MPI provides a standardized semantics by an API. MPI programs can be compiled for a large variety of parallel and multicore computers.
most-frequently-used scheduling (MFU)	scheduling	<p>The most frequently used sort algorithm places the ports most frequently used at the top of the list and the least frequently at the bottom. New port mappings are served from the top of the list.</p> <p>The difference in the performance of the algorithms <i>MFU</i> and <i>MRU</i> depends on the height and width of the data <i>hot spots</i>. <i>Hot spots</i>, in this discussion, refers to the source-destination combination in the network to which a higher percentage of I/O requests are targeted. Which one of these is to be preferred depend on the shape of the hot spots. See Also scheduler.</p>
most-recently-used scheduling (MRU)	scheduling	<p>This places ports in an order where the most recently used ports are at the top of the 'list', and the least recently used ports are at its bottom. When requests requiring new port mappings are serviced, ports at the bottom of the list are used. See Also scheduler.</p>
multiple replications in parallel (MRIP)		Simulating the whole model independently in parallel with different sets of random numbers to reach faster the desired precision and confidence level. All independent measures are collected to estimate the final results.

N

net class		The graphical user interface <i>cinsim-gui</i> of the CINSim project uses a generic editor framework for net graphs specified in XML, allowing schema-driven editing. The framework must be extended by a net class to define the type of net graph to be edited. A net class consists of a XML schema, extending the framework's base types for net components, Java classes, defining the behaviour of the components, and graphic descriptions, defining the components' visualization.
-----------	--	--

O

observation type		The observation type defines the kind of observation that is performed by the measurement variables. See Also delay , latency , throughput , queue length , priority observation .
oldest-packet-first scheduling (OPF)	scheduling	The OPF scheduling algorithm [6] was originally devised for input-queued packet switches. The oldest packet will be scheduled. See Also scheduler .

P

packet switching		A packet switching technique divides a message into packets that are passed through the network independent from each other. A path connecting the source of a packet and its destination is only partly allocated, depending
------------------	--	--

	on the buffer components holding fragments of the packet. See Also circuit switching .
partial cut-through routing	See partial cut-through switching .
partial cut-through switching (partial cut-through routing)	In comparison to the full virtual cut-through switching the head can be forwarded while the rest of the message is still being received. See Also wormhole switching .
priority observation	This kind of observation only considers packets or flits of a given priority. See Also observation type .
priority scheduling	Conflicting ports will be ordered by their packets priority. Whether a high value indicates a high priority or the other way round depends on interpretation (CINSim prefers packets with high values for the packet priority).

Q

quantile	The p -quantile gives the value of the measurement variable for which the fraction p (means: $p \times 100\%$) of all its calculated values is less than this p -quantile value. See Also measurement statistics type .
queue length	The queue length of a buffer denotes the number of flits in this buffer. See Also observation type .

R

round-robin scheduling	<p>Round-robin is the simplest scheduling algorithm for schedulers, which assigns output ports to each input port in equal portions and order, handling all ports as having the same priority. In prioritized scheduling systems, ports on an equal priority are often addressed in a round-robin manner (CINSim does a random scheduling here, an advantage of software implementations).</p> <p>Local round-robin will store the last winning port at the appropriate switch. Thus each switches scheduling is independent from the others decisions.</p> <p>Global round-robin is a static scheduling. The assignment of an input- to an output port still differs from router to router. That's why routers with empty ports will not schedule. Apart from this the basic rules for a decision are the same at each router. If during the last round the preferred port is the first one, the second round will prefer the second port at each router and so on. See Also scheduler.</p>
routing	See routing type .
routing type	<p>Routing type provides the means of discovering paths along which packets can be sent. The route of a packet depends on routing techniques which switches use to guess where to move a packet. See Also bitmask routing, shortest-path routing, west-first routing, standard-xy routing.</p>
routingType	This is a CINSim-gui parameter (attribute) used to set the routing type of a simulation. See Also routing type .

S

scheduler	<p>Scheduling is the process of assigning input ports to a set of resources (output ports). It is an important concept in many areas such as switching processes of routers. A scheduler uses a schedule-routine like round-robin scheme to balance the incoming load over time.</p> <p>See Also round-robin scheduling, fixed-order scheduling, least-frequently-used scheduling, most-frequently-used scheduling, least-recently-used scheduling, most-recently-used scheduling, oldest-packet-first scheduling, longest-waiting-packet-first scheduling, deadline scheduling, priority scheduling.</p>
shortest-path routing	<p>Shortest-path routing is a routing type used to calculate the shortest path to a destination at a switch. Actually the implementation of CINSim does not calculate the shortest path at each node. The distances in a network will be calculated right after network creation before the simulation. Our algorithm is not an implementation of the Open Shortest Path First (http://www.ietf.org/html.charters/ospf-charter.html) (OSPF) Algorithm and does not make use of the well-known Dijkstra's algorithm. That is because we simulate static networks without edge weights.</p>
simulationType	<p>This is a CINSim-gui parameter used to set the type of a simulation.</p> <p>See Also simulation type.</p>
simulation type	<p>We distinguish steady state and transient simulations.</p> <p>Steady simulations are simulations that analyse characteristics of a network until the steady-state is reached.</p> <p>Transient simulations are terminating simulations. A simulation interval will be observed.</p>
standard-xy routing	<p>Like the west-first routing standard-xy routing is a deadlock-free routing algorithm, too. Considering a two-dimensional array network-model turns from the Y- to the X-dimension are not allowed. XY-routing allows only 4 of 8 possible turns.</p>
statistics type	<p>See measurement statistics type.</p>
store-and-forward switching	<p>Store and forward is a switching technique in which packets are sent to an intermediate station where they are buffered and sent at a later time to the final destination or to another intermediate station.</p> <p>CINSim provides store-and-forwarded packets of variable size. It is not intended to switch wormhole packets, for example, using store-and-forward switching. The latter does not guarantee keeping the worm intact.</p>
switching	<p>See switching type.</p>
switchingType	<p>This is a CINSim-gui parameter used to set the switching type of a simulation.</p> <p>See Also switching type.</p>
switching type	<p>Independent from the routing type we distinguish between four implemented switching types. These describe basic rules for handling packets. Apart from the type of a packet, different switching types are needed or make sense. The switching type does not determine the path along the net. What it simply does is to handle things like reservation or higher rules like preventing bubbles within a wormhole packet. There are also known switching types that combine several switching types. For instance. The Hybrid switching combines wormhole and virtual cut-through switching.</p>

See Also [store-and-forward switching](#), [wormhole switching](#), [virtual cut-through switching](#), [partial cut-through switching](#).

T

throughput The throughput denotes the number of flits per clock cycle, that pass the input (source throughput) or output (target throughput) of a network, respectively.
See Also [observation type](#).

V

virtual cut-through routing See [virtual cut-through switching](#).

virtual cut-through switching (virtual cut-through routing) In comparison to the [wormhole switching](#) the head of the message is blocked at the current node if the output port is not free. The head will pass the node if the message is received and buffered. Thus as a crucial prerequisite the size of each buffer must be greater or equal than the length of the message. A message does not block ports while waiting for a free output port.
See Also [partial cut-through switching](#).

W

west-first routing **West-first routing** is used in case of mesh networks to prevent [deadlock](#) situations. A packet is forwarded to the west port first if necessary and then adaptively routed to the south, the east or the north. A turn to the west is prohibited.

wormhole routing See [wormhole switching](#).

wormhole switching (wormhole routing) **Wormhole switching** has been extensively deployed in the current generation of high-end parallel systems. Its based on special communication packets called worms. A worm consists of at least three different kind of flits - head, body and tail flits. The main idea of wormhole switching is to shorten the transmission of a packet by establishing a path along the net between current positions of the head and the tail flits of a packet. The head of the packet is usually tagged with the routing information. A router that switches a head flit using the routing tag reserves the appropriate local output until the tail flit of the same packet passed that output. The head will cause reservations at each node on its path. The crucial advantage is that collisions are dealt by the head only. The body and the tail will pass all switches behind the head without collisions.

However wormhole switching has also some disadvantages. A main one is that messages block in place when the link requested by the header is busy. So data flits spawn over multiple routers, leading to significant link blocking. Blockings can be reduced by using separate buffers, or virtual ports, associated with each physical link. Apart from this there are solutions, most of them using virtual ports that also solve the deadlock problem.

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] Krysztow 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 Lüdtkke, 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 Lüdtkke, 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.

Index

A

analyser
 measurement statistics type, 90
 observation type, 89

B

backpressure (see coroutine)

C

cinsim-gui
 attribute tables, 37
 CINSimPlus menu, 39
 connections, 35
 Edit menu, 33
 File menu, 33
 tool bar
 bottom, 35
 top, 34
 View menu, 33
 Window menu, 34
component
 analyser, 89
 buffer, 82
 router, 84
 source buffer, 77
 target buffer, 83
coroutine, 70
 global backpressure, 71
 local backpressure, 70

D

deadline scheduling, 88
distribution, 77
 cumulative, 79
 geometric, 78
 multicast, 81
 pareto, 78
 periodic, 78
 priority, 81
 random burst, 78
 spatial, 81

F

first fit selection, 88
fixed order scheduling, 87

G

gui (see cinsim-gui)

L

least frequently used scheduling, 88
least recently used scheduling, 87
longest waiting packet first, 88

M

measurement
 mean, 90
 quantile, 90
menu bar (see cinsim-gui)
Message-Passing Interface, 20, 23
most frequently used scheduling, 87
most recently used scheduling, 87
MPI (see Message-Passing Interface)
MRIP (see multiple replications in parallel)
multiple replications in parallel, 20

O

observation
 delay (at buffers), 89
 delay (at targets), 89
 latency (at buffers), 89
 latency (at targets), 89
 priority, 90
 queue length, 90
 source throughput, 89
 target throughput, 90
offered load (see distribution)
oldest packet first scheduling, 88

P

packet
 constant size packets, 77
priority scheduling, 88

R

reconfiguration
 concepts, 17
 parameters, 91
 setup, 57
round robin
 global, 85
round robin scheduling
 local, 87
routing
 bitmask, 72
 shortest-path, 72
 west-first, 73
 xy, 73

S

simulation
 steady-state, 76
 terminating, 76
 transient (see terminating)
switching
 partial cut-through, 75
 store-and-forward, 74
 virtual cut-through, 75
 wormhole, 75